Learning as Optimization

A Dissertation
Presented to the
Department of Computer Science
Brigham Young University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Kevin S. Van Horn
August 1994

This dissertation by Kevin S. Van Horn is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Tony Martinez, Committee Chairman

Douglas Campbell, Committee Member

William Barrett, Committee Member

Date

David Embley, Graduate Coordinator

Acknowledgments

This research was supported by grants from Wordperfect and Novell, and by the financial assistance of the BYU Computer Science Department. My thanks go to them for helping me to keep my family fed.

Thanks are also due to Dr. Douglas Campbell for teaching me more about technical writing than I have learned from anyone else; Dr. William Barrett for the insights and perspective gained from his Pattern Recognition class; and Dr. Tony Martinez for his support and encouragement as an advisor.

Finally, I would like to thank several people whose encouragement was crucial: Dr. James Kajiya, whose expression of confidence in me as a graduate student at Caltech stayed with me through the years; Dr. Alain Martin, whose positive remarks encouraged me when I decided to return to school; and most importantly my wife Emma, who believed in me when I did not believe in myself, persuaded me to go back to school even though it would mean economic hardship, and has provided invaluable moral support throughout my time at BYU.

Contents

Learning as Optimization

Kevin S. Van Horn
Department of Computer Science
Ph.D. Degree, August 1994

ABSTRACT

This dissertation is concerned with inductive learning from examples, and the reduction of learning problems to associated optimization problems. The emphasis is on learning to classify. Theoretical results include (1) examining the application of Occam's Razor in a general learning setting; (2) investigating two optimization problems associated with learning using linear threshold functions, and showing it to be NP-hard to even approximate them to within a constant factor; and (4) a comparison of the expressive power of decision trees and rule lists.

Practical results include a number of methods of randomly generating learning problems on which to compare learning algorithms, and a new rule induction algorithm called BBG. The problem generators allow one to see how a learning algorithm's performance varies as various parameters such as number of examples, size of target hypothesis, or noise level are varied. BBG learns rule lists by combining the greedy choice of the best new rule to insert into the current rule list with a branch-and-bound algorithm to find this best new rule.

Chapter 1

Introduction

This dissertation is concerned with machine learning, in particular inductive learning from examples. Roughly speaking, the problem of inductive learning is the following: given a collection of *examples* (x, y) drawn from some probability distribution $\mathcal{D}$ over $X \times Y$, construct a function $h : X \to Y$ that predicts well the value of y given x when (x, y) is drawn from $\mathcal{D}$. (As we will see in Chapter 3, the problem of inductive learning can be phrased in even more general terms than this.)

During the last decade the field of computational learning theory has developed considerably, providing guidelines for inductive learning and worst-case analyses of learning algorithms. This work has shown how inductive learning problems can be reduced to problems of combinatorial optimization. This provides the pervading theme of this dissertation: reducing learning problems to optimization problems, analyzing those optimization problems, and developing heuristic approaches to the required (approximate) optimization.

Chapter 2 provides the necessary theoretical background and introduction to computational learning theory, emphasizing the PAC model. An important concept here is the notion of the *VC dimension* of a hypothesis space; a smaller VC dimension gives a smaller bound on the number of training examples needed to achieve a given level of error in learning. Among other things, Chapter 2 discusses the notion of an Occam algorithm, which is an application of the Occam's Razor principle to the PAC model. Most of these results are due to Blumer et al. [6].

Chapter 3 discusses Haussler's extension of the PAC model. Haussler's model is a very general learning model covering many interesting learning situations not covered by the PAC model. Chapter 3 then gives my results on applying a version of Occam's Razor to Haussler's model, producing results similar to those already obtained for Occam Algorithms under the more restricted PAC model. These results make use of what I call loose empirical risk minimization (ERM) algorithms. An alternative approach would be to use what I call loose Occam algorithms, which are a generalization of Occam algorithms. The last part of the chapter shows that, under some weak conditions, for every polynomial-time loose ERM algorithm there is an equivalent polynomial-time loose Occam algorithm, and vice versa.

Chapter 4 looks at reducing the number of examples needed for learning a linear threshold function (a.k.a. perceptron or single-layer neural net) by trying to minimize the number of non-zero weights used. It examines two relevant issues. First a bound is given on the VC dimension of the set of linear-threshold functions that have non-zero weights for at most s of n inputs. Second, it is shown that the problem of minimizing the number of non-zero input weights used (without misclassifying any training examples) is not only NP-hard, it cannot even be approximinated to within any constant factor unless P = NP.

Chapter 5 discusses a minimization problem related to learning linear threshold functions in the presence of noise, or learning linear-threshold approximations to arbitrary functions; among other things, it is shown that this minimization problem, and several weakened versions of it, cannot be approximated to within any constant factor unless P = NP. Thus the learning problem of approaching the minimum error achievable with a linear threshold function cannot be approximated to within any constant factor c in polynomial time, no matter how large c is chosen, unless P = NP.

Chapter 6 discusses decision trees and rule lists, quantifying the oft-observed expressive advantage of rule lists. It also discusses methods for randomly generating learning problems whose underlying structure is best described by a decision tree or rule list. The advantage of such synthetic problems is that we can see how the average misclassification error of a learning algorithm varies as we vary parameters such as the number of examples, noise level, or size of the defining decision tree or rule list.

Chapter 7 discusses a heuristic approach to learning with rule lists, called BBG. BBG combines the greedy choice of the best new rule to insert into the current rule list with a branch-and-bound algorithm to find this best new rule. BBG is compared with the C4.5 and CN2 rule induction algorithms both on data sets drawn from the UC Irvine Repository of Machine-Learning Databases, and on data sets generated using the problem generators of Chapter 6. BBG is found to have a marked advantage for problems that require the full expressive power of rule lists and have even a little noise, and to be competitive with the best of C4.5 and CN2 in most other circumstances.

Chapter 8 reviews the results of this dissertation and outlines directions for further research.

Chapter 2

Theoretical Background

This chapter presents the basics of computational learning theory, as applied to inductive learning from examples. This material will be used in Chapters 3, 4, and 5.

2.1 Review of Computational Complexity

This section defines some complexity classes which we will make use of later. It is assumed that the reader already has some familiarity with the basic notions of computational complexity theory, and merely needs some review and an introduction to some of the less commonly-used complexity classes. The material in this section is taken from [13] and [28].

Definition 2.1 A language L (set of strings over some finite alphabet) is *computed* by an algorithm A if, on input x, A outputs **true** when $x \in L$ and **false** when $x \notin L$.

Definition 2.2 We say that an algorithm *runs in polynomial time* if its worst-case execution time is bounded by a polynomial l^d in the size l of the input. (The size of an input x is the length of the string of symbols representing x.) P is the set of all languages computable by a polynomial-time algorithm.

If $L \notin$ P then, although L may be computable in principle, it is not practically computable, as the execution time increases too rapidly with the size of the input.

Definition 2.3 RP is the set of all languages L computable by a probabilistic algorithm A with the following properties:

- If $x \in L$ then $\mathbf{Pr}[A(x) = \mathbf{true}] \geq 1/2$.
- If $x \notin L$ then $A(x) = \mathbf{false}$.
- The worst-case execution time of $A(x)$ is bounded by a polynomial in the size of x.

(A probabilistic algorithm is allowed to make random choices.)

Although A above has only a probability $1/2$ of getting the right answer when $x \in L$, we can gain greatly increased reliability at little additional cost by simply running A multiple times. In particular, let $A'(x,n)$ be the algorithm that runs A on x for n times, then outputs **true** if $A(x)$ ever returns **true** and **false** if $A(x)$ always returns **false**. Then A' has the following properties:

- If $x \in L$ then $\mathbf{Pr}[A'(x,n) = \mathbf{true}] > 1 - (1/2)^n$.
- If $x \notin L$ then $A'(x,n) = \mathbf{false}$.
- The worst-case execution time of $A'(x,n)$ is bounded by a polynomial in n and the size of x.

Thus membership in RP is almost as good as membership in P.

Definition 2.4 If $T(n)$ is a function from the positive integers to the positive integers, then DTIME$[T(n)]$ is the set of all languages computable by an algorithm whose worst-case execution time is $O(T(n))$, where n is the size of the input.

For example, if $L \in$ P then $L \in$ DTIME$[n^k]$ for some k.

Definition 2.5 NP is the set of all languages verifiable in polynomial time. That is to say, $L \in$ NP iff there exists a total function v with the following properties:

- If $x \in L$ then there exists some y such that $v(x,y) = \mathbf{true}$.
- If $x \notin L$ then there does not exist any y such that $v(x,y) = \mathbf{true}$.
- $v(x,y)$ is computable by an algorithm whose worst-case execution time is bounded by a polynomial in the size of x only.

It is easily shown that P $\subseteq$ RP $\subseteq$ NP. In decreasing order of confidence, it is generally believed that P $\neq$ NP, RP $\neq$ NP, and P $\neq$ RP, but none of these has been proven. Nevertheless, if one can show that some computational problem cannot be solved in polynomial time unless P = NP (or RP = NP), this is generally considered very good evidence that the problem is intractable.

Definition 2.6 Let L_1 and L_2 be two languages. A *polynomial transformation* from L_1 to L_2 is a function τ with the following properties:

- There is a polynomial-time algorithm for computing τ.
- For all x, $x \in L_1$ iff $\tau(x) \in L_2$.

Suppose there is a polynomial transformation from L_1 to L_2. Then $L_2 \in$ P implies $L_1 \in$ P, and $L_2 \in$ RP implies $L_1 \in$ RP, since one can determine if $x \in L_1$ by first computing $\tau(x)$ and then running the polynomial-time algorithm for L_2 on the result.

Definition 2.7 A language L is NP-*complete* if $L \in$ NP and for every $L' \in$ NP there is a polynomial transformation from L' to L.

If any NP-complete language can be computed in polynomial time, then *all* languages in NP can be computed in polynomial time, i.e. P = NP. Thus if we can show that a language is NP-complete, this is considered strong evidence that it cannot be computed in polynomial time. The usual way of proving that a language L is NP-complete is to construct a polynomial transformation from some known NP-complete problem to L.

Definition 2.8 A computational problem is said to be NP-*hard* if any polynomial-time algorithm for solving it can be used as a subroutine to obtain a polynomial-time algorithm for computing an NP-complete language.

For brevity's sake the above definition is a bit vague, but it will suffice for our purposes. The important point is that an NP-hard problem cannot be solved in polynomial time unless P = NP. Note the difference between NP-complete and NP-hard: an NP-complete problem has only yes/no answers (is the string in the specified language?) and must belong to NP; an NP-hard problem, on the other hand, need not have only yes/no answers and need not belong to NP, as long as it is *at least* as hard to solve as computing an NP-complete language.

2.2 The PAC Model

We now discuss the PAC model of learning. The PAC (probably approximately correct) model of learning [6, 23, 32, 39] was first introduced by Valiant [39]. There are a number of variants of the model which have been proven to be essentially equivalent [19]; the variant presented here is a slightly simplified version of the one presented in [6]. The problem addressed by the PAC model is that of learning a Boolean function.

2.2.1 The Unstratified PAC Model

The simplest form of the PAC model we call the "unstratified" PAC model, for reasons that will be clear later. Its elements are the following:

- An *instance space* X.
- A *hypothesis space* H over X. H is a set of Boolean functions with domain X.
- An unknown *target function* $f \in H$ which is to be learned.
- An unknown probability distribution D governing the frequency of occurrence of elements of X.
- A sequence of *training examples* $z_i = (x_i, f(x_i))$, where the x_i are randomly and independently chosen according to D. This sequence is called the *training sample.*

An algorithm A for learning H takes the training sample $\mathbf{z}$ as input, and outputs a hypothesis $h \in H$. Note that since $\mathbf{z}$ is randomly chosen, the output hypothesis $h = A(\mathbf{z})$ is in general defined only probabilistically, even if A is deterministic. Thus we do not require A to be deterministic; instead we allow A to make random choices.

Definition 2.9 The *error* of a hypothesis $h \in H$, denoted $\mathrm{err}(h)$, is defined as the probability that $h(x) \neq f(x)$ (h misclassifies x) when x is drawn at random from the distribution D.

Definition 2.10 The *sample complexity* $\mathcal{S}(\epsilon, \delta)$ of an algorithm for learning H is the worst-case number of examples needed to have probability $1-\delta$ or better of returning a hypothesis with error at most ϵ, when f may be any element of H and D may be any distribution over X.

It is often the case that the instance space X is implicitly parameterized; thus we actually have a family of instance spaces $X^{[n]}$ and associated hypothesis spaces $H^{[n]}$, indexed by n. For example, we might have $X^{[n]} = \{0,1\}^n$ and $H^{[n]}$ defined as the set of all linear threshold functions on $X^{[n]}$. In this case the sample complexity of a learning algorithm should also be considered a function of n. We will often leave such a parameterization implicit to avoid cumbersome notation.

A practical learning algorithm must have a sample complexity that does not grow too quickly with ϵ^{-1}, δ^{-1}, and n, and its execution time must not grow too quickly with the size of its input. This motivates the following definition.

Definition 2.11 A learning algorithm which runs in polynomial time, and whose sample complexity is bounded by a polynomial in ϵ^{-1}, δ^{-1}, and n, is called a *polynomial algorithm for learning H*.

2.2.2 A Strategy for Unstratified PAC Learning

Definition 2.12 A hypothesis $h \in H$ is *consistent* with a training sample $\mathbf{z}$ if $h(x_i) = y_i$ for all examples $z_i = (x_i, y_i)$. We say that a learning algorithm *uses H consistently* if it returns a hypothesis $h \in H$ that is consistent with the training sample given it, whenever such an h exists.

Under certain circumstances a learning algorithm can achieve an acceptable sample complexity simply by using H consistently. This happens when H has a small enough *VC dimension*, a notion that we now define.

Definition 2.13 Let H be a hypothesis space over X. For any finite $I \subseteq X$ we define

$$\Pi_H(I) = \{h \cap I : h \in H\},$$

where we regard the elements of H as subsets of X. Note that each $h \in H$ defines a dichotomy of I, partitioning it into $I \cap h$ and $I - I \cap h$. $\Pi_H(I)$ can then be thought of as the set of such dichotomies on I defined by elements of H. For any integer m, $1 \leq m \leq |X|$, we define

$$\Pi_H(m) = \max_{I \subseteq X, |I|=m} |\Pi_H(I)|.$$

$\Pi_H(m)$ is just the maximum number of ways a set of m elements of X can be dichotomized by elements of H.

Note that $\Pi_H(m) \leq 2^m$, since an m-element set has only 2^m distinct subsets.

Definition 2.14 Let H be a hypothesis space over X. The *Vapnik-Chervonenkis dimension* of H, denoted $\mathrm{VCdim}(H)$, is the largest integer m s.t. $\Pi_H(m) = 2^m$. In other words, $\mathrm{VCdim}(H)$ is the cardinality of the largest subset of X which can be arbitrarily dichotomized by elements of H.

Note that the VC dimension of any finite hypothesis space H is at most $\log_2 |H|$. When this bound is tight, the VC dimension of H may be viewed as the number of bits required to specify an arbitrary hypothesis in H. Other hypothesis spaces are infinite but specified by a finite number of real-valued parameters, and their VC dimension is related to the number of parameters. For example, the set of all linear threshold functions of n inputs has VC dimension $n+1$ [6].

If the VC dimension of $H^{[n]}$ grows polynomially in n, then any polynomial-time learning algorithm that uses H consistently is a polynomial algorithm for learning H. In particular, we have the following theorem [6]:

Theorem 2.1 Let hypothesis space H have finite VC dimension. Then any learning algorithm that uses H consistently has sample complexity

$$O(\epsilon^{-1}(\mathrm{VCdim}(H) \log \epsilon^{-1} + \log \delta^{-1})).$$

It may be that there is no polynomial algorithm for learning H, even if $\mathrm{VCdim}(H)$ is finite and is bounded by a polynomial in n. This can happen if the problem of finding a consistent hypothesis is intractable.

The following result is shown in [24, 34]:

Definition 2.15 The *consistency problem* for H is the following:
Input: A sequence $\mathbf{z}$ of examples.
Question: Does there exist a hypothesis $h \in H$ that is consistent with $\mathbf{z}$?

Theorem 2.2 If RP $\neq$ NP and the consistency problem for H is NP-hard, then there is no polynomial algorithm for learning H.

2.2.3 The Stratified PAC Model

Theorem 2.1 suggests that the larger the VC dimension of a hypothesis space, the harder it is to learn. In many interesting cases the VC dimension of H is either infinite or excessively large. To handle this problem we resort to a *stratified* hypothesis space.

Definition 2.16 A *stratified hypothesis space* on X is a (possibly infinite) sequence of hypothesis spaces $(H_i)_{i \geq 1}$ such that $H_i \subseteq H_{i+1}$ for all i. We write H for either $\bigcup_i H_i$ or $(H_i)_{i \geq 1}$, and rely on context to disambiguate the usage.

The purpose of stratifying the hypothesis space is to introduce a simplicity bias into our learning methods. Work in machine learning often makes use of the Occam's Razor principle: given two explanations of the data, all other things being equal, the simpler of the two is preferable. Occam's Razor can be applied once we have a measure of the complexity of a hypothesis. Such a measure is obtained by choosing some means of representing hypotheses, and defining the complexity of a hypothesis in terms of the size of its smallest representation. For example, choosing $X = \{0,1\}^n$ we could let H be the set of *all* Boolean functions on X, and define the size of $h \in H$ to be the number of literals in the smallest sum-of-products formula for h. We can then define H_i to be the set of $h \in H$ of size at most i.

If we knew in advance the size s of the target function, we could simply apply the method of Section 2.2.2 using H_s for the hypothesis space H of that section. The sample-complexity bound thus obtained depends on $\mathrm{VCdim}(H_s)$, which increases as s increases. Thus we modify the definition of sample complexity and of a polynomial learning algorithm to take the size of the target into account.

Definition 2.17 The *sample complexity* $\mathcal{S}(\epsilon, \delta, s)$ of an algorithm for learning a *stratified* hypothesis space H is the worst-case number of examples needed to have probability $1 - \delta$ or better of returning a hypothesis with error at most ϵ, when f may be any element of H_s and D may be any distribution over X.

As before, if the instance space is implicitly parameterized by a variable n, we will consider the sample complexity to be also a function of n.

Definition 2.18 Let H be a stratified hypothesis space. A learning algorithm for H which runs in polynomial time, and whose sample complexity is bounded by a polynomial in ϵ^{-1}, δ^{-1}, s, and n, is called a *polynomial algorithm for learning H.*

Blumer et al. [6] show that learning with a stratified hypothesis space can be done by approximately solving the following optimization problem: find the smallest hypothesis that is consistent with the training sample.

Definition 2.19 Let $0 \leq a < 1$ and let $p(s) \geq 1$ be a function bounded from above by a polynomial in s. Let H be a stratified hypothesis space and $b(s,m)$ be some function satisfying $\mathrm{VCdim}(H_{b(s,m)}) \leq p(s)m^a$. Suppose A is an approximation algorithm that is guaranteed to return a hypothesis $h \in H$ of size at most $b(s,m)$, consistent with the m training examples input to A, whenever there exists a consistent hypothesis $h' \in H$ of size s. Then we call A an *Occam algorithm* for H, with bound $p(s)m^a$.

Theorem 2.3 [6] Any Occam algorithm for H with bound $p(s)m^a$ serves as a learning algorithm with sample complexity

$$O\left(\epsilon^{-1}\log\delta^{-1} + (p(s)\epsilon^{-1}\log\epsilon^{-1})^{\frac{1}{1-a}}\right).$$

Note that exact minimization is not needed — even a quite weak approximation algorithm can give us a polynomial algorithm for learning a stratified hypothesis space.

Chapter 3

Extending Occam's Razor

3.1 Introduction

In Section 2.2.3 we discussed the application of Occam's Razor, in the form of Occam algorithms, to learning under the PAC model. The PAC model, however, has a number of limitations. It assumes that the problem is to learn the correct classification of instances, and that some member of the given hypothesis space correctly classifies all instances. This rules out problems such as learning real-valued functions, probability distributions, class probability distributions as a function of the instance to be classified, and the Bayes-optimal classifier in a stochastic setting.

Haussler [18] has generalized the PAC model to deal with these situations and others. In this chapter I prove, for Haussler's model, an analog of Theorem 2.3 (sample complexity for Occam algorithms). The approach I analyze is essentially the "hold-out" method often used in applied statistics. That is, one tries to minimize the empirical risk (a measure of error on the training sample) with different bounds on the complexity of the hypotheses to be considered, then one uses a separate hold-out set to choose the best complexity bound. As with Theorem 2.3, attention is paid to avoiding exact minimization and its attendant intractability — a limited increase in hypothesis complexity is allowed over what is strictly needed to attain a given level of empirical risk. We obtain sample complexity bounds similar to those of Theorem 2.3, but for Haussler's more general learning model.

The approach analyzed here uses what I have called a *loose empirical risk minimization* algorithm. An alternate approach would be to use what I call a *loose Occam algorithm*, where one tries to minimize hypothesis size with different upper bounds on allowed empirical risk, then uses a separate hold-out set to choose the best risk bound. The chapter concludes by showing that, under some reasonable assumptions, the two approaches have equivalent computational complexities.

3.2 Haussler's Generalization of the PAC Model

Haussler [18] has extended the PAC model to handle more general learning situations. The elements of his model are the following:

- An *instance space* X.
- An *outcome space* Y.
- A *decision space* Y'.
- A *hypothesis space* H of functions $h : X \to Y$.
- An *assumption space* $\mathcal{A}$ of probability distributions over $X \times Y$.
- An unknown *target* $D \in \mathcal{A}$ which determines the frequency of occurrence of instances $x \in X$ and their outcomes $y \in Y$.
- A sequence of *training examples* $z_i = (x_i, y_i)$, where the z_i are randomly and independently chosen according to D. This sequence is called the *training sample.*
- A *loss function* $l : Y' \times Y \to [0, M]$ (for some $M > 0$). $l(y', y)$ is the loss incurred when a hypothesis outputs y' for an instance x whose outcome is y.

The PAC model can be considered a special case of this model, with $Y = Y' = \{0, 1\}$, $l(y', y) = 1$ if $y' \neq y$ and 0 otherwise, and $\mathcal{A}$ being the set of all distributions over $X \times Y$ satisfying $\mathbf{Pr}[y = f(x)] = 1$ for some $f \in H$.

Definition 3.1 The *risk* $\mathbf{r}(h)$ of a hypothesis $h \in H$ is the expected value of $l(h(x), y)$ when (x, y) is drawn at random from the distribution D.

A learning algorithm takes training examples drawn from this same distribution D. The goal of learning is to find a hypothesis whose error is close to the minimum possible for hypotheses from H.

Haussler defines sample complexity in terms of the following family of metrics on the real numbers:

Definition 3.2 For any real $\nu > 0$, d_ν is a measure of relative distance defined by

$$d_\nu(r, s) = \frac{|r - s|}{\nu + r + s}$$

for any $r, s \geq 0$.

Note that the condition $d_\nu(r, s) \leq \alpha$ is equivalent to

$$\left(\frac{1 - \alpha}{1 + \alpha}\right) r - \frac{\alpha\nu}{1 + \alpha} \leq s \leq \left(\frac{1 + \alpha}{1 - \alpha}\right) r + \frac{\alpha\nu}{1 - \alpha};$$

for small α this says that s differs from r by at most a multiplicative factor of about $1 + 2\alpha$ and an additive term of $\alpha\nu$.

Definition 3.3 The sample complexity $\mathcal{S}(\alpha, \nu, \delta)$ of a learning algorithm is the worst-case number of examples needed to have probability $1 - \delta$ or better of returning a hypothesis $h \in H$ satisfying

$$d_\nu(\mathbf{r}(h), \inf\{\mathbf{r}(h') : h' \in H\}) \leq \alpha,$$

when the target distribution may be any $D \in \mathcal{A}$.

The literature contains no analog, for Haussler's model, of the Occam algorithm result of Theorem 2.3. (Nor does it contain such an analog for the simpler PAC extensions of simply allowing stochastic targets or targets not in the hypothesis space.) There are, however, useful sample complexity results for the case that H has a finite *pseudodimension*. The pseudodimension of H, denoted $\mathrm{pdim}(H)$, depends on the loss function l and may be considered a generalization of the VC dimension; in fact the two are equal if the hypotheses of H are Boolean-valued and $l(y', y) = (y' \neq y)$. (Note: we abuse terminology by writing $\mathrm{pdim}(H)$ when we really mean $\mathrm{pdim}\{f_h : h \in H\}$, where $f_h(x, y) = l(h(x), y)$.) Details on the pseudodimension may be found in [18].

Definition 3.4 Given a sequence of examples $\mathbf{z} = (x_1, y_1), \ldots, (x_m, y_m)$, the *empirical risk* $\hat{\mathbf{r}}(h; \mathbf{z})$ of a hypothesis h is the average loss of h on $\mathbf{z}$, i.e. $\hat{\mathbf{r}}(h; \mathbf{z}) \overset{\text{def}}{=} \frac{1}{m} \sum_{i=1}^{m} l(h(x_i), y_i)$. We write $\hat{\mathbf{r}}(h)$ when $\mathbf{z}$ is understood.

A learning algorithm is said to use *empirical risk minimization* [18, 42, 43] if it returns a hypothesis whose empirical risk is the minimum possible for hypotheses from H. Combining Lemma 1 and Theorem 7 of Haussler [18] gives this result: any learning algorithm that uses empirical risk minimization has a sample complexity that is

$$O((\alpha^2\nu)^{-1}(\mathrm{pdim}(H) \log(\alpha\nu)^{-1} + \log \delta^{-1})). \tag{3.1}$$

Note the similarity between this bound and that of Theorem 2.1: they are identical except that $\mathrm{pdim}(H)$ replaces $\mathrm{VCdim}(H)$ and $\alpha^2\nu$ replaces ϵ.

3.3 Summary of Sample-Complexity Results

We begin with some definitions.

Definition 3.5 A *dimension-stratified hypothesis space* is a (possibly infinite) sequence of hypothesis spaces $(H_i)_{i \geq 1}$ such that $H_i \subseteq H_{i+1}$ and $\mathrm{pdim}(H_i) \leq i$ for all i. We abuse notation and write H for either $(H_i)_{i \geq 1}$ or $\bigcup_i H_i$. We define the *size* of $h \in H$, denoted $\mathrm{siz}(h)$, to be $\min\{i : h \in H_i\}$.

Definition 3.6 Let $p(s) \geq 1$ be monotonic and polynomially bounded in s, $0 \leq a < 1$, and H be a dimension-stratified hypothesis space. We say that A is a *loose ERM (empirical risk minimization) algorithm* for H, with bound $p(s)m^a$, if

- A takes as input a sequence of examples $\mathbf{z}$ and an integer *size bound* i satisfying $1 \leq i \leq m \overset{\text{def}}{=} |\mathbf{z}|$;

- If $p(j)m^a < i + 1$ for some j s.t. $H_j \neq \emptyset$, then A outputs a hypothesis $h \in H_i$ satisfying $\hat{\mathbf{r}}(h; \mathbf{z}) \leq \hat{\mathbf{r}}(h'; \mathbf{z})$ for all $h' \in H$ s.t. $p(\mathrm{siz}(h'))m^a < i + 1$.

Thus a loose ERM algorithm loosens the requirements of empirical risk minimization in hopes of making the problem tractable. Associated with each loose ERM algorithm for H is a learning algorithm for H:

Definition 3.7 Let H be a dimension-stratified hypothesis space, A be a loose ERM algorithm for H, $0 < c < 1$, and $0 < \rho \leq 1$; then $\mathrm{HO}[A, c, \rho]$ is the following algorithm:

1. Input a sequence of examples $\mathbf{z}$.
2. Split $\mathbf{z}$ into two sequences: $\mathbf{z}_1$ (the *training sample*), containing the first $\lfloor |\mathbf{z}|/(1 + \rho) \rfloor$ elements of $\mathbf{z}$, and $\mathbf{z}_2$ (the *test sample*), containing the remaining elements of $\mathbf{z}$. Let $m \overset{\text{def}}{=} |\mathbf{z}_1|$.
3. For all $1 \leq i \leq \lfloor \log m / \log(1/c) \rfloor$, compute $h_i \overset{\text{def}}{=} A(\mathbf{z}_1, \lfloor mc^i \rfloor)$.
4. Output that h_i minimizing $\hat{\mathbf{r}}(h_i; \mathbf{z}_2)$.

The above is essentially the "hold-out" method often used in applied statistics (see [12], for example). Our contribution is to analyze the sample complexity of $\mathrm{HO}[A, c, \rho]$ given the weakened requirement that A be a loose ERM algorithm (as opposed to doing strict empirical risk minimization), and show that we get sample complexity bounds analogous to those of Theorem 2.3.

We modify Haussler's definition of sample complexity to take into account hypothesis complexity by adding a size parameter s. We also need to take into account the possibility that the hypothesis returned has size greater than s, and hence might have a risk less than the minimum achievable by hypotheses from H_s.

Definition 3.8 $d'_\nu(r_1, r_2) \overset{\text{def}}{=} d_\nu(r_1, \max\{r_1, r_2\})$.

Definition 3.9 Let H be a dimension-stratified hypothesis space. The sample complexity $\mathcal{S}(\alpha, \nu, \delta, s)$ of a learning algorithm for H is the worst-case number of examples needed to have probability $1 - \delta$ or better of returning a hypothesis h satisfying

$$d'_\nu(\inf\{\mathbf{r}(h') : h' \in H_s\}, \mathbf{r}(h)) \leq \alpha$$

when the target distribution may be any $D \in \mathcal{D}$.

Our result is that if H is a dimension-stratified hypothesis space and A is a loose ERM algorithm for H, with bound $p(s)m^a$, then the learning algorithm $\mathrm{HO}[A, c, \rho]$ has sample complexity

$$O((\alpha^2\nu)^{-1} \log \delta^{-1} + ((\alpha^2\nu)^{-1} p(s) \log(\alpha\nu)^{-1})^{\frac{1}{1-a}}). \tag{3.2}$$

Comparing this bound to that of Theorem 2.3, we see that they are the same except that $\alpha^2\nu$ replaces ϵ and the pseudodimension replaces the VC dimension.

3.4 Proof of Sample-Complexity Result

Our proof will require some theorems from Haussler [18] and two lemmas. We will assume throughout that M is a bound on the loss function for whatever hypothesis space H is being discussed. The theorems from Haussler give sample complexity bounds for hypothesis spaces of finite cardinality or finite pseudodimension.

Theorem 3.1 (Theorem 1 of [18].) Let H be a finite set of hypotheses; let a sequence $\mathbf{z}$ of m examples be drawn randomly and independently from the distribution D; and let $\nu > 0$ and $0 < \alpha < 1$. Then

$$\mathbf{Pr}[\exists h \in H.\ d_\nu(\hat{\mathbf{r}}(h;\mathbf{z}), \mathbf{r}(h)) > \alpha] \le 2|H| \exp(-\alpha^2 \nu m / M).$$

For $\delta > 0$ and $m \ge M(\alpha^2\nu)^{-1}(\ln|H| + \ln(2/\delta))$, this probability is at most δ.

Theorem 3.2 (Theorem 7 of [18].) Let H be a set of hypotheses with $\mathrm{pdim}(H) = s$ (s finite); let a sequence $\mathbf{z}$ of $m \ge 1$ examples be drawn randomly and independently from the distribution D; and let $0 < \nu \le 8M$ and $0 < \alpha < 1$. Then

$$\mathbf{Pr}[\exists h \in H.\ d_\nu(\hat{\mathbf{r}}(h;\mathbf{z}), \mathbf{r}(h)) > \alpha] \le 8\left(\frac{16eM}{\alpha\nu} \ln \frac{16eM}{\alpha\nu}\right)^s \exp\left(-\frac{\alpha^2 \nu m}{8M}\right),$$

where e is the base of the natural logarithm.

The following lemma is used to bound the risk of the hypothesis output by a loose ERM algorithm.

Definition 3.10 We write $\mathrm{lln}\,(x)$ for $\ln(x \ln x)$. Note that $\mathrm{lln}\,(x) = O(\ln x)$.

Lemma 3.3 Let m be a positive integer and $0 \le a < 1$; let H be a set of hypotheses with $\mathrm{pdim}(H) \le sm^a$; let a sequence $\mathbf{z}$ of m examples be drawn randomly and independently from the distribution D; and let $0 < \alpha < 1$, $\delta > 0$ and $0 < \nu \le 8M$. Then

$$\mathbf{Pr}[\exists h \in H.\ d_\nu(\hat{\mathbf{r}}(h;\mathbf{z}), \mathbf{r}(h)) > \alpha] \le \delta$$

whenever

$$m \ge \max\left\{ \left(\frac{16M}{\alpha^2\nu} s \,\mathrm{lln}\, \frac{16eM}{\alpha\nu}\right)^{\frac{1}{1-a}},\ \frac{16M}{\alpha^2\nu} \ln \frac{8}{\delta} \right\}. \tag{3.3}$$

Proof. By Theorem 3.2 it suffices to show that

$$8\left(\frac{16eM}{\alpha\nu} \ln \frac{16eM}{\alpha\nu}\right)^{sm^a} \exp\left(-\frac{\alpha^2\nu m}{8M}\right) \le \delta$$

when the bound (3.3) on m holds. Taking logarithms of both sides of the above inequality and rearranging yields

$$m \ge \frac{8M}{\alpha^2\nu}\left(sm^a \,\mathrm{lln}\, \frac{16eM}{\alpha\nu} + \ln\frac{8}{\delta}\right). \tag{3.4}$$

From the bound (3.3) on m we have that

$$\frac{m}{2} \geq \frac{8M}{\alpha^2\nu} \ln \frac{8}{\delta};$$

thus (3.4) will be satisfied if

$$\frac{m}{2} \geq Bm^a \quad \text{where} \quad B \stackrel{\text{def}}{=} \frac{8M}{\alpha^2\nu} s \,\text{lln}\, \frac{16eM}{\alpha\nu},$$

which can be rewritten as $m^{1-a} \geq 2B$ and then as $m \geq (2B)^{\frac{1}{1-a}}$. This latter inequality follows directly from (3.3). $\square$

The next lemma is used to bound the error incurred in step 4 of HO$[A, c, \rho]$.

Lemma 3.4 Let $\rho, C, m > 0$; let H be a set of $\lfloor C \ln m \rfloor$ hypotheses; let a sequence $\mathbf{z}$ of at least ρm examples be drawn randomly and independently from the distribution D; and let $\alpha^2\nu \leq M/(3\rho)$; then

$$\mathbf{Pr}[\exists h \in H.\ d_\nu(\hat{\mathbf{r}}(h; \mathbf{z}), \mathbf{r}(h)) > \alpha] \leq \delta$$

whenever

$$m \geq \frac{2M}{\alpha^2\nu\rho} \max\left\{0.44 + \ln\ln \frac{2M}{\alpha^2\nu\rho},\ \ln \frac{2C}{\delta}\right\} \tag{3.5}$$

Proof. By Theorem 3.1, the above-mentioned probability will be at most δ if

$$\rho m \geq \frac{M}{\alpha^2\nu}(\ln(C \ln m) + \ln(2/\delta)),$$

which can be rewritten as

$$m \geq \frac{M}{\alpha^2\nu\rho}(\ln\ln m + \ln(2C/\delta)).$$

This inequality in turn will hold if

$$m \geq \frac{2M}{\alpha^2\nu\rho} \ln \frac{2C}{\delta} \tag{3.6}$$

and

$$m \geq B \ln\ln m \quad \text{where} \quad B \stackrel{\text{def}}{=} \frac{2M}{\alpha^2\nu\rho}. \tag{3.7}$$

From the bound (3.5) on m we see that (3.6) holds, so it remains only to show that (3.7) holds. We shall use the fact that $B \geq 6$, since $\alpha^2\nu \leq M/(3\rho)$ from the statement of the lemma.

Let $f(b) \stackrel{\text{def}}{=} b \ln(1.55 \ln b)$. Since $0.44 > \ln 1.55$, we have by (3.5) that $m > f(B)$. In fact, $m = f(b')$ for some $b' > B$, since $f(b)$ is continuous and increasing for $b > 1$, and $f(b) \to \infty$ as $b \to \infty$. Thus (3.7) holds if $f(b) \geq B \ln\ln f(b)$ for all $b \geq B$; this in turn holds if

$$f(b) \geq b \ln\ln f(b) \ \text{ for all } \ b \geq 6 \tag{3.8}$$

Defining $g(b) \overset{\text{def}}{=} b^{0.55}/\ln(1.55 \ln b)$, for all $b \geq 6$ we have

$$f(b) \geq b \ln\ln f(b) \Leftrightarrow 1.55 \ln b \geq \ln(b \ln(1.55 \ln b)) \Leftrightarrow b^{1.55} \geq b \ln(1.55 \ln b) \Leftrightarrow g(b) \geq 1.$$

Writing $\mathrm{sgn}(x)$ for the sign of x and noting that $1.55 \ln b > 1$ for $b \geq 6$, we have that

$$\begin{aligned}\mathrm{sgn}(dg/db) &= \mathrm{sgn}(\ln(1.55 \ln b) \cdot 0.55 b^{-0.45} - b^{0.55} \cdot (1.55 \ln b)^{-1} 1.55 b^{-1}) \\ &= \mathrm{sgn}(s(b)) \ \text{ where } \ s(b) \overset{\text{def}}{=} 0.55 \ln(1.55 \ln b) - (\ln b)^{-1}.\end{aligned}$$

But $s(6) > 0$ and $s(b)$ is an increasing function of b, so $s(b) > 0$ for all $b \geq 6$. Hence $dg/db > 0$ for all $b \geq 6$. But $g(6) \simeq 2.6 > 1$, hence $g(b) > 1$ for all $b \geq 6$. Thus (3.8) holds, which means that (3.7) holds, and the theorem is proven. □

We now mention some properties of d_ν and d'_ν that we use in the proof of Theorem 3.5. Haussler proves the following properties of d_ν:

1. d_ν is a metric on the nonnegative reals, i.e. $d_\nu(r,s) \geq 0$, $d_\nu(r,s) = 0$ iff $r = s$, $d_\nu(r,s) = d_\nu(s,r)$, and $d_\nu(r,t) \leq d_\nu(r,s) + d_\nu(s,t)$ (triangle inequality.)
2. d_ν is compatible with the ordering on the reals, i.e. if $r \leq s \leq t$ then $d_\nu(r,s) \leq d_\nu(r,t)$ and $d_\nu(s,t) \leq d_\nu(r,t)$.

We state without proof the following easily-verified properties of d'_ν:

1. If $r_2 \leq r_1$ then $d'_\nu(r_1, r_2) = 0$.
2. $0 \leq d'_\nu(r_1, r_2) \leq d_\nu(r_1, r_2)$.
3. $d'_\nu(r_1, r_3) \leq d'_\nu(r_1, r_2) + d'_\nu(r_2, r_3)$ (triangle inequality).

Finally, we are ready for the main theorem.

Theorem 3.5 Let H be a dimension-stratified hypothesis space; let S be a positive integer and $r_0 \overset{\text{def}}{=} \inf\{\mathbf{r}(h) : h \in H_S\}$; let A be a loose ERM algorithm for H with bound $p(s)m^a$; let $0.05 \leq c < 1$ and $0 < \rho \leq 1$; let $\mathbf{z}$ be a sequence of m examples drawn randomly and independently from the distribution D; let $h_{out} \overset{\text{def}}{=} \mathrm{HO}[A, c, \rho](\mathbf{z})$; and let $0 < \delta \leq 1$, $0 < \alpha < 1$, and $0 < \nu \leq 8M$. Then

$$\mathbf{Pr}[d'_\nu(r_0, \mathbf{r}(h_{out})) > \alpha] \leq \delta \tag{3.9}$$

whenever $m \geq (1+\rho)(B+1)$, where

$$B = \max\left\{ \frac{55M}{\alpha^2 \nu \rho}\left(0.44 + \ln\ln \frac{55M}{\alpha^2 \nu \rho}\right), \frac{55M}{\alpha^2 \nu \rho} \ln \frac{2(C+6)}{\delta}, \left(\frac{55M}{\alpha^2 \nu c} p(S) \,\mathrm{lln}\, \frac{81M}{\alpha \nu}\right)^{\frac{1}{1-a}} \right\}$$

and $C = 1/\ln(1/c)$.

Before giving the proof, we have a few comments. Recall that HO$[A, c, \rho]$ runs A with various size bounds of the form $\lfloor mc^i \rfloor$, where i goes from 1 to a maximum value. The requirement $0.05 \le c < 1$ merely says that the size bound decreases as i increases, but not by more than a factor of 20 at each step. Recall also that A is run on the first $\lfloor |\mathbf{z}|/(1+\rho) | \rfloor$ examples, with the remaining examples used to choose the best size bound. Thus the requirement $0 < \rho \le 1$ says that at least (about) half — but not all — of the examples are used as input to A. Finally, note that a, c, ρ, and M are constants, $\operatorname{lln}(x) = O(\ln x)$, and

$$\ln \ln \frac{55M}{\alpha^2 \nu \rho} = O\left(\ln \frac{1}{\alpha \nu}\right),$$

so that $B = O((\alpha^2\nu)^{-1} \ln \delta^{-1} + ((\alpha^2\nu)^{-1} p(S) \ln(\alpha\nu)^{-1})^{\frac{1}{1-a}})$. (Compare with (3.2).)

Proof. If $H_S = \emptyset$ then $r_0 = M$, and the theorem trivially holds. So we assume that $H_S \ne \emptyset$. HO$[A, c, \rho]$ is run with $m_* = \lfloor (1+\rho)^{-1} m \rfloor$ training examples and $m_{**} = m - m_*$ test examples. Since $m \ge (1+\rho)(B+1)$, we have that

$$m_* \ge \lfloor (1+\rho)^{-1}(1+\rho)(B+1) \rfloor = \lfloor B+1 \rfloor > B$$

and

$$m_{**} = m - m_* \ge m - \frac{m}{1+\rho} = \frac{\rho}{1+\rho} m \ge \rho \lfloor (1+\rho)^{-1} m \rfloor = \rho m_*.$$

Thus it will suffice to show that (3.9) holds whenever steps 3 and 4 of HO$[A, c, \rho]$ are run with $m_* > B$ training examples and $m_{**} \ge \rho m_*$ test examples. We will write $\mathbf{z}_1$ for the training examples and $\mathbf{z}_2$ for the test examples.

Viewing B as a function of α, we have that $B(\alpha)$ is continuous and decreasing in α, and $B(\alpha) \to \infty$ as $\alpha \to 0$; hence $m_* > B(\alpha)$ implies that $m_* = B(\alpha_*)$ for some $\alpha_* < \alpha$. By the definition of r_0 as an infimum, there are hypotheses $h \in H_S$ with risk arbitrarily close to r_0; in particular, there exists some $h_* \in H_S$ with $d'_\nu(r_0, \mathbf{r}(h_*)) \le \alpha - \alpha_*$. By the triangle inequality for d'_ν, it then suffices to show that

$$\mathbf{Pr}[d'_\nu(\mathbf{r}(h_*), \mathbf{r}(h_{out})) > \alpha_*] \le \delta \tag{3.10}$$

whenever steps 3 and 4 of HO$[A, c, \rho]$ are run with $m_* = B(\alpha_*)$ training examples and $m_{**} \ge \rho m_*$ test examples.

It will be useful to have a simpler lower bound on m_*. By the requirements that $\nu \le 8M$, $\alpha < 1$ and $c < 1$ we have that $55M/(\alpha_*^2 \nu c) > 55/8$ and $\operatorname{lln}(81M/(\alpha\nu)) > \operatorname{lln}(81/8)$; hence using $m_* = B(\alpha_*)$ we have

$$m_* > \left(\frac{55}{8} p(S) \operatorname{lln} \frac{81}{8}\right)^{\frac{1}{1-a}} > (21.6\, p(S))^{\frac{1}{1-a}} \tag{3.11}$$

Let us define the following:

- $h_i \overset{\text{def}}{=} A(\mathbf{z}_1, \lfloor m_* c^i \rfloor)$ for all $1 \le i \le \lfloor C \ln m_* \rfloor$, as in the definition of HO$[A, c, \rho]$.
- $j \overset{\text{def}}{=} \max\{i \in \mathcal{Z} : m_* c^i \ge p(S) m_*^a\}$.

- $S' \overset{\text{def}}{=} \lfloor m_* c^j \rfloor$.

We wish to ensure that $1 \leq j \leq \lfloor C \ln m_* \rfloor$, so that $h_j = A(\mathbf{z}_1, S')$ will be one of the hypotheses considered in step 4 of HO$[A, c, \rho]$. We will have $j \geq 1$ if $m_* c \geq p(S) m_*^a$, i.e. $c \geq p(S) m_*^{a-1}$. From (3.11) and the fact that $a < 1$ we have that

$$p(S) m_*^{a-1} < p(S)(21.6\, p(S))^{\frac{a-1}{1-a}} = 21.6^{-1};$$

but $c \geq 0.05$ (from the statement of the theorem) $> 21.6^{-1} > p(S) m_*^{a-1}$, so $j \geq 1$. For the upper bound on j, note that since $m_* c^j \geq p(S) m_*^a$, we have

$$j \leq \ln(p(S) m_*^{a-1}) / \ln c = \ln(p(S)^{-1} m_*^{1-a}) / \ln(1/c) \leq \ln m_* / \ln(1/c)$$

(the last inequality uses $p(S) \geq 1$, $m_* > 1$, and $a \geq 0$); but since j is an integer we have

$$j \leq \lfloor \ln m_* / \ln(1/c) \rfloor = \lfloor C \ln m_* \rfloor.$$

By the triangle inequality for d'_ν, we will have $d'_\nu(\mathbf{r}(h_*), \mathbf{r}(h_{out})) \leq \alpha_*$ if the following hold for appropriate positive values a_i summing to 1:

1. $d'_\nu(\mathbf{r}(h_*), \hat{\mathbf{r}}(h_*; \mathbf{z}_1)) \leq a_1 \alpha_*$;
2. $d'_\nu(\hat{\mathbf{r}}(h_*; \mathbf{z}_1), \hat{\mathbf{r}}(h_j; \mathbf{z}_1)) = 0$;
3. $d'_\nu(\hat{\mathbf{r}}(h_j; \mathbf{z}_1), \mathbf{r}(h_j)) \leq a_2 \alpha_*$;
4. $d'_\nu(\mathbf{r}(h_j), \hat{\mathbf{r}}(h_j; \mathbf{z}_2)) \leq a_3 \alpha_*$;
5. $d'_\nu(\hat{\mathbf{r}}(h_j; \mathbf{z}_2), \hat{\mathbf{r}}(h_{out}; \mathbf{z}_2)) = 0$;
6. $d'_\nu(\hat{\mathbf{r}}(h_{out}; \mathbf{z}_2), \mathbf{r}(h_{out})) \leq a_4 \alpha_*$.

Thus we prove (3.10) by showing that conditions 2 and 5 always hold, and that the following hold for appropriate positive values f_i summing to 1:

$$\begin{array}{rcl} \mathbf{Pr}[\text{Condition 1 does not hold}] & \leq & f_1 \delta \\ \mathbf{Pr}[\text{Condition 3 does not hold}] & \leq & f_2 \delta \\ \mathbf{Pr}[\text{Condition 4 does not hold}] & \leq & f_3 \delta \\ \mathbf{Pr}[\text{Condition 6 does not hold}] & \leq & f_4 \delta. \end{array} \tag{3.12}$$

By definition, $\hat{\mathbf{r}}(h_{out}; \mathbf{z}_2) \leq \hat{\mathbf{r}}(h_j; \mathbf{z}_2)$, so condition 5 holds. Since $h_* \in H_S$ we have $\text{siz}(h_*) \leq S$, hence using the monotonicity of p and the definition of j,

$$p(\text{siz}(h_*)) m_*^a \leq p(S) m_*^a \leq m_* c^j < \lfloor m_* c^j \rfloor + 1 = S' + 1.$$

But $h_j = A(\mathbf{z}_1, S')$ and A is a loose ERM algorithm, so $\hat{\mathbf{r}}(h_j; \mathbf{z}_1) \leq \hat{\mathbf{r}}(h_*; \mathbf{z}_1)$, and condition 2 holds.

We now specify the values f_i summing to 1 and a_i summing to 1:

- $f_1 = f_3 = (C+6)^{-1}$, $f_2 = 4(C+6)^{-1}$, and $f_4 = C(C+6)^{-1}$.

- $a_1 = a_3 = (6+\sqrt{2})^{-1}$, $a_2 = 4(6+\sqrt{2})^{-1}$, and $a_4 = \sqrt{2}(6+\sqrt{2})^{-1}$.

Using $m_* = B(\alpha_*)$ and $55 > (6+\sqrt{2})^2$ we obtain

$$m_* \geq \frac{M}{a_1^2 \alpha_*^2 \nu \rho} \ln \frac{2}{f_1 \delta}.$$

Now apply Theorem 3.1 with a singleton hypothesis set, $a_1\alpha_*$ for α and $f_1\delta$ for δ; then using the facts that $\rho \leq 1$ and $d'_\nu(r,s) \leq d_\nu(r,s)$ we obtain

$$\mathbf{Pr}[\text{Condition 1 doesn't hold}] \leq f_1\delta.$$

Using $a_1 = a_3$ and $f_1 = f_3$ we also obtain

$$m_* \geq \frac{M}{a_3^2 \alpha_*^2 \nu \rho} \ln \frac{2}{f_3 \delta}.$$

Since the choice of h_j is independent of $\mathbf{z}_2$, and $|\mathbf{z}_2| = m_{**} \geq \rho m_*$, we can again apply Theorem 3.1 to obtain

$$\mathbf{Pr}[\text{Condition 4 doesn't hold}] \leq f_3\delta.$$

Using $m_* = B(\alpha_*)$, $55 > (6+\sqrt{2})^2$, $81 > 4e(6+\sqrt{2})$, and $\rho \leq 1$, we obtain

$$m_* \geq \max\left\{ \left(\frac{16M}{a_2^2\alpha_*^2\nu} c^{-1} p(S) \operatorname{lln} \frac{16eM}{a_2\alpha_*\nu} \right)^{\frac{1}{1-a}}, \frac{16M}{a_2^2\alpha_*^2\nu} \ln \frac{8}{f_2\delta} \right\}$$

Referring back to the definitions of j and S', we note that $m_* c^{j+1} < p(S) m_*^a$ (recall that $0 < c < 1$), hence

$$S' = \lfloor m_* c^j \rfloor \leq m_* c^j < c^{-1} p(S) m_*^a.$$

Then applying Lemma 3.3 with this upper bound on S' and the preceding lower bound on m_* we obtain

$$\mathbf{Pr}[\exists h \in H_{S'}.\ d_\nu(\hat{\mathbf{r}}(h;\mathbf{z}_1), \mathbf{r}(h)) > a_2\alpha_*] \leq f_2\delta.$$

Since $h_j = A(\mathbf{z}_1, S')$ and hence $h \in H_{S'}$, we then obtain

$$\mathbf{Pr}[\text{Condition 3 doesn't hold}] \leq f_2\delta.$$

Finally, we look at condition 6. In step 4 of the algorithm we look at $k \overset{\text{def}}{=} \lfloor C \ln m_* \rfloor$ hypotheses, which are tested on $m_{**} \geq \rho m_*$ examples. Using the facts that $\nu \leq 8M$ and $\alpha_* < 1$, we have

$$(a_4\alpha_*)^2 \nu < \frac{2}{(6+\sqrt{2})^2} 8M < M/3 \leq M/(3\rho).$$

Furthermore, using $m_* = B(\alpha_*)$ and $55 > (6+\sqrt{2})^2$ we obtain

$$m_* \geq \frac{2M}{a_4^2\alpha_*^2\nu\rho} \max\left\{0.44 + \ln\ln\frac{2M}{a_4^2\alpha_*^2\nu\rho},\ \ln\frac{2C}{f_4\delta}\right\}.$$

Thus the conditions of Lemma 3.4 hold; applying this lemma, we obtain

$$\mathbf{Pr}[\exists 1 \leq i \leq k.\ d_\nu(\hat{\mathbf{r}}(h_i;\mathbf{z}_2), \mathbf{r}(h_i)) > a_4\alpha_*] \leq f_4\delta.$$

and hence $\mathbf{Pr}$[Condition 6 doesn't hold] $\leq f_4\delta$.

Thus we have proven that the inequalities (3.12) hold, which completes the proof of (3.10), and hence of the theorem itself. □

3.5 Loose Occam Algorithms

The title of this chapter is "Extending Occam's Razor," but the relationship between its main result and Occam's Razor is not entirely obvious. To clarify the relationship we look at another class of algorithms that is obviously related to the Occam algorithms of Blumer et al. An Occam algorithm tries to find a near-smallest hypothesis with zero empirical risk. The obvious generalization is an algorithm that is given a risk bound R and tries to find a near-smallest hypothesis with empirical risk at most R.

Definition 3.11 Let $p(s) \geq 1$ be monotonic and polynomially bounded in s, $0 \leq a < 1$, and H be a dimension-stratified hypothesis space. We say that A is a *loose Occam algorithm for* H with bound $p(s)m^a$ if

- A takes as input a sequence of examples $\mathbf{z}$ and a risk bound $R \geq 0$;
- A outputs a hypothesis $h \in H$ which satisfies $\hat{\mathbf{r}}(h;\mathbf{z}) \leq R$ and $\text{siz}(h) \leq p(S)m^a$, where $m \stackrel{\text{def}}{=} |\mathbf{z}|$ and $S \stackrel{\text{def}}{=} \min\{\text{siz}(h') : \hat{\mathbf{r}}(h';\mathbf{z}) \leq R\}$, whenever

$$\exists h' \in H.\ \hat{\mathbf{r}}(h';\mathbf{z}) \leq R \wedge p(\text{siz}(h'))m^a < m+1.$$

(When this existence condition holds we say that $A(\mathbf{z}, R)$ *succeeds*.)

We might have proposed the following approach to learning using a stratified hypothesis space: split the examples into a training set and a test set; run a loose Occam algorithm on the training set with various risk bounds; then, of the hypotheses so obtained, choose the one with the least empirical risk on the test set. We could then have proven an analog of the main result, but for loose Occam algorithms. We chose the loose ERM algorithm approach because it appeared somewhat simpler. But does it matter which approach we choose? It could matter if, for some H, one of the approaches is computationally tractable and the other is not. We will see from Theorems 3.6 and 3.7 that, under certain mild assumptions, the two approaches are of equivalent computational complexity.

Before stating the required assumptions, let us consider some matters of representation. When we say that an algorithm returns a hypothesis $h \in H$, what we really mean is that it returns some *representation* or *encoding* of a hypothesis $h \in H$. We then must carefully distinguish between the size of a hypothesis $h \in H$, denoted $\text{siz}(h)$ (Definition 3.5), and the size of the representation of h, which is the length of the string of symbols representing h.

Definition 3.12 The *repsize* of an abstract value x, denoted $\text{rsiz}(x)$, is the length of the string of symbols representing x.

Note that there may not be any bound at all on $\text{rsiz}(h)$ even when requiring $h \in H_i$ for some fixed i. For example, if the hypotheses in H_i are defined by j rational-valued parameters, then we can find hypotheses in H_i of arbitrarily large repsize simply by choosing the parameter values to have the form p/q for very large, relatively-prime integers p and q.

We make the following assumptions:

1. $|\mathbf{z}|$ is bounded by a polynomial in $\text{rsiz}(\mathbf{z})$.

2. A rational number r is represented by (the representation of) a pair of integers (p, q) such that $r = p/q$; or we can convert the actual representation to and from this representation in polynomial time.

3. $\hat{\mathbf{r}}(h; \mathbf{z})$ is computable in polynomial time.

4. $\text{siz}(h)$ and $|\mathbf{z}|$ are computable in polynomial time.

5. If $1 \leq i \leq |\mathbf{z}|$ and $H_i \neq \emptyset$, then there exists a hypothesis $h \in H_i$ achieving the minimum empirical risk on $\mathbf{z}$, and its repsize is bounded by a polynomial in i and $\text{rsiz}(\mathbf{z})$. (Note that, combined with Assumption 1, this means that $\text{rsiz}(h)$ is bounded by a polynomial q in $\text{rsiz}(\mathbf{z})$ only.)

Assumption 1 will be satisfied by any standard representation of sequences. To violate it would require a representation that does some sort of data compression to allow a long but very structured sequence of examples to have a short representation. Assumption 2 just specifies the standard representation of rational numbers, or a polynomial-time equivalent representation. Assumption 3 is a standard kind of assumption in computational learning theory [32] — if you can't even evaluate a hypothesis on the training data in polynomial time, it seems pointless to hope for a polynomial learning algorithm. Assumption 4 merely says that we haven't chosen such perverse representations that it's an intractable problem just to determine the size of a hypothesis or length of a sequence of examples. If Assumption 5 does not hold, then the problem of minimizing empirical risk for a given hypothesis-size bound is absurd to even consider, since the answer cannot even be written in polynomial time!

Definition 3.13 A loose Occam algorithm A is said to be *fully polynomial* if both of the following hold:

1. Its worst-case run time is bounded by a polynomial in the repsize of its inputs.

2. The repsize of $\hat{\mathbf{r}}(A(\mathbf{z}, R); \mathbf{z})$ is bounded by a polynomial in $\text{rsiz}(\mathbf{z})$.

Let us consider why it might be reasonable to expect (2) above to hold. Let q be the polynomial mentioned in Assumption 5 and $\sigma \overset{\text{def}}{=} \text{rsiz}(\mathbf{z})$. Then there is no need to even consider hypotheses of repsize greater than $q(\sigma)$. If our loose Occam algorithm A nevertheless considers hypotheses of repsize up to $q'(q(\sigma), \sigma)$, where q' is a polynomial, the repsize of $A(\mathbf{z}, R)$ is still bounded by a polynomial in $\text{rsiz}(\mathbf{z})$. Then by Assumption 3 the repsize of $\hat{\mathbf{r}}(A(\mathbf{z}, R); \mathbf{z})$ will be bounded by a polynomial in σ.

Our result is then the following: there exists a polynomial-time loose ERM algorithm for H iff there exists a fully polynomial loose Occam algorithm for H. We prove this in two parts.

Theorem 3.6 If there exists a polynomial-time loose ERM algorithm for H with bound $p(s)m^a$ then there exists a fully polynomial loose Occam algorithm for H with the same bound.

Proof. Let A be a loose ERM algorithm for H with bound $p(s)m^a$. We define A' to be the algorithm that takes as input a sequence of examples $\mathbf{z}$ and a risk bound $R \geq 0$, and does the following:

1. For each $1 \leq i \leq m \overset{\text{def}}{=} |\mathbf{z}|$, compute $h_i \overset{\text{def}}{=} A(\mathbf{z}, i)$.

2. If $\hat{\mathbf{r}}(h_i; \mathbf{z}) > R$ for all $1 \leq i \leq m$, then output a fixed hypothesis $h_0 \in H$.

3. Otherwise return h_k, where $k \overset{\text{def}}{=} \min\{i : \hat{\mathbf{r}}(h_i; \mathbf{z}) \leq R\}$.

To see that A' runs in polynomial time, note that m can be computed in polynomial time and is bounded by a polynomial in $\text{rsiz}(\mathbf{z})$. Thus each h_i can be computed in polynomial time, which furthermore implies that $\text{rsiz}(h_i)$ is bounded by a polynomial in $\text{rsiz}(\mathbf{z})$. Then since $\hat{\mathbf{r}}(h_i; \mathbf{z})$ can be computed in time polynomial in $\text{rsiz}(\mathbf{z})$ and $\text{rsiz}(h_i)$, and we have $O(m)$ such computations, the entire algorithm runs in polynomial time.

We now show that A' is a loose Occam algorithm with bound $p(s)m^a$. If $\hat{\mathbf{r}}(h_i) > R$ then, since A is a loose ERM algorithm, we know that $\hat{\mathbf{r}}(h) \geq \hat{\mathbf{r}}(h_i) > R$ for all $h \in H$ satisfying $p(\text{siz}(h))m^a < i + 1$. Thus if $\hat{\mathbf{r}}(h_i) > R$ for all $1 \leq i \leq m$, we have that $\hat{\mathbf{r}}(h) > R$ for all $h \in H$ s.t. $p(\text{siz}(h))m^a < m + 1$, in which case we see from the definition of a loose Occam algorithm that it doesn't matter what A' outputs.

Otherwise, A' returns h_k and we have $\hat{\mathbf{r}}(h_k) \leq R$ and $\hat{\mathbf{r}}(h_i) > R$ for all $i < k$. Suppose that $k > 1$. Then, since A is a loose ERM algorithm, we have

$$R < \hat{\mathbf{r}}(h_{k-1}) \leq \min\{\hat{\mathbf{r}}(h) : h \in H \wedge p(\text{siz}(h))m^a < k\},$$

and from this we derive that $\hat{\mathbf{r}}(h) > R$ for all $h \in H$ s.t. $p(\text{siz}(h))m^a < k$. Then $p(\text{siz}(h))m^a \geq k \geq \text{siz}(h_k)$ for all $h \in H$ s.t. $\hat{\mathbf{r}}(h) \leq R$, satisfying the requirements of a loose Occam algorithm. If $k = 1$ then, since $p(\text{siz}(h)) \geq 1$ and $m \geq 1$, we also have $p(\mathrm{siz}(h))m^a \geq k \geq \mathrm{siz}(h_k)$ for all $h \in H$. So A' is a loose Occam algorithm for H with bound $p(s)m^a$.

Finally, since h_0 is a constant (making $\mathrm{rsiz}(h_0)$ also a constant) and $\mathrm{rsiz}(h_i)$ is bounded by the same polynomial in $\mathrm{rsiz}(\mathbf{z})$ for all $1 \leq i \leq m$, the repsize of the hypothesis output is bounded by a polynomial in $\mathrm{rsiz}(\mathbf{z})$. Since $\hat{\mathbf{r}}(h_k; \mathbf{z})$ can be computed in polynomial time, the repsize of $\hat{\mathbf{r}}(h_k; \mathbf{z})$ is bounded by a polynomial in $\mathrm{rsiz}(\mathbf{z})$, and we see that A' is fully polynomial. □

Theorem 3.7 If there exists a fully polynomial loose Occam algorithm for H with bound $p(s)m^a$, then there exists a polynomial-time loose ERM algorithm for H with the same bound.

Proof. Let us define the following:

- A is a fully polynomial loose Occam algorithm for H with bound $p(s)m^a$.
- q_H is a polynomial such that the repsize of the minimum empirical error on $\mathbf{z}$ achievable by hypotheses from H_i is at most $q_H(\sigma)$, where $\sigma \overset{\mathrm{def}}{=} \mathrm{rsiz}(\mathbf{z})$, when $1 \leq i \leq |\mathbf{z}|$. Such a polynomial exists by Assumptions 3 and 5.
- q_A is a polynomial such that the repsize of $\hat{\mathbf{r}}(A(\mathbf{z}, R); \mathbf{z})$ is at most $q_A(\sigma)$, for all $\mathbf{z}$ and R. This exists because A is fully polynomial.
- $q(\sigma) \overset{\mathrm{def}}{=} \max(q_H(\sigma), q_A(\sigma))$.
- $E(\mathbf{z})$ is the finite set of rational numbers containing $\hat{\mathbf{r}}(A(\mathbf{z}, R); \mathbf{z})$ for $0 \leq R \leq M$, and $\min\{\hat{\mathbf{r}}(h; \mathbf{z}) : h \in H_i\}$ for all $1 \leq i \leq |\mathbf{z}|$. (We define $\min \emptyset$ to be M).
- $d(\mathbf{z})$ is any positive integer having repsize $2q(\sigma) + 1$.

The elements of $E(\mathbf{z})$ have a repsize of at most $q(\sigma)$; thus their denominators have a repsize of at most $q(\sigma)$, and the difference of any two elements of $E(\mathbf{z})$ has a denominator of repsize at most $2q(\sigma)$, and hence a magnitude greater than $1/d(\mathbf{z})$. So any interval $[j/d(\mathbf{z}), (j+1)/d(\mathbf{z})]$ contains at most one element of $E(\mathbf{z})$.

We then define A' to be the following algorithm:

1. Input a sequence of examples $\mathbf{z}$ and an integer i s.t. $1 \leq i \leq m \overset{\mathrm{def}}{=} |\mathbf{z}|$. In the following define
   - $\mathbf{h}[R] \overset{\mathrm{def}}{=} A(\mathbf{z}; R)$;
   - $P(R) \overset{\mathrm{def}}{=} (\hat{\mathbf{r}}(\mathbf{h}[R]; \mathbf{z}) \leq R \wedge \mathbf{h}[R] \in H_i)$.
2. $d := d(\mathbf{z})$; $N := \lceil Md \rceil$
3. If $P(0)$ then output $\mathbf{h}[0]$.
4. If $\neg P(M)$ then output some constant hypothesis $h_0 \in H$.

5. Otherwise, starting with 0 and N as bounds, do a binary search on the set of integers from 0 to N to find j_0 such that $P(j_0/d)$ and $\neg P((j_0-1)/d)$. Output $\mathbf{h}[j_0/d]$.

Note that d can be computed in polynomial time, and since M is a constant, the computation of N takes place in polynomial time. Thus $\log(N)$ is polynomial in $\sigma = \mathrm{rsiz}(\mathbf{z})$. The binary search takes $O(\log(N))$ steps, which is polynomial in σ. Since empirical risk, the size of a hypothesis, and the size of a sequence of examples can be computed in polynomial time, the entire algorithm runs in polynomial time.

We now show that A' is a loose ERM algorithm for H with bound $p(s)m^a$.

Case 1: $P(0)$ holds. Then A' outputs a hypothesis $h \in H_i$ with zero empirical risk, trivially satisfying the requirements of a loose ERM algorithm.

Case 2: There is no $h \in H$ s.t. $p(\mathrm{siz}(h))m^a < i+1$. Then we see from the definition of a loose ERM algorithm that it doesn't matter what hypothesis A' outputs.

Case 3: $\neg P(0)$ and there is some $h \in H$ s.t. $p(\mathrm{siz}(h))m^a < i+1 \leq m+1$. Then $A(\mathbf{z}, M)$ succeeds, hence $P(M)$ holds and the binary search is carried out. A' outputs $\mathbf{h}[j_0/d]$, where $j_0 > 0$, $P(j_0/d)$ holds and $P((j_0-1)/d)$ does not. Let k be the greatest integer such that $p(k)m^a < i+1$, and let $R_0 \overset{\mathrm{def}}{=} \min\{\hat{\mathbf{r}}(h;\mathbf{z}) : h \in H_k\}$. Since A is a loose Occam algorithm and $\neg P((j_0-1)/d)$, we have that $R_0 > (j_0-1)/d$. Let $R_1 \overset{\mathrm{def}}{=} \hat{\mathbf{r}}(\mathbf{h}[j_0/d];\mathbf{z})$. Since $P(j_0/d)$, we have that $R_1 \leq j_0/d$. But R_0 and R_1 are both elements of $E(\mathbf{z})$, and the interval $[(j_0-1)/d, j_0/d]$ contains at most one element of $E(\mathbf{z})$. Thus $R_1 \leq R_0$, and since $\mathbf{h}[j_0/d] \in H_i$ we see that the requirements of a loose ERM algorithm have been met. □

Chapter 4

The Minimum Feature Set Problem

4.1 Introduction

Consider the task of learning from examples an (unknown) linear threshold function f of n real-valued features. Assuming that a learning algorithm uses the hypothesis space consistently, Theorem 2.1 gives an upper bound on its sample complexity that is linear in the VC dimension of the hypothesis space, which in this case is $n + 1$. If f depends on only a small subset of the features, this argues for removing the superfluous features; but often it is not known beforehand on which specific features f depends. Thus there has long been interest in methods of reducing the number of features actually used (i.e. the number of non-zero input weights), while still correctly classifying all the training examples. Some heuristic techniques for doing this are given in [38], chapter 7; while useful, such techniques come without guarantees as to how closely the minimum number of features is approximated. (Note that Littlestone [29] gives an alternative approach to the problem of many unnecessary features, for a restricted class of linear threshold functions with Boolean inputs.)

In Section 4.2 we quantify the benefits of minimizing the number of features used, by giving an upper bound on the VC dimension of the set of linear threshold functions with n inputs and at most s non-zero input weights. For $s \ll n$ this bound is much less than $n + 1$. This VC dimension bound improves on the bound that follows from the (more general) results of Baum and Haussler [3].

In section 4.3 we examine the time-complexity of this feature-minimization problem, which we call MINIMUM FEATURE SET (MIN FS). We show that MIN FS is both NP-hard and difficult to approximate. In particular, we show the following:

- No polynomial-time algorithm can approximate MIN FS to within a constant factor, unless P = NP.
- No polynomial-time algorithm can approximate MIN FS to within an $o(\log m)$ factor, where m is the number of training examples, unless NP $\subseteq$ DTIME$[n^{\log \log n}]$.

4.2 Benefits of Minimization

Let H be the set of linear threshold functions on n real-valued features. We will identify $h \in H$ with its defining weight vector $\mathbf{w}$, i.e. the vector $\mathbf{w} \in \mathcal{Q}^{n+1}$ s.t.

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } w_{n+1} + \sum_{i=1}^{n} w_i x_i \geq 0 \\ 0 & \text{if } w_{n+1} + \sum_{i=1}^{n} w_i x_i < 0. \end{cases}$$

($\mathcal{Q}$ is the set of rational numbers.) We have $\text{VCdim}(H) = n + 1$ [6].

Definition 4.1 The *cost* of $\mathbf{w} \in \mathcal{Q}^{n+1}$ (or the corresponding $h \in H$) is the number of nonzero weights other than the bias weight w_{n+1}, i.e. $|\{i : w_i \neq 0, 1 \leq i \leq n\}|$.

Definition 4.2 Let $S \subseteq \mathcal{N} \overset{\text{def}}{=} \{1, \ldots, n\}$ and $1 \leq s \leq n$; then we define

- $H[S]$ is the set of $h \in H$ which have nonzero weights (other than the bias) only for the features in S, and
- $H[s]$ is the set of $h \in H$ whose cost is at most s.

We now have a stratified hypothesis space $(H[s])_{0 \leq s \leq n}$. Suppose a learning algorithm $\mathcal{A}$ uses H consistently and also minimizes the cost of the hypothesis it returns. If s is the cost of the function being learned, then $\mathcal{A}$ also uses $H[s]$ consistently, and we have an upper bound on sample complexity that is linear in $\text{VCdim}(H[s])$ instead of $\text{VCdim}(H)$. Even if $\mathcal{A}$ only does approximate minimization, Theorem 2.3 can give reduced sample-complexity bounds that use $\text{VCdim}(H[s])$ rather than $\text{VCdim}(H)$. Thus it is useful to have good bounds on $\text{VCdim}(H[s])$.

We will need to use the following theorem given in [6], which relates $\text{VCdim}(H)$ and $\Pi_H(m)$:

Theorem 4.1 If $\text{VCdim}(H) = d$ and $m \geq d \geq 1$, then $\Pi_H(m) \leq (em/d)^d$, where e is the base of the natural logarithm.

($\Pi_H(m)$ was defined in Section 2.2.2.) We now state and prove our bound on $\text{VCdim}(H[s])$.

Theorem 4.2 Let $1 \leq s < n$. Then $\text{VCdim}(H[s]) < 2.71(s+1)\log_2(en/(s+1))$.

Proof. We can assume that $s + 1 \leq n/2$. If not, i.e. $s + 1 > n/2$, then $n \geq s + 1 \geq (n+1)/2$ (since s and n are integers), and writing lg for $\log_2$ we have

$$2.71(s+1)\lg\left(\frac{en}{s+1}\right) \geq 2.71(s+1)\lg e \geq \frac{2.71}{2}(n+1)\lg e > n + 1.$$

But since $H[s] \subseteq H$ we have $\text{VCdim}(H[s]) \leq \text{VCdim}(H) = n + 1$, which is less than our bound.

We now compute $\Pi_{H[s]}(m)$. The set of linear threshold functions of s inputs has VC dimension $s + 1$. Thus $\text{VCdim}(H[S]) = s + 1$ for all $S \subseteq \mathcal{N}$, $|S| = s$, and by

Theorem 4.1 this gives $\Pi_{H[S]}(m) \leq (em/(s+1))^{s+1}$ for all $m \geq s+1$. Using this inequality we obtain

$$\Pi_{H[s]}(m) \leq \sum_{|S|=s} \Pi_{H[S]}(m) \leq \binom{n}{s} (em/(s+1))^{s+1}.$$

By the assumption that $s+1 \leq n/2$ we have $\binom{n}{s+1} > \binom{n}{s}$. A variant of Stirling's formula states that $\binom{j}{k} \leq (ej/k)^k$, for all j and k ([11], p. 102). Combining these we obtain

$$\Pi_{H[s]}(m) < (en/(s+1))^{s+1}(em/(s+1))^{s+1} \text{ for all } m \geq s+1.$$

Let $x_0 \overset{\text{def}}{=} \lg(en/(s+1))$ and $m_0 \overset{\text{def}}{=} 2.71(s+1)x_0$. To prove the theorem it suffices to show that $\Pi_{H[s]}(m_0) < 2^{m_0}$, by the definition of the VC dimension. This inequality will hold if $m_0 - \lg(\Pi_{H[s]}(m_0)) > 0$. Dividing this inequality by $s+1$, and using the fact that $m_0 \geq s+1$, we find

$$\frac{m_0 - \lg(\Pi_{H[s]}(m_0))}{s+1} > \frac{m_0}{s+1} - x_0 - \lg\left(\frac{em_0}{s+1}\right) = 1.71x_0 - \lg(2.71ex_0)$$

Thus it suffices to show that $f(x_0) \geq 0$, where $f(x) \overset{\text{def}}{=} 1.71x - \lg(2.71ex)$. We have $f'(x) = 1.71 - (x \ln 2)^{-1}$, hence $f(x)$ is non-decreasing for all $x \geq x_{min} \overset{\text{def}}{=} (1.71 \ln 2)^{-1}$. Thus it suffices to show that $f(x) \geq 0$ for some $x_{min} \leq x \leq x_0$. Let us choose $x = \lg(2e)$. Since $s+1 \leq n/2$ we have $x_0 \geq \lg(2e)$. We also have

$$\lg(2e) = \ln(2e)/\ln 2 > 1.71^{-1}/\ln 2 = x_{min}.$$

Finally, by computation we find $f(\lg(2e)) \approx 7.55 \times 10^{-3} > 0$. □

The above VC dimension bound is a significant improvement over $n+1$ when $s \ll n$. It is also an improvement over the $\Theta(s \log(sn))$ bound that can be obtained from Corollary 5 of [3], which is a more general result applicable to multi-layered neural nets.

4.3 The complexity of minimization

We can formally state the minimization problem we have been discussing as follows:

Definition 4.3 MINIMUM FEATURE SET (MIN FS)
Instance: Integers $m \geq 1$, $n \geq 1$, and an $m \times (n+1)$ matrix $\mathbf{A}$ of rational numbers s.t. $a_{i,n+1} \in \{1, -1\}$ for all i.
Problem: Find a $\mathbf{w} \in \mathcal{Q}^{n+1}$ satisfying $\mathbf{Aw} > 0$, of minimum cost. (Recall that the cost of $\mathbf{w}$ is the number of nonzero w_i, $i \neq n+1$.)

In the above, m is the number of training examples and n is the number of features. $\mathbf{A}_i$, the i-th row of the matrix $\mathbf{A}$, is obtained from the i-th training example $(\mathbf{x}_i, y_i)$ as follows: first, augment $\mathbf{x}_i$ with 1 to obtain an $(n+1)$-element vector $\alpha(\mathbf{x}_i)$; then, if it is a negative example ($y_i = 0$), negate this vector. The requirement that $\mathbf{Aw} > 0$ is equivalent to $\mathbf{A}_i\mathbf{w} > 0$ for all i, so this gives us $\alpha(\mathbf{x}_i) \cdot \mathbf{w} > 0$ for positive examples and $\alpha(\mathbf{x}_i) \cdot \mathbf{w} < 0$ for negative examples.

The requirement that $a_{i,n+1} \in \{1, -1\}$ arises from the above construction, in which the example vectors were augmented with 1 to provide for the bias term; thus $a_{i,n+1} = 1$ for positive examples and $a_{i,n+1} = -1$ for negative examples.

The requirement $\mathbf{Aw} > 0$ may seem too strong, given that the linear threshold function corresponding to $\mathbf{w}$ classifies $\mathbf{x}$ as positive if $\mathbf{w} \cdot \mathbf{x} \geq 0$. To see that this is not a problem, suppose that we have a weight vector $\mathbf{w}$ satisfying the relaxed requirement that $\mathbf{A}_i\mathbf{w} \geq 0$ when $a_{i,n+1} = 1$ (positive example), and $\mathbf{A}_i\mathbf{w} > 0$ when $a_{i,n+1} = -1$ (negative example). Let $\gamma \overset{\text{def}}{=} \min\{\mathbf{A}_i\mathbf{w} : a_{i,n+1} = -1\}$. Then if we add $\gamma/2$ to w_{n+1} to obtain $\mathbf{w}'$, we see that $\mathbf{w}'$ has the same cost as $\mathbf{w}$ and additionally satisifies $\mathbf{Aw} > 0$.

We now give some standard definitions that will be needed in our investigation of the difficulty of solving or approximating MIN FS.

Definition 4.4 A *minimization* problem $\mathcal{M}$ has the following form, for some pair of predicates P and Q and integer-valued *cost* function κ: Given x satisfying $P(x)$ (a problem *instance*), find some y satisfying $Q(x, y)$ (a *solution* for x) that minimizes $\kappa(x, y)$.

Definition 4.5 A *polynomial-time approximation algorithm (PTAA)* $\mathcal{A}$ for a minimization problem $\mathcal{M}$ is a polynomial-time algorithm which takes as input some instance x satisfying $P(x)$ and outputs a y satisfying $Q(x, y)$. We say that $\mathcal{A}$ *approximates* $\mathcal{M}$ *to within a factor* $\phi(x)$ if, additionally, $\kappa(x, y) \leq \phi(x)\kappa(x, z)$ for any z such that $Q(x, z)$. We say that $\mathcal{M}$ *can be approximated to within a factor* $\phi(x)$ if there is a PTAA that approximates it to within a factor $\phi(x)$.

Definition 4.6 A *cost-preserving* polynomial transformation from minimization problem $\mathcal{M}$ to minimization problem $\mathcal{M}'$ is a pair of functions (t_1, t_2) with the following properties, where x is taken to be an instance of $\mathcal{M}$:

1. t_1 maps instances of $\mathcal{M}$ to instances of $\mathcal{M}'$. t_2 maps pairs (y, x), where y is a solution for $t_1(x)$, to solutions for x.
2. t_1 and t_2 are both computable in polynomial time.
3. If x has a solution of cost k, then $t_1(x)$ has a solution of cost at most k.
4. If y is a solution for $t_1(x)$ of cost k, then $t_2(y, x)$ has cost at most k.

Suppose there exist both a cost-preserving polynomial transformation from $\mathcal{M}$ to $\mathcal{M}'$ and a PTAA that approximates $\mathcal{M}'$ to within a factor $\phi(x')$. Then there is a PTAA that approximates $\mathcal{M}$ to within a factor $\phi(t_1(x))$: compute $x' = t_1(x)$, run the PTAA for $\mathcal{M}'$ on x' to get y, and output $t_2(y, x)$. Thus any limits on the approximability of $\mathcal{M}$ give corresponding limits on the approximability of $\mathcal{M}'$. We will show that MIN FS is both NP-hard and difficult to approximate via a transformation from MIN SET COVER.

Definition 4.7 MINIMUM SET COVER (MIN SC)
Instance: A finite set S and a collection C of subsets of S.
Problem: Find a cover of S (a set $C' \subseteq C$ s.t. $\bigcup C' = S$) of minimum cardinality.

Theorem 4.3 There is a cost-preserving polynomial transformation from MIN SC to MIN FS in which instances (S, C) of MIN SC are mapped to instances $(m, n, \mathbf{A})$ of MIN FS with $m = |S| + 1$.

Proof. In the following we shall use the notation (Φ), where Φ is a logical formula, for (if Φ then 1 else 0). This notation is from [16].

Let (S, C) be an instance of MIN SC. Define $m = |S| + 1$ and $n = |C|$. Enumerate the elements of S as $s_1, s_2, \ldots, s_{m-1}$ and the elements of C as $c_1, c_2, \ldots, c_n$. Define $t_1(S, C) = (m, n, \mathbf{A})$, where $\mathbf{A}$ is an $m \times (n+1)$ matrix constructed from the column vectors of 1's and 0's corresponding to each c_j, as follows:

$$\begin{array}{|c|c|c|c|c|}
\hline
 & & & & \vdots \\
c_1 & c_2 & \cdots & c_n & -1 \\
 & & & & \vdots \\
\hline
\multicolumn{4}{|c|}{\cdots 0 \cdots} & 1 \\
\hline
\end{array}$$

(Thus, $a_{ij} = (s_i \in c_j)$ for $i < m$ and $j \leq n$.)

Note that the inequality $\mathbf{Aw} > 0$ is equivalent to the set of inequalities $\mathbf{A}_i\mathbf{w} > 0$, $1 \leq i \leq m$. This set of inequalities is in turn equivalent to

$$\sum_{j \leq n} (s_i \in c_j) w_j > w_{n+1} > 0, \quad 1 \leq i < m \tag{4.1}$$

Suppose that (S, C) has a solution of cost k, i.e. there exists some $C' \subseteq C$ which is a cover of S, and $|C'| = k$. Define the vector $\mathbf{w}$ by

$$w_j = \begin{cases} (c_j \in C') & \text{if } j \leq n \\ 0.5 & \text{if } j = n+1. \end{cases}$$

Note that $\mathbf{w}$ has cost k and $w_{n+1} > 0$. Using the fact that C' is a cover of S we have also that, for $1 \leq i < m$,

$$\sum_{j \leq n} (s_i \in c_j) w_j = \sum_{j \leq n} (s_i \in c_j)(c_j \in C') > 1 > w_{n+1},$$

so $\mathbf{w}$ satisfies (4.1). Thus we have shown that $t_1(S, C) = (m, n, \mathbf{A})$ has a solution of cost k.

We now consider the reverse mapping. For all $\mathbf{w} \in \mathcal{Q}^{n+1}$, define

$$t_2(\mathbf{w}, S, C) = \{c_j : w_j > 0, 1 \leq j \leq n\}.$$

Suppose that $\mathbf{w}$ is a solution for $t_1(S, C) = (m, n, \mathbf{A})$, of cost k. Let $C' = t_2(\mathbf{w}, S, C)$. It is clear from the definition of t_2 that $C' \subseteq C$ and $|C'| \leq k$. We have furthermore that $\mathbf{Aw} > 0$ and hence $\mathbf{w}$ satisfies the inequalities (4.1). Thus for each i there is some j s.t. $s_i \in c_j$ and $w_j > 0$. But $w_j > 0$ implies $c_j \in C'$, so C' is a cover for S. Thus we have shown that $\mathbf{w}$ is a solution for (S, C) of cost at most k. $\square$

Corollary 4.4 MIN FS is NP-hard. In addition, we have the following approximability limits:

1. For every constant $c \geq 1$, MIN FS cannot be approximated to within a factor c, unless P = NP.

2. MIN FS cannot be approximated to within an $o(\log m)$ factor, unless NP $\subseteq$ DTIME$[n^{\log\log n}]$.

Proof. MIN SC is NP-hard [13], hence by Theorem 4.3, so is MIN FS.

From Theorem 4.3, if MIN FS can be approximated to within some constant factor c, so can MIN SC. But Bellare, Goldwasser, Lund and Russel [4] have shown that MIN SC cannot be approximated to within a constant factor unless P = NP.

Suppose that MIN FS can be approximated to within a factor $g(m, n, \mathbf{A}) = f(m)$, where $f(m) = o(\log m)$. Then MIN SC can be approximated to within a factor $g(t_1(S, C)) = f(|S| + 1) = o(\log(|S| + 1)) = o(\log |S|)$. But Bellare et al. have also shown that MIN SC cannot be approximated to within an $o(\log |S|)$ factor unless NP $\subseteq$ DTIME$[n^{\log\log n}]$. $\square$

Haussler [17] looks at the problem of finding a pure conjunctive concept consistent with a set of examples, with the minimum number of conjuncts, and shows that it is NP-hard via a reduction from MIN SC. His reduction also defines a cost-preserving polynomial transformation that produces $|S| + 1$ examples. Thus the conclusions of Corollary 4.4 also apply to Haussler's problem.

We note in closing that Corollary 4.4 should *not* be taken as evidence against the possibility of obtaining improved sample-complexity bounds by approximating MIN FS. For example, if the approximation factor grows polylogarithmically with the number of examples, then Theorem 3.2.1 of [6] (which contains a variant of Theorem 2.3 in which the size bound $p(s)m^a$ is replaced by $p(s)(\log m)^l$ for some $l > 0$), or Lemma 5.6 of [17], can be used to establish improved upper bounds on sample complexity.

Chapter 5

Robust Learning with Linear-Threshold Functions

5.1 Introduction

In Chapter 4 we touched on the problem of learning linear threshold functions (also known as *halfspaces*, due to their geometric interpretation) under the PAC model. It is well known [6] that linear programming methods can be applied to obtain a polynomial algorithm for learning halfspaces. (The weight minimization discussed in Chapter 4, although it may reduce sample complexity, is not needed to obtain a polynomial learning algorithm.) In this chapter we analyze how learning performance degrades when the representational power of halfspaces is overstrained, i.e., if more complex target functions than just halfspaces are allowed.

A rigorous analysis of this question is based on the notion of PAO (probably almost optimal) learnability [20], because we must not insist on probably almost correct hypotheses. If PAO learning is possible for arbitrary target functions, we speak of robust learning. If we can come within a constant factor of the optimal error, we speak of learning with limited degradation. We show that neither robust learning nor learning with limited degradation is possible when using halfspaces, unless RP = NP. The proofs are based on a method that associates an optimization problem with a learning problem. If the optimization problem is NP-hard, then its corresponding learning problem is intractable unless RP = NP.

5.2 Robust Learning and Minimization

The version of PAO learning discussed in this chapter is a slight variant of the unstratified PAC model of Section 2.2.1. The only new element is the following:

- We have a *target space* T, whose elements are Boolean functions on the instance space X. The *target function* f may be any element of T.

The definition of the error of a hypothesis remains the same. The relaxed requirement on the target function, however, leads to a modified definition of sample complexity.

Definition 5.1 A hypothesis h is *ϵ-optimal* if

$$\mathrm{err}(h) \le \mathrm{opt}(H) + \epsilon, \text{ where } \mathrm{opt}(H) \overset{\mathrm{def}}{=} \inf_{h' \in H} \mathrm{err}(h'),$$

i.e. if the error of h comes within ϵ of the minimum achievable by hypotheses from H.

Definition 5.2 The *PAO sample complexity* $\mathcal{S}(\epsilon, \delta)$ of an algorithm for learning T by H is the worst-case number of examples needed to have probability $1 - \delta$ or better of returning an ϵ-optimal hypothesis, when the target f may be any element of T and any distribution over the instance space X is allowed.

As for the PAC model, X, H and T may be implicitly parameterized by a variable n, in which case the sample complexity is also a function of n.

Definition 5.3 An algorithm for learning T by H which runs in polynomial time, and whose PAO sample complexity is bounded by a polynomial in ϵ^{-1}, δ^{-1}, and n, is called a *polynomial algorithm for learning T by H*. If such an algorithm exists, we say that *T is PAO learnable by H*.

An important special case of PAO learning occurs when $T \subseteq H$. In this case the target $f \in H$, so $\mathrm{opt}(H) = 0$ and an ϵ-optimal hypothesis has error at most ϵ. Thus the notion of PAO learnability becomes equivalent to PAC learnability.

Definition 5.4 We assume that X is augmented with a σ-algebra $\mathcal{A}$ of events. For instance, $\mathcal{A}$ might be the powerset of X if X is countable, or the system of Borel sets if $X = \mathcal{R}^n$. $\mathcal{A}$ is the set of all subsets S of X for which it is meaningful to speak of the probability of S. (For definitions of "σ-algebra" and "Borel sets," and a discussion of technical issues in the foundations of probability theory, see for example [2], chapters 13 and 14.)

The notion of PAO learnability becomes more interesting if $H \subset T$, in which case we may have $\mathrm{opt}(H) > 0$. An extreme situation occurs if we allow arbitrary concepts, i.e., $T = \mathcal{A}$. It is practically important that learning algorithms be robust in the sense that their performance degrades 'gracefully' when $\mathrm{opt}(H) > 0$. This considerations motivates the following definition:

Definition 5.5 We say that *H allows robust learning* if $\mathcal{A}$ is PAO learnable by H.

From Theorem 2.2 we saw that the PAC learnability of H was related to the consistency problem for H. Similarly, the question of whether H allows robust learning is related to the following *minimizing disagreements problem*:

Definition 5.6 MinDis(H).
Input: A sequence $\mathbf{z}$ of examples $z_i \in X \times \{0,1\}$. (We call $\mathbf{z}$ the *input sample.*)
Output: A hypothesis $h \in H$ which misclassifies the minimum number of examples z_i possible.

Definition 5.7 If T is a target space, we say that $\mathbf{z}$ is T*-legal* if T contains a function f which is consistent with $\mathbf{z}$.

Theorem 5.1 If RP $\neq$ NP and MinDis(H) restricted to T-legal input samples is NP-hard, then T is not PAO learnable by H.

Theorem 5.1 is proven in [21], and a slightly different form of it was previously proven in [1]. A stronger version is proven in Section 5.3. Setting $T = \mathcal{A}$ gives the following corollary:

Corollary 5.2 If RP $\neq$ NP and MinDis(H) is NP-hard, then H does not allow robust learning.

We now turn our attention to the case of $H =$ Halfspaces.

Theorem 5.3 MinDis(Halfspaces) is NP-hard.

Proof. Follows from the stronger Theorem 5.8 which we prove in Section 5.3. A. Blum [5] independently suggested another proof using a polynomial transformation from the NP-complete problem Open Hemisphere (problem MP6 in [13]) to the decision problem corresponding to MinDis(Halfspaces). □

Open Hemisphere is just the decision problem corresponding to MinDis(Homogeneous Halfspaces). (A homogeneous halfspace is a halfspace whose separating hyperplane passes through the origin, or equivalently, a linear threshold function with a bias weight of 0.) Thus MinDis(Homogeneous Halfspaces) is also NP-hard. Using this fact and Theorem 5.3, we immediately obtain the following corollary:

Corollary 5.4 Neither Halfspaces nor Homogeneous Halfspaces allows robust learning, unless RP = NP.

This result is strengthened in Section 5.3.

5.3 Limited Degradation and Halfspaces

We have ruled out the possibility that halfspaces allow robust learning (unless RP = NP). Robust learning requires in the confident case that the error of the hypothesis is bounded by $\text{opt}(H) + \epsilon$. We shall consider two ways of relaxing the requirements of robust learning. One is to only require that we approach some fixed multiple $d \geq 1$ of the minimum error achievable by hypotheses from H. This can still be of practical use if d is not too large. The other is to enlarge our learning environment. Assume that we have a hypothesis space H', which defines the optimal error $\text{opt}(H')$ achievable using hypotheses in H' to approximate the given concept. To relax the restrictions of the learning problem we may now increase the power of the hypothesis space while preserving the value of $\text{opt}(H')$ as the measure of our learning success. This means we use the hypothesis space H' as a touchstone to define the goal of the learning problem, but allow our learning algorithm to consider hypotheses from a larger space $H \supseteq H'$. This framework was introduced by Kearns et al. in [25]. We combine these relaxations in the following definitions:

Definition 5.8 Let L be an algorithm for learning T by H. Suppose that, given $p(\epsilon, \delta, n)$ training examples, there is a probability of $1 - \delta$ or better that L returns a hypothesis $h \in H$ whose error is at most $d \cdot \text{opt}(H') + \epsilon$, regardless of the target function T and distribution D over $X^{[n]}$. Then we say that the *H'-degradation* of L is *limited by d*, with sample complexity $p(\epsilon, \delta, n)$.

Definition 5.9 We say that *the H'-degradation of H is limited by d when learning T* if there exists a polynomial-time algorithm for learning T by H whose H'-degradation is limited by d, with a sample complexity polynomial in ϵ^{-1}, δ^{-1}, and n.

- We say that *the H'-degradation of H is unlimited when learning T* if the degradation is not limited by any fixed d when learning T.
- We use H and $\mathcal{A}$ as the default values for H' and T respectively, e.g. "the degradation of H is unlimited" means that the H-degradation of H is unlimited when learning $\mathcal{A}$.

The question of whether the H'-degradation of H is limited is related to the hardness of approximating $\text{MinDis}(H)$. The reader may want to review the definitions of a minimization problem (Definition 4.4) and PTAA (Definition 4.5), from Chapter 4, before proceeding. The following definition will also be needed.

Definition 5.10 We say that a sample $\mathbf{z}$ is (H, H')-*indifferent* if the minimum number of misclassifications on $\mathbf{z}$ achievable by hypotheses from H' is the same as the minimum achievable by hypotheses from H.

Theorem 5.5 Let $H' \subseteq H$. If RP $\neq$ NP and it is NP-hard to approximate $\text{MinDis}(H)$, restricted to T-legal and (H, H')-indifferent samples, to within a factor $d \in \mathcal{Q}$, then the H'-degradation of H is not limited by d when learning T.

Proof. The proof is similar to that of Theorem 5.1 given in [21], but with additional complications. We show that a polynomial-time algorithm L for learning T by H, whose H'-degradation is limited by d with a polynomial sample complexity, can be converted into a polynomial-time probabilistic algorithm A for approximating $\text{MinDis}(H)$, restricted to T-legal and (H, H')-indifferent samples, to within a factor d.

Let $p(\epsilon^{-1}, \delta^{-1}, n)$ be a polynomial bounding the sample complexity of L. Let $d = d_1/d_2$, where d_1 and d_2 are positive integers. Algorithm A is given $\mathbf{z}$ and n as input, and proceeds as follows:

1. Let $\epsilon := (d_2(|\mathbf{z}|+1))^{-1}$, $\delta := 1/2$, $m := \lceil p(\epsilon, \delta, n) \rceil$, and D be the distribution defined by $D(x) \stackrel{\text{def}}{=} w(x)/|\mathbf{z}|$, where
$$w(x) \stackrel{\text{def}}{=} |\{i : \exists y.\, z_i = (x, y)\}|,$$
i.e., $w(x)$ is the number of occurrences of instance x in $\mathbf{z}$.

2. Randomly and independently select m instances x_i' from the distribution D, and let y_i' be such that (x_i', y_i') is in the sequence $\mathbf{z}$. Define $\mathbf{z}' \stackrel{\text{def}}{=} (x_1', y_1'), \ldots, (x_m', y_m')$.

3. Output $h = L(\mathbf{z}')$.

Since d_2 and δ are constants and p is a polynomial, m is bounded by a polynomial in n and rsiz($\mathbf{z}$) (the repsize of $\mathbf{z}$). Assuming we have a reasonable parameterization of the instance space (i.e., $\mathrm{rsiz}(x) = \Omega(n^a)$ for some $a > 0$ when $x \in X^{[n]}$), n is bounded by a polynomial in rsiz($\mathbf{z}$), so m is bounded by a polynomial in rsiz($\mathbf{z}$) only. Combining this with the fact that L runs in polynomial time, it is easy to see that A runs in polynomial time.

Since $\mathbf{z}$ is T-legal, there is some $f \in T$ that is consistent with $\mathbf{z}$. Combining this with the fact that x_i' is known to occur in $\mathbf{z}$, we see that y_i' in step 2 above is unambiguously defined (it is $f(x_i')$).

Let k be the minimum number of misclassifications on $\mathbf{z}$ achievable by hypotheses in H, and k' the minimum achievable by hypotheses in H'. Since $\mathbf{z}$ is (H, H')-indifferent, we have $k = k'$. In addition, for distribution D and target f we have $\mathrm{opt}(H) = \mathrm{opt}(H') = k/|\mathbf{z}|$. Since L learns T by H, with H'-degradation limited by d, we have a probability of at least $1 - \delta = 1/2$ that
$$\mathrm{err}(h) \leq d \cdot \mathrm{opt}(H') + \epsilon = dk/|\mathbf{z}| + \epsilon.$$

Suppose h misclassifies $j > dk$ examples. Since dk is a multiple of $1/d_2$, and j is an integer and hence a multiple of $1/d_2$, we have $j \geq dk + 1/d_2$. Then
$$\mathrm{err}(h) \geq dk/|\mathbf{z}| + (d_2|\mathbf{z}|)^{-1} > dk/|\mathbf{z}| + \epsilon,$$
a contradiction. Hence we conclude that, with a probability of at least 1/2, h misclassifies at most dk examples. Thus A approximates MinDis(H) to within a factor of d. □

The main result of this chapter is to demonstrate limits on the approximability of MinDis(Halfspaces) and variants. Our tool for doing this is the cost-preserving polynomial transformation (recall Definition 4.6).

Definition 5.11 Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two minimization problems. We write $\mathcal{M}_1 \leq_{pol}^{cp} \mathcal{M}_2$ to mean that there is a cost-preserving polynomial transformation (*CPPT*) from $\mathcal{M}_1$ to $\mathcal{M}_2$.

Lemma 5.6 If $\mathcal{M} \leq_{pol}^{cp} \mathcal{M}'$ and $\mathcal{M}' \leq_{pol}^{cp} \mathcal{M}''$, then $\mathcal{M} \leq_{pol}^{cp} \mathcal{M}''$.

Proof. If (t_1, t_2) is the CPPT from $\mathcal{M}$ to $\mathcal{M}'$, and (t'_1, t'_2) is the CPPT from $\mathcal{M}'$ to $\mathcal{M}''$, it is easily verified that (u_1, u_2), where $u_1(x) = t'_1(t_1(x))$ and $u_2(y, x) = t_2(t'_2(y, t_1(x)), x)$, is a CPPT from $\mathcal{M}$ to $\mathcal{M}'$. □

We are now ready to strengthen Corollary 5.4, showing that Halfspaces also has unlimited degradation. We do this by giving a cost-preserving polynomial transformation from Hitting Set to MinDis(Halfspaces).

Definition 5.12 Hitting Set.
Input: A finite set S and a collection C of nonempty subsets of S.
Output: A hitting set for C of minimum cardinality. (A hitting set is a set $R \subseteq S$ s.t. $R \cap M \neq \emptyset$ for all $M \in C$).
Note: The restriction of Hitting Set in which we require all $M \in C$ to have equal cardinality $s \geq 2$ we call **Uniform Hitting Set**.

Lemma 5.7 Hitting Set $\leq^{cp}_{pol}$ Uniform Hitting Set.

Proof. Let (S', C') be an instance of Hitting Set. Let s be the maximum of 2 and the cardinality of the largest set in C'. To every $M \in C'$ add $s - |M|$ new, distinct dummy elements, and call the result C. Add to S' all the dummy elements, and call the result S. We then define $t_1(S', C') = (S, C)$. It is clear that any hitting set for C' is also a hitting set for C, thus $t_1(S', C')$ has a solution of cost no greater than that of any solution for (S', C').

Conversely, given any hitting set R for C we can obtain a hitting set R' for C' s.t. $|R'| \leq |R|$, as follows: replace any dummy element i of R by any non-dummy element of the unique $M \in C$ s.t. $i \in M$. Thus we define $t_2(R, (S, C)) = R'$, and we see that the cost of R' is at most the cost of R. Hence (t_1, t_2) is the desired CPPT. □

Theorem 5.8 Hitting Set $\leq^{cp}_{pol}$ MinDis(Halfspaces).

Proof. From Lemmas 5.7 and 5.6 it suffices to show that Uniform Hitting Set $\leq^{cp}_{pol}$ MinDis(Halfspaces). Let (S, C) be an instance of Uniform Hitting Set, with each element of C having cardinality $s \geq 2$. WLOG we will assume that $S = \{1, \ldots, n\}$. We define example vectors of dimension sn, which should be viewed as s groups of n dimensions. We write $\mathbf{1}_{i_1,\ldots,i_p}$ for the n-dimensional vector having 1 at positions $i_1, \ldots, i_p$ and 0 at all other positions, and $\mathbf{0}$ for the n-dimensional null vector. We then define

- X_+ is the set of sn-dimensional "element vectors" $(\mathbf{1}_i, \ldots, \mathbf{1}_i)$, $1 \leq i \leq n$;
- X_- is the set of sn-dimensional "set vectors"

$$(\mathbf{1}_{i_1,\ldots,i_s}, \mathbf{0}, \ldots, \mathbf{0}), \ \ldots \ , (\mathbf{0}, \ldots, \mathbf{0}, \mathbf{1}_{i_1,\ldots,i_s})$$

for each set $M = \{i_1, \ldots, i_s\} \in C$.

- $\mathbf{z}$ is a sequence containing $(x, 1)$ for every $x \in X_+$, $(x, 0)$ for every $x \in X_-$, and no other elements.

- $t_1(S, C) = \mathbf{z}$.

Note that $|X_+| = n$ and $|X_-| = s|C|$.

Claim 1: If $R \subseteq S$ is a hitting set for C, then there exists a halfspace misclassifying $|R|$ examples from $\mathbf{z}$.

Proof of Claim 1: Given R, consider the halfspace obtained from $\theta(\mathbf{x}) \overset{\text{def}}{=} \sum_{j=1}^{s} \sum_{l \in R} -x_{j,l}$. If $\mathbf{x}$ is a set vector derived from $M \in C$ then $\theta(\mathbf{x}) = -|R \cap M|$; since $R \cap M \neq \emptyset$, we have $\theta(\mathbf{x}) < 0$. Hence all set vectors are correctly classified as negative. If $\mathbf{x}$ is an element vector derived from $i \in S$, then $\theta(\mathbf{x}) = -s$ if $i \in R$, and $\theta(\mathbf{x}) = 0$ if $i \notin R$. In the first case the vector is incorrectly classified as negative, and in the second case the vector is correctly classified as positive. This gives $|R|$ misclassifications altogether, proving Claim 1.

Thus if (S, C) has a solution of cost k, $t_1(S, C)$ has a solution of cost k.

We now define t_2. Let $(\mathbf{w}, \tau)$ be the $(sn + 1)$-dimensional weight vector defining a halfspace h. We find a corresponding hitting set $R \subseteq S$ for C as follows:

- If h misclassifies an element vector derived from $i \in S$, then insert i into R.

- If h misclassifies a set vector derived from $M \in C$, then insert an arbitrary element of M into R.

We define $t_2((\mathbf{w}, \tau), (S, C)) = R$.

Claim 2: If $(\mathbf{w}, \tau)$ defines a halfspace misclassifying k examples in $\mathbf{z} = t_1(S, C)$, then $R = t_2((\mathbf{w}, \tau), (S, C))$ is a hitting set for C with cardinality at most k.

Proof of Claim 2: It follows directly from its definition that R has at most k elements, and that it is a subset of S. Assume for the sake of contradiction that R is not a hitting set, i.e., there exists some $M \in C$ with $i \notin R$ for all $i \in M$. This implies that the s element vectors derived from the elements of M are classified as positive, leading to

$$\forall l \in M, \ \sum_{j=1}^{s} w_{j,l} \geq \tau, \text{ hence } \sum_{j=1}^{s} \sum_{l \in M} w_{j,l} \geq s\tau;$$

it also implies that the s set vectors corresponding to M are classified as negative, leading to

$$\forall 1 \leq j \leq s, \ \sum_{l \in M} w_{j,l} < \tau, \text{ hence } \sum_{j=1}^{s} \sum_{l \in M} w_{j,l} < s\tau,$$

which is a contradiction, and Claim 2 is proven.

From Claims 1 and 2, and the evident fact that t_1 and t_2 can be computed in polynomial time, we see that (t_1, t_2) is a CPPT from Uniform Hitting Set to Min-Dis(Halfspaces). □

Note that in the polynomial transformation given above, only Boolean example vectors were produced. Thus we have the following.

Definition 5.13 Boolean Halfspaces is the set of linear threshold functions restricted to Boolean vectors.

Corollary 5.9 Hitting Set $\leq_{pol}^{cp}$ MinDis(Boolean Halfspaces).

We now consider the hypothesis space of hyperplanes:

Definition 5.14 Hyperplanes is the set of Boolean-valued functions f having the form $f(\mathbf{x}) = \mathbf{true}$ if $\mathbf{w} \cdot \mathbf{x} = \tau$, and **false** otherwise, for some $\mathbf{w}$ and τ.

A hyperplane is simply the intersection of two halfspaces, and so would seem to be little more complex than a halfspace. We obtain, however, the following:

Corollary 5.10 Hitting Set $\leq_{pol}^{cp}$ MinDis(Boolean Halfspaces) restricted to Hyperplanes-legal samples.

Proof. Combine Corollary 5.9 with the fact that the positive examples in the proof of Theorem 5.8 all lie on the hyperplane $\sum_{l=1}^{n} w_{1,l} = 1$, and none of the negative examples lie on this hyperplane. $\square$

Now let us consider the hypothesis spaces of monomials and decision lists:

Definition 5.15 In the following a *literal* is an expression of either of the forms x_i or $\neg x_i$.

1. *Monomials* is the set of functions on Boolean vectors which can be expressed as the conjunction of 0 or more literals.

2. Decision Lists is the set of functions f on Boolean vectors, of the form $f(\mathbf{x}) =$ **if** l_1 **then** c_1 **else if** l_2 **then** $c_2 \ldots$ **else if** l_p **then** c_p **else** c_{p+1}, where $p \geq 0$, each c_i is either 1 or 0, and each l_i is a literal. We represent such a function by the list of pairs $(l_1, c_1) \cdots (l_p, c_p)(\mathbf{true}, c_{p+1})$.

Lemma 5.11 Monomials $\subseteq$ Decision Lists $\subseteq$ Boolean Halfspaces.

Proof. The monomial $\wedge_{i=1}^{p} l_i$ is the same as the decision list $(\neg l_1, 0) \cdots (\neg l_p, 0)(\mathbf{true}, 1)$. Thus every monomial is also a decision list.

Now consider the decision list $L = (l_1, c_1) \cdots (l_p, c_p)(\mathbf{true}, c_{p+1})$. Let $\sigma(1) = 1$ and $\sigma(0) = -1$. Noting that $\neg x_i = (1 - x_i)$, we see that each l_i is a linear function of $\mathbf{x}$. A straightforward induction on p shows that the halfspace defined by

$$\sum_{i=1}^{p} \sigma(c_i) 2^{p+1-i} l_i \geq -\sigma(c_{p+1})$$

is equivalent to L. Thus every decision list is also a Boolean halfspace. $\square$

Corollary 5.12 Hitting Set $\leq^{cp}_{pol}$ MinDis(Boolean Halfspaces) restricted to Hyperplanes-legal, (Boolean Halfspaces, Monomials)-indifferent samples.

Proof. Returning to the proof of Theorem 5.8, let the instance (S, C) of Uniform Hitting Set transform to the instance $\mathbf{z}$ of MinDis(Halfspaces). It follows from the definition of a CPPT (Definition 4.6) that if k is the cardinality of the smallest hitting set for C, then k is also the minimum number of misclassifications on $\mathbf{z}$ achievable by hypotheses from Halfspaces.

It was shown that if C has a hitting set R, then the halfspace defined by $\sum_{j=1}^{s}\sum_{l\in R} -x_{j,l} \geq 0$ misclassifies $|R|$ examples from $\mathbf{z}$. But this halfspace, when restricted to Boolean vectors, is in fact a monomial, viz., $\bigwedge_{j=1}^{s}\bigwedge_{l\in R}\neg x_{j,l}$. Thus the minimum number of misclassifications on $\mathbf{z}$ achievable by monomials is at most the minimum achievable by halfspaces. By Lemma 5.11 the minimum achievable by halfspaces is at most the minimum achievable by monomials. Thus the two minima are equal, and $\mathbf{z}$ is (Boolean Halfspaces, Monomials)-indifferent. □

We have a similar corollary for homogeneous halfspaces:

Corollary 5.13 Hitting Set $\leq^{cp}_{pol}$ MinDis(Halfspaces) restricted to Hyperplanes-legal, (Halfspaces, Homogeneous Halfspaces)-indifferent samples.

Proof. Same as the proof of Corollary 5.12, except that this time we note that the halfspace defined by $\sum_{j=1}^{s}\sum_{l\in R} -x_{j,l} \geq 0$ is homogeneous. □

We can readily extend these results to cover the degradation of Monomials, Decision Lists, and Homogeneous Halfspaces using the following lemma:

Lemma 5.14 If $H'' \subseteq H' \subseteq H$, then MinDis($H$) restricted to (H, H'')-indifferent samples $\leq^{cp}_{pol}$ MinDis(H') restricted to (H', H'')-indifferent samples.

Proof. The required cost-preserving polynomial transformation is just (t_1, t_2) where $t_1(x) = x$ and $t_2(y, x) = y$:

1. If x is an instance of MinDis(H) restricted to (H, H'')-indifferent samples then the minimum number of misclassifications achievable with H is the same as the minimum achievable with H''; and this the same as the minimum achievable with H' (since $\text{opt}(H) \leq \text{opt}(H') \leq \text{opt}(H'')$).
2. Any solution for MinDis(H') is a hypothesis from $H' \subseteq H$, and hence is also a solution for MinDis(H).

□

Recall from the end of Chapter 4 that Min Set Cover cannot be approximated to within any constant factor (in polynomial time), unless P = NP. The Hitting Set problem is isomorphic to Min Set Cover [22, 27], so these approximation limits apply also to Hitting Set. Thus we have the following results:

Corollary 5.15 The following problems cannot be approximated to within any constant factor, unless P = NP:

1. MinDis(Boolean Halfspaces), even when restricted to (Boolean Halfspaces, Monomials)-indifferent samples.
2. MinDis(Decision Lists), even when restricted to (Decision Lists, Monomials)-indifferent samples.
3. MinDis(Monomials).
4. MinDis(Halfspaces), even when restricted to (Halfspaces, Homogeneous Halfspaces)-indifferent samples.
5. MinDis(Homogeneous Halfspaces).

These results hold even when the problems are further restricted to Hyperplanes-legal samples.

Proof. (1) follows from Corollary 5.12. (2) and (3) follow from (1), Lemma 5.11, and Lemma 5.14. (4) follows from Corollary 5.13. (5) follows from (4) and Lemma 5.14. □

Corollary 5.16 Unless RP = NP,

1. Boolean Halfspaces has unlimited (Monomials-)degradation;
2. Decision Lists has unlimited (Monomials-)degradation;
3. Monomials has unlimited degradation;
4. Halfspaces has unlimited (Homogeneous Halfspaces-)degradation;
5. Homogeneous Halfspaces has unlimited degradation.

These results hold even when the problem is restricted to learning Hyperplanes.

Proof. Follows directly from Corollary 5.15 and Theorem 5.5. □

The negative results of Corollary 5.16 are rather surprising in light of the fact that there exist polynomial algorithms for learning all of the hypothesis spaces mentioned therein, under the PAC model [6, 37, 39]. By simply allowing targets that are not in the hypothesis space, we go from a tractable problem to a problem that is not only intractable, but cannot even be approximated to within any constant factor! Thus it would appear that PAO learning is much more difficult than PAC learning.

5.4 Acknowledgments

This chapter is excerpted, with modifications and some additions, from a paper on which I collaborated with Klaus-Uwe Höffgen and Han-Ulrich Simon of the University of Dortmund [21]. The original proof showing that MinDis(Halfspaces) could not be approximated to within any constant factor unless Min Set Cover could was mine. Klaus-Uwe Höffgen then modified my proof to remove some aesthetically unpleasing features of it (for example, my proof required multiple copies of some examples). I then introduced the notion of an (H, H')-indifferent sample and H'-degradation for $H' \neq H$, and further modified the proof to allow us to obtain the H'-degradation results for $H' \neq H$, and the degradation results for Monomials and Decision Lists.

Chapter 6

Decision Trees and Rule Lists

In this chapter we look at the stratified hypothesis spaces of decision trees and rule lists, as a prelude to the discussion of rule induction in Chapter 7. Decision trees and rule lists are used for classification; in Haussler's model (see Chapter 3) this means that the outcome space Y and decision space Y' are equal and finite, and the loss function most often used is $l(y', y) = (y \neq y')$ (a loss of 1 for misclassification, 0 otherwise). Heuristic algorithms for learning decision trees have been much-studied in both the machine learning [35] and applied statistics [7] communities. Decision trees have some expressive limitations, however, which can be overcome with the use of rule lists. Sections 6.1 and 6.2 define decision trees and rule lists, discuss their use, and compare their expressive power. Section 6.3 discusses the generation of synthetic test problems for evaluating heuristic learning algorithms that use these hypothesis spaces.

6.1 Decision Trees

If C is the set of possible classes and X the instance space, a decision tree defines a function $h : X \to C$ by hierarchically partitioning X and assigning each part of the resultant partition a class $c \in C$. (We hereafter use C rather than Y to denote the outcome space, to emphasize that it is a finite set of classes.)

Definition 6.1 We assume that we have a (possibly infinite) set $\mathcal{P}$ of finite partitions of the instance space X. Each $\Pi \in \mathcal{P}$ we call a *primitive partition*. Each such Π can be considered a finite set of mutually-exclusive and exhaustive predicates P on X. Each such predicate $P \in \Pi \in \mathcal{P}$ we call a *primitive test*.

Definition 6.2 A *decision tree* is a tree that is labeled as follows:

- Each leaf node is labeled by some class $c \in C$.
- Corresponding to each internal node is a partition $\Pi \in \mathcal{P}$ such that the node has $|\Pi| > 1$ children, and the branches emanating from it are labeled by the predicates $P \in \Pi$.

We write c for the single-node decision tree labeled with class c, and we write $(P_1 : T_1 \mid \cdots \mid P_k : T_k)$ for the decision tree comprised of a root node with k outgoing branches labeled $P_1, \ldots, P_k$, and coresponding subtrees $T_1, \ldots, T_k$.

Definition 6.3 We say that a leaf node of a decision tree *covers* an instance $x \in X$ if $P(x)$ holds for every predicate P labeling a branch on the path from the root to the leaf. A decision tree represents the function $h : X \to C$, where $h(x)$ is the class labeling the unique leaf node that covers x.

Definition 6.4 The *size* of a decision tree is the number of leaf nodes.

Note that, by defining the size of a decision tree, we have defined the stratified hypothesis space of decision trees: H_i is just the set of functions $f : X \to C$ which are represented by some decision tree of size i or less.

It is common in machine learning tasks for the instance space X to be the set of *attribute vectors* of a particular type. That is to say, $X = D_1 \times \cdots \times D_n$ for some sequence of domains $D_1, \ldots, D_n$. Typically, each D_i is either

- a finite set of unordered values (a *nominal attribute*),
- a finite set of fully-ordered values (a *discrete linear attribute*), or
- $\mathcal{Q}$, the set of all rational numbers (a *continuous attribute*).

In such a case the set $\mathcal{P}$ typically contains the following partitions, and no others:

- For each nominal attribute i with $\mathcal{D}_i = \{v_1, \ldots, v_k\}$, the partition $\{P_1, \ldots, P_k\}$, where $P_j(\mathbf{x}) \stackrel{\text{def}}{=} (x_i = v_j)$.
- For each discrete linear or continuous attribute i and value $v \in \mathcal{D}_i$, the partition $\{P_\leq, P_>\}$, where $P_\leq(\mathbf{x}) \stackrel{\text{def}}{=} (x_i \leq v)$ and $P_>(\mathbf{x}) \stackrel{\text{def}}{=} (x_i > v)$.

Whenever we have an instance space of attributes vectors as defined above, we will assume that the set $\mathcal{P}$ of primitive partitions is as defined above also, unless otherwise noted.

Although decision trees have found wide use in machine learning, and have even been used in commercial machine learning systems, they have some expressive limitations. It has been noted by machine-learning researchers [33] that some seemingly-simple Boolean functions apparently require large decision trees that contain many identical subtrees. For example, let $\psi(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$. For this function the instance space has 3 natural regions: those vectors covered by the first conjunct $x_1 \wedge y_1$, those vectors covered by the second conjunct $x_2 \wedge y_2$, and all remaining vectors. But one can verify by exhaustive search that the smallest decision tree representing ψ is the following tree of size 7:

$$\begin{array}{l}(x_1 : (y_1 : 1 \mid \neg y_1 : (x_2 : (y_2 : 1 \mid \neg y_2 : 0) \mid \neg x_2 : 0)) \\ \mid \neg x_1 : (x_2 : (y_2 : 1 \mid \neg y_2 : 0) \mid \neg x_2 : 0))\end{array}$$

Note that there are two identical sub-trees for $x_2 \wedge y_2$. We now prove that the obvious extension of ψ to n-bit vectors (for arbitrary n) *requires* a decision tree whose size is exponential in n.

Theorem 6.1 Let $\psi_n(\mathbf{x}, \mathbf{y}) \overset{\text{def}}{=} (x_1 \wedge y_1) \vee \cdots \vee (x_n \wedge y_n)$, where $\mathbf{x}$ and $\mathbf{y}$ are n-element Boolean vectors. Then the smallest decision tree representing ψ_n has size $2^{n+1} - 1$.

Proof. We prove the theorem by induction on n.

Base case ($n = 1$): We have $\psi_1(x, y) = (x_1 \wedge y_1)$. It is straightforward to enumerate the decision trees of size 1 or 2 and verify that none of them represents ψ_1. But the size-3 decision tree

$$(x_1 : (y_1 : 1 \mid \neg y_1 : 0) \mid \neg x_2 : 0)$$

does represent ψ_1, and we have $2^{1+1} - 1 = 3$.

Induction step ($n > 1$): Assume the theorem holds for all ψ_l, $l < n$. Let T' be a minimum-size decision tree representing ψ_n. Let $\pi_i(\mathbf{x})$ be the permutation of $\mathbf{x}$ in which x_i and x_n are exchanged, everything else remaining the same. Note that $\psi_n(\mathbf{y}, \mathbf{x}) = \psi_n(\mathbf{x}, \mathbf{y})$ and $\psi_n(\pi_i(\mathbf{x}), \pi_i(\mathbf{y})) = \psi_n(\mathbf{x}, \mathbf{y})$. Thus if we exchange the roles of $\mathbf{x}$ and $\mathbf{y}$ in T', or exchange the roles of x_i and x_n while simultaneously exchanging the roles of y_i and y_n, we obtain an equivalent tree of the same size. The root node of T' is partitioned on the value of x_i or y_i, for some i; by applying one or both of the mentioned transformations, we can then obtain from T' a minimum-sized tree T representing ψ_n, whose root node is partitioned on the value of x_n.

Let $T = (\neg x_n : T_0 \mid x_n : T_1)$. Neither T_0 nor T_1 contains any node partitioned on the value of x_n, since T is of minimum size. (If there were such a node, then the subtree rooted at that node could be replaced by one of the node's two child subtrees, making T smaller while still representing ψ_n.) Thus both T_0 and T_1 represent functions that do not depend on x_n.

T_0 represents a function g such that, when $x_n = \textbf{false}$, $g(\mathbf{x}, \mathbf{y}) = \psi(\mathbf{x}, \mathbf{y}) = \bigvee_{i=1}^{n-1}(x_i \wedge y_i)$. Since g does not depend on x_n, we have that $g(\mathbf{x}, \mathbf{y}) = \bigvee_{i=1}^{n-1}(x_i \wedge y_i)$ for all $\mathbf{x}$ and $\mathbf{y}$. T_0 is a minimum-size decision tree representing g; by the induction hypothesis, we then know that T_0 has size $2^n - 1$.

T_1 represents a function h such that, when $x_n = \textbf{true}$, $h(\mathbf{x}, \mathbf{y}) = \psi(\mathbf{x}, \mathbf{y}) = y_n \vee g(\mathbf{x}, \mathbf{y})$. Since neither h nor g depends on x_n, we have that $h(\mathbf{x}, \mathbf{y}) = y_n \vee g(\mathbf{x}, \mathbf{y})$ for all $\mathbf{x}$ and $\mathbf{y}$. The function h can be represented by the decision tree

$$(y_n : 1 \mid \neg y_n : T_0)$$

since T_0 represents g. This is a tree of size $(2^n - 1) + 1 = 2^n$. Since T_1 is a minimum-sized tree representing h, its size is then at most 2^n.

Let T_1' be the tree obtained from T_1 by repeatedly choosing any node partitioned on the value of y_n and replacing the subtree rooted at that node with its child subtree corresponding to the test $y_n = \textbf{false}$. T_1' represents a function h' that does not depend on y_n and is identical to h when $y_n = \textbf{false}$, i.e., T_1' represents g. By the induction hypothesis, the size of T_1' is at least $2^n - 1$. But T_1' is smaller than T_1, since h depends on y_n and hence T_1 has at least one node partitioned on the value of y_n. Thus the size of T_1 is at least $(2^n - 1) + 1 = 2^n$. Combining this with the previous paragraph, the size of T_1 is exactly 2^n.

The size of T is just the sum of the sizes of T_0 and T_1, i.e., $(2^n - 1) + 2^n = 2^{n+1} - 1$. $\square$

6.2 Rule Lists

Each leaf of a decision tree may be thought of as expressing the rule "if $\mathbf{t}$ then output class c", where c is the class labeling the leaf and $\mathbf{t}$ is the conjunction of all tests labeling branches on the path from the root to the leaf. Thus each decision tree may be thought of as a collection of such rules. The function ψ_n of Theorem 6.1 may also be expressed by a set of rules: we have the rules "if $x_i \wedge y_i$ then output **true**" for each i, $1 \leq i \leq n$, and a *default rule* "output **false**" to be used if none of the other rules apply. But the structure of a decision tree imposes a rigid structure on the set of rules expressed by the tree, which causes problems for representing functions such as ψ_n. This has motivated some machine learning researchers [10, 35, 37, 44] to use lists of rules themselves to represent hypotheses. A question that immediately arises is how to evaluate the application of a rule list to an instance if more than one rule applies. The simplest answer is to use the first rule that applies. We now define these notions more precisely.

Definition 6.5 A *rule* is a pair $(\mathbf{t}, c)$, where $c \in C$ and $\mathbf{t}$ is a conjunction of primitive tests. $\mathbf{t}$ is called the *precondition* of the rule, and c its *output class*.

Definition 6.6 A *rule list* is a nonempty sequence of rules

$$(\mathbf{t}_r, c_r) \ldots (\mathbf{t}_1, c_1)(\mathbf{t}_0, c_0),$$

with $\mathbf{t}_0 = \mathbf{true}$. It represents the function $h : X \to Y$ such that $h(x) = c_i$ if $\mathbf{t}_i(x)$ is true and $\mathbf{t}_j(x)$ is false for all $j > i$; in this case we say that rule i *captures* instance x. The final rule of a rule list is called the *default rule*.

Note that, for a rule list of r (non-default) rules, rule r is the *first* rule, rule 1 is the *last* non-default rule, and rule 0 is the default rule.

Definition 6.7 Let ς be a positive number to be fixed later, which we will call the *start size*. The *size* of a rule is ς plus the number of primitive tests in the precondition of the rule. The *size* of a rule list is the sum of the rule sizes, i.e., $r\varsigma + p$, where r is the number of rules and p the total number of primitive tests.

As with decision trees, by defining the size of a rule list we have defined a stratified hypothesis space of rule lists. The exact stratification is determined by the start-size parameter ς, which lets us choose a relative weighting between the number of rules and the total number of primitive tests in determining the size of a rule list. If we use some variant of Occam's Razor for learning, for example, using the results on loose

ERM algorithms of Chapter 3 as a guide, then different values of ς define different learning biases.

It is evident that the function ψ_n of Theorem 6.1 can be represented by a rule list of $n+1$ rules (including the default) with a total of $2n$ primitive tests — a far cry from the $2^{n+1}-1$ leaves needed to represent ψ_n with a decision tree. But to argue that we have gained in expressive power by using rule lists, we must show that for any decision tree there is an equivalent rule list that is not too much "larger". Corresponding to each leaf of a decision tree is a rule whose precondition is the conjunction of the primitive tests on the path from the root to the leaf, and whose output class is the class labeling the leaf. If our instances are attribute vectors, as discussed in Section 6.1, then some of the primitive tests may be superflous — e.g., if we have the test "$x_i \leq 4$" then we can dispense with the test "$x_i \leq 5$". After removing any superfluous tests, the rule corresponding to a leaf will have at most $N+2L$ primitive tests, where N is the number of nominal attributes and L is the number of linear attributes. If we put these rules together into a rule list, replacing the precondition of the last rule with **true**, we obtain a rule list equivalent to our tree, with s rules and at most $(N+2L)(s-1)$ primitive tests, where s is the size of the tree.

The bound of $(N+2L)(s-1)$ primitive tests is not the best we can do, however; we can construct an equivalent rule list using s rules and at most $s\log_4 s$ primitive tests, and the construction does not assume that instances are attribute vectors. For all but small values of N and L or very large values of s, this $s\log_4 s$ bound is an improvement over $(N+2L)(s-1)$. We prove this result beginning with a technical lemma.

Lemma 6.2 Let $1 \leq u \leq v$ and $1 < b \leq 4$. Then $u + u\log_b u + v\log_b v \leq (u+v)\log_b(u+v)$.

Proof. We reduce the the problem as follows:

$$\begin{aligned}
(u+v)\log_b(u+v) &= u\log_b(u+v) + v\log_b(u+v) \\
&= u\log_b\left(u\frac{u+v}{u}\right) + v\log_b\left(v\frac{u+v}{v}\right) \\
&= u\log_b u + v\log_b v + u\log_b\frac{u+v}{u} + v\log_b\frac{u+v}{v};
\end{aligned}$$

thus it suffices to show that $y \geq u$, where

$$y \overset{\text{def}}{=} u\log_b\frac{u+v}{u} + v\log_b\frac{u+v}{v}.$$

If $v = u$, then $y = 2u\log_b 2$; but $b \leq 4$ implies that $\log_b 2 \geq 0.5$, hence $y \geq u$. We show that $y \geq u$ still holds for $v > u$ by showing that $dy/dv > 0$:

$$\begin{aligned}
dy/dv &= \frac{u}{\ln b}\left(\frac{u}{u+v}\right)\frac{1}{u} + \frac{v}{\ln b}\left(\frac{v}{u+v}\right)\left(\frac{-u}{v^2}\right) + 1\cdot\log_b\frac{u+v}{v} \\
&= \frac{u}{(u+v)\ln b} - \frac{u}{(u+v)\ln b} + \log_b\frac{u+v}{v} \\
&> 0.
\end{aligned}$$

□

Theorem 6.3 For each decision tree T of size s there is an equivalent rule list with s rules and a total of at most $\lfloor s \log_4 s \rfloor$ primitive tests.

Proof. We prove the theorem by induction on s.

Base case ($s = 1$): T consists of a single node labeled by some class c. The equivalent rule list is $(\mathbf{true}, c)$, which has 1 rule and 0 primitive tests, trivially satisfying the theorem.

Induction step ($s > 1$): Assume the theorem holds for all trees of size less than s. Let $K \geq 2$ be the number of children of the root node of T. Let $T_1, \ldots, T_K$ be the subtrees rooted at the children of T's root node, in order of decreasing size, and let $s_1, \ldots, s_K$ be the sizes of the corresponding trees. Thus we have $1 \leq s_i \leq s_j$ for all $i > j$, and $\sum_{i=1}^{K} s_i = s$. By the induction hypothesis, for each T_i there is an equivalent rule list L_i with s_i rules and a total of at most $\lfloor s_i \log_4 s_i \rfloor$ primitive tests. We write $(\mathbf{t}_{i,j}, c_{i,j})$ for rule j of L_i.

Let P_i be the primitive test labeling the branch leading from the root of T to subtree T_i. We then define L'_i to be the sequence of rules obtained from L_i by replacing each rule precondition $\mathbf{t}_{i,j}(x)$ with $\mathbf{t}_{i,j}(x) \wedge P_i(x)$. Note that L'_i is not a rule list, because its final rule no longer has the trivial precondition **true**. Finally, we define

$$L \overset{\text{def}}{=} L'_K L'_{K-1} \cdots L'_2 L_1.$$

L *is* a rule list. In addition, it is equivalent to T. To see this, compare what happens when we apply L and T to an instance x. Let a be such that $P_a(x)$ holds and $P_i(x)$ does not for $i \neq a$. Such an a exists because the P_i define a partition of the instance space X. Let b be the rule of L_a that captures x when L_a is applied to x. Then $T(x) = T_a(x) = L_a(x) = c_{a,b}$. Now consider what happens when L is applied to x. Since $P_i(x)$ is false for $i \neq a$, the precondition of every rule in every L'_i, $i > a$, is false. Suppose $a < K$. The j-th rule in L'_a has precondition $P_a(x) \wedge \mathbf{t}_{a,j}(x)$. Since $\mathbf{t}_{a,j}(x)$ is false for $j > b$ and true for $j = b$, rule b of L'_a captures x. Similarly, if $a = K$ then rule b of L_a captures x. So $L(x) = c_{a,b} = T(x)$.

Now we consider the number of rules and primitive tests in L. Clearly, L has $\sum_{i=1}^{K} s_i = s$ rules. Let p be the number of primitive tests in L. In addition to the primitive tests in each L_i, L also has s_i occurrences of the primitive test P_i, for each $i > 1$. Thus

$$p \leq \sum_{i=2}^{K} s_i + \sum_{i=1}^{K} s_i \log_4 s_i.$$

Since $1 \leq s_{i+1} \leq s_i$ for all i, repeated application of Lemma 6.2 with $u = s_{i+1}$ and $v = \sum_{j=1}^{i} s_j$ for $i = 1, 2, \ldots, K-1$ gives us

$$\sum_{i=2}^{K} s_i + \sum_{i=1}^{K} s_i \log_4 s_i \leq s \log_4 s,$$

and hence $p \leq s \log_4 s$. But p is an integer, so $p \leq \lfloor s \log_4 s \rfloor$, and the theorem is proven. □

6.3 Generating Test Problems

It is not known whether there exists any polynomial algorithm for learning the stratified hypothesis spaces of decision trees or rule lists, under the PAC model or PAO model. The results of Chapter 5 are certainly enough to provoke some pessimism as to the possibility of finding interesting PAO learnability results. Still, it is worthwhile to look at heuristic approaches for learning using these hypothesis spaces; we may find algorithms which, even if it is known that their worst-case behavior is terrible, have acceptable performance in some "average-case" sense.

Heuristic learning algorithms, like heuristic optimization algorithms in general, are usually evaluated and compared empirically. The usual procedure is to collect sets of examples from various "real-world" or somewhat artificial, but standard, learning problems and compare performance on these sets of examples. But it may be difficult to interpret the results of such a comparison, because little is known about the structure of the particular problem. For example, it is generally not known what the minimum achievable error is, nor how large of a hypothesis is required to approach this minimum error, etc. This difficulty may be overcome by the use of synthetic data sets. The idea is to have a parameterized procedure for randomly generating target distributions $\mathcal{D}$ over $X \times C$. The procedure itself defines a probability distribution over target distributions. Once a target distribution has been generated, one can then generate independent training and test samples by drawing examples randomly and independently from the target distribution. The learning algorithms to be compared are then given the training sample as input, and the hypotheses they output are compared on the test sample. By iterating this procedure, each time randomly choosing a new target distribution, and averaging the results, we can then estimate the expected error of the learning algorithms for the particular parameter setting we have chosen.

In the remainder of this chapter we look at some methods for randomly generating target distributions. Each of these methods uses an instance space of the form $X = D^n$, where D is a domain of nominal, discrete linear, or continuous values. Each of these methods takes the following parameters:

- n (the number of attributes);
- k (the number of classes);
- η (the noise level);
- m (the size of training sample);
- m' (the size of test sample).

Each method also takes one or more additional parameters depending on the method. In outline, they all work as follows:

1. Randomly generate a target function h.
2. Construct a probability distribution $\mathcal{D}_X$ over X.

3. Construct a distribution $\mathcal{D}$ over $X \times C$ by adding a level η of uniform noise. Specifically, the probability of (x, c) under $\mathcal{D}$ is pq, where p is the probability of x under $\mathcal{D}_X$, $q = 1 - \eta$ if $c = h(x)$, and $q = \eta/(k-1)$ if $c \neq h(x)$.

4. The m training examples and m' testing examples are randomly and independently selected from the distribution $\mathcal{D}$.

Note that the target h has an error of η on the distribution $\mathcal{D}$, and that this is the minimum error achievable by any hypothesis.

For the methods described below, the target function h is either a decision tree or a rule list. We try to make it so that the subset of X captured by each rule (or covered by each leaf) has roughly the same probability as the subset captured by any other rule (or leaf). This is an attempt to make it unlikely that there exists a hypothesis with near-optimal error that is significantly smaller than h.

6.3.1 BRGEN0

BRGEN0 generates a rule list for the target hypothesis h, with the instance space $X \overset{\text{def}}{=} \{0,1\}^n$, and primitive tests of the form x_i and $\neg x_i$. BRGEN0 takes additional parameters r and l, which specify the number of non-default rules and total number of literals h must have. The hypothesis h is generated as follows:

1. Classes are (nearly) evenly distributed among the $r+1$ rules. Each class is used between $\lfloor (r+1)/k \rfloor$ and $\lceil (r+1)/k \rceil$ times, with the $(r+1) \bmod k$ classes that occur an extra time being chosen randomly without replacement from a uniform distribution over C. The assignment of classes to rules is obtained by initializing an array of length $r+1$ with the appropriate numbers of each class, then randomly permuting the array, with all permutations being equally likely.

2. The literals are randomly chosen by selecting l pairs (i, j) randomly without replacement from a uniform distribution over $\{1, \ldots, r\} \times \{1, \ldots, n\}$. For each pair (i, j) we add either x_j or $\neg x_j$ to the precondition of rule i, the choice being made at random with equal probabilities.

3. Rule lists with superfluous rules are disallowed. A rule is superfluous if it captures no $x \in X$. Superfluous rules are disallowed by repeatedly generating h as described in the previous steps, until a hypothesis with no superfluous rules is generated.

Let $\mathbf{t}_i$ be the precondition of rule i of h, and let W_i be the set of $\mathbf{x} \in X$ captured by rule i. The distribution $\mathcal{D}_X$ is defined by the following two properties:

1. The rules are equally likely to be chosen, i.e. if $\mathbf{x}$ is randomly selected according to $\mathcal{D}_X$ then $\mathbf{Pr}[\mathbf{x} \in W_i] = 1/(r+1)$ for all i.

2. The distribution is uniform "within" each rule, i.e. for each i the elements of W_i are equally likely.

```
for ($i$ := 0 to $r$) $R[i] := \{t_i\}$
for ($i$ := 1 to $r$)
   for ($j$ := 0 to $r - 1$)
      $R[j] := \bigcup_{t \in R[j]} \text{AnotB}(t, t_i)$
```

Figure 6.1: Computation of the sets $R[i]$

```
**AnotB**$(t, u)$ :
if (there is a literal $l$ in $t$ with $\neg l$ in $u$) return $\{t\}$
$S := \emptyset$
for (each literal $l$ in $u$ but not $t$)
   $S := S \cup \{t \wedge \neg l\}$; $t := t \wedge l$
return $S$
```

Figure 6.2: Computation of AnotB

To implement $\mathcal{D}_X$ we first produce a disjunctive normal form expression E_i for each rule i. E_i holds for an instance $\mathbf{x}$ iff rule i captures $\mathbf{x}$. E_i has the form $\bigvee_{j=1}^{l_i} t_{i,j}$, where the $t_{i,j}$ are mutually exclusive conjunctions of primitive tests and l_i is the number of terms in the disjunction. Thus the $t_{i,j}$, $1 \leq j \leq l_i$, partition the set of instances captured by rule i. Note that rule i is superfluous iff E_i is the empty disjunction (i.e., **false**). Corresponding to each $t_{i,j}$, we compute

- $\text{cnt}[i,j] \overset{\text{def}}{=}$ the number of $\mathbf{x} \in X$ for which $t_{i,j}(\mathbf{x})$ holds; and
- $\text{sum}[i,j] \overset{\text{def}}{=} \sum_{a=1}^{i} \text{cnt}[a,j]$.

To generate an instance $\mathbf{x}$, we first randomly choose a number i from 0 to r. We then randomly choose a number p in the range 0 to $\text{cnt}[l_i] - 1$, and choose $j = \min\{j' : \text{sum}[i,j'] > p\}$. This ensures that the parts $t_{i,j}$ are chosen with probability proportional to the number of instances they contain. Since $t_{i,j}$ is a conjunction of literals, it specifies the values of certain entries of $\mathbf{x}$; these are fixed at the required values, and the remaining entries of $\mathbf{x}$ are chosen independently and randomly.

Let t_i be the precondition of rule i. If t and u are conjunctions of primitive tests, let AnotB(t, u) be a set of conjunctions that partition $t \wedge \neg u$. Let $R[i] \overset{\text{def}}{=} \{t_{i,j} : 1 \leq j \leq l_i\}$. Figure 6.1 gives the algorithm to compute the sets $R[i]$, and Figure 6.2 gives the algorithm to compute AnotB.

There is one problem with the above scheme: the number of parts $t_{i,j}$ can grow exponentially with the number of rules, since each time the last line of Figure 6.1 is executed the size of $R[j]$ can grow by a factor of k_i, where k_i is the number of literals in t_i. The problem is only exacerbated when one tries to extend this approach to nominal attributes with more than two possible values. This difficulty motivated

```
rtree($s, A$):
if ($s = 1$) return leaf
if ($s = 2$) return mktree(rand($A$), leaf, leaf)
$(l, u)$ := setBounds($s, |A|$)
$ls$ := rand($\{l, \ldots, u\}$); $rs := s - ls$
$i$ := rand($A$); $A' := A - \{i\}$
return mktree($i$, rtree($ls, A'$), rtree($rs, A'$))
```

Figure 6.3: rtree

the development of BRGEN1, described below. Nevertheless, I have successfully used BRGEN0 with r (number of non-default rules) as high as 13, without execution times of more than a few minutes.

6.3.2 BTGEN

BTGEN generates a decision tree for the target hypothesis h, with the instance space $X \overset{\text{def}}{=} \{0,1\}^n$, and primitive partitions of the form $\{x_i, \neg x_i\}$. BTGEN takes the additional parameter s, which specifies the size of h. The structure of h (the decision tree with class labels omitted) is generated by calling rtree($s, \{1, \ldots, n\}$), where the procedure rtree is given in Figure 6.3.

Mktree(i, t_1, t_2) returns the decision tree whose root node is partitioned on x_i, with left subtree t_1 and right subtree t_2. For any finite set F, rand(F) returns an element of F randomly selected from the uniform distribution. We write leaf for an unlabeled leaf node. SetBounds() computes the minimum and maximum values of ls for which $1 \leq ls, s - ls \leq 2^{|A|-1}$. Choosing ls within these bounds ensures that each subtree has at least one node, and that we won't run out of attributes ($A = \emptyset$) on a recursive call of rtree.

Leaf nodes are labeled as follows. If the leaf node is not the right sibling of a pair of sibling leaf nodes then its class is randomly chosen from a uniform distribution over C. Otherwise the leaf node's class is randomly chosen from a uniform distribution over $C - \{c\}$, where c is the class of its sibling leaf node. (There is no point in having two sibling leaf nodes with the same class, as the tree could be simplified by deleting them and labeling their parent with the common class.) In addition, labelings in which some class labels more than twice as many leaves as another class are disallowed. This is implemented by generating a labeling, testing to see if it is disallowed, and if so repeating the process.

Let W_i be the set of $\mathbf{x} \in X$ covered by leaf i. The distribution $\mathcal{D}_X$ is defined by the following two properties:

1. The leaves are equally likely to be chosen, i.e. if $\mathbf{x}$ is randomly selected according to $\mathcal{D}_X$ then $\mathbf{Pr}[\mathbf{x} \in W_i] = 1/s$ for all i.

2. The distribution is uniform "within" each leaf, i.e. for each i the elements of W_i are equally likely.

The implementation is as follows. Randomly and uniformly choose an integer i between 1 and s. The instance $\mathbf{x}$ is then chosen among those covered by leaf i. Each primitive test on the path from the root to the leaf is of the form $x_j = v$ for some $1 \leq j \leq n$ and $v \in \{0, 1\}$. Fix the corresponding bits of $\mathbf{x}$ at the required values, and randomly and independently choose the remaining bits of $\mathbf{x}$.

6.3.3 BRGEN1

BRGEN1 generates a rule list for the target hypothesis h, with the instance space $X \overset{\text{def}}{=} \{0,1\}^n$ and primitive tests of the form x_i and $\neg x_i$. BRGEN1 takes an additional parameter r, which specifies the number of non-default rules h must have. The instance distribution $\mathcal{D}_X$ is just the uniform distribution on X. The target hypothesis h is generated as follows:

1. The output classes of the rules are generated as for BRGEN0.
2. Let $a_i \overset{\text{def}}{=} \log_2(i+1)$, $l_i \overset{\text{def}}{=} \lfloor a_i \rfloor$, and $p_i \overset{\text{def}}{=} a_i - l_i$. Then rule i's precondition has l_i literals (with probability $1 - p_i$) or $l_i + 1$ literals (with probability p_i). This gives an expected value of a_i literals, and a geometric mean of $1/(i+1)$ of the instance space covered by the rule's precondition. The attributes to test are chosen by randomly permuting the list of attributes and choosing the first l_i (or $l_i + 1$) of them. The test values ($x_i = 0$ or $x_i = 1$) are chosen at random, with 0 and 1 equally probable.

This generation method is intended to make each class roughly equally probable, and make each rule capture a roughly equal portion of the instance space. Those instances that are not covered by rules $i+1$ through r should be equally divided among the remaining $i+1$ rules; thus rule i should cover a fraction $1/(i+1)$ of the instance space not covered by rules $i+1$ through r. This is approximated by having rule i cover a fraction $1/(i+1)$ of the entire instance space, and relying on the fact that the rule preconditions are independently generated.

6.3.4 NRGEN

NRGEN is an extension of BRGEN1 to nominal attributes with more than two possible values. NRGEN takes the additional parameters r (number of non-default rules) and V (number of possible values of an attribute). The instance space is $\{1, \ldots, V\}^n$. The primitive tests are of the form $x_i = v$, for $1 \leq i \leq n$ and $1 \leq v \leq V$.

A problem with directly applying the method used with BRGEN1 is that too few primitive tests are used. If rule i ($i > 0$) has an expected $a_i = \log_V(i+1)$ primitive tests, so that it covers a geometric mean of $1/(i+1)$ of the instance space, then we will have $a_i < 1$ for $i < V - 1$, giving a positive probability that the rule has the precondition **true**. Furthermore, the expected number of primitive tests in a rule precondition increases rather slowly — only logarithmically in the rule number. For example, if $V = 5$, then $a_i < 2$ until $i \geq 24$. We need to increase the number of primitive tests for a rule somehow, but this raises a problem of its own: the fraction of the input space covered by non-default rules is thereby decreased, possibly leading to most instances being captured by the default rule. The method we describe in the following paragraphs handles this problem as well, by having $\mathcal{D}_X$ be a non-uniform distribution.

Let $\alpha > 0$ be a number to be determined later. The target rule list h is generated as follows:

1. The output classes of the rules are generated as for BRGEN0.
2. Letting $a_i \overset{\text{def}}{=} \log_V(\alpha + i)$, rule i has either $\lfloor a_i \rfloor$ or $\lceil a_i \rceil$ primitive tests, with the respective probabilities chosen to give an expected value of a_i. The attributes to test are chosen randomly without replacement from a uniform distribution over $\{1, \ldots, n\}$. The value v for an attribute test $x_j = v$ is chosen randomly from a uniform distribution over $\{1, \ldots, V\}$.

We choose α such that the total number of primitive tests is the same as for BRGEN1. Define

$$L \overset{\text{def}}{=} \sum_{i=1}^{r} \log_2(i+1);$$

L is the expected total number of primitive tests that BRGEN1 produces. We want

$$f(\alpha) \overset{\text{def}}{=} \sum_{i=1}^{r} \log_V(i+\alpha) = L.$$

Note that f is a strictly increasing and convex-$\cap$ function. Thus, once we find values α_0, α_1 such that $f(\alpha_0) \leq L$ and $f(\alpha_1) \geq L$, we can use bisection (the continuous version of binary search) to find that α for which $f(\alpha) = L$, to within a precision of $(f(\alpha_1) - f(\alpha_0))/2^N$ when N iterations are used. Since $V > 2$ we have $\log_V(i+\alpha) < \log_2(i+\alpha)$, thus $f(1) < L$, and so we choose

- $\alpha_0 = 1$.

Note that $f(\alpha) \geq r \log_V(1+\alpha)$. Then to obtain $f(\alpha_1) \geq L$ it suffices that $r \log_V(1+\alpha_1) = L$; solving for α_1, we get

- $\alpha_1 = V^{L/r} - 1$.

We now consider what fraction of the instance space is captured by each rule. We make use of the following theorem.

Theorem 6.4 Suppose that, for each $i > 0$, rule i covers a fraction $1/(i+\alpha)$ of the instance space not covered by rules $i+1$ through r. Then each non-default rule captures an equal fraction $1/(r+\alpha)$ of the instance space.

Proof. We prove the theorem by downward induction on i, the rule number.

Base case ($i = r$): Rule r covers a fraction $1/(r+\alpha)$ of the instance space; since there are no rules preceding rule r, it also captures this same fraction.

Induction step ($0 < i < r$): Assume that rules $i+1$ through r each capture a fraction $1/(r+\alpha)$ of the instance space. Then together they capture a total fraction $(r-i)/(r+\alpha)$ of the instance space. Rule i then captures a fraction

$$\left(1 - \frac{r-i}{r+\alpha}\right)\left(\frac{1}{i+\alpha}\right) = \left(\frac{i+\alpha}{r+\alpha}\right)\left(\frac{1}{i+\alpha}\right) = \frac{1}{r+\alpha}.$$

□

As with BRGEN1, we use the facts that rule i covers a geometric mean fraction $1/(i+\alpha)$ of the total instance space, and that the rules are generated independently, as a substitute for rule i covering a fraction $1/(i+\alpha)$ of the instance space not covered by rule $i+1$ through r.

Assuming that each of the r non-default rules captures a fraction $1/(r+\alpha)$ of the input space, this leaves a fraction $\alpha/(r+\alpha)$ of the input space captured by the default rule. In order to make all rules equiprobable, we then use an instance distribution $\mathcal{D}_X$ that is uniform within the set of instances captured by the default rule, and uniform within the set of instances captured by non-default rules, but assigns different probabilities to the two types of instances. Since the default rule captures a factor α more instances than a non-default rule, the instances captured by α must be a factor α less probable than those captured by non-default rules. Thus we use the following procedure to generate instances:

1. Randomly select an instance $\mathbf{x}$ from the uniform distribution over X.
2. If $\mathbf{x}$ is captured by a non-default rule, then exit outputting $\mathbf{x}$.
3. If $\mathbf{x}$ is captured by the default rule, then, with probability $1/\alpha$, exit outputting $\mathbf{x}$.
4. Go to 1.

On each iteration of the above procedure, the probability of exiting is

$$\frac{r}{r+\alpha} + \frac{1}{\alpha} \cdot \frac{\alpha}{r+\alpha} = \frac{r+1}{r+\alpha}.$$

Thus the procedure iterates an average of $I \stackrel{\text{def}}{=} (r+\alpha)/(r+1)$ times per instance generated. How does this grow as a function of V and r? We can't write a closed-form expression for I in terms of V and r, since we don't have a closed form for α. However, we can bound I. In computing α by bisection we used $V^{L/r} - 1$ as an upper bound on α. But from the definition of L it is clear that $L/r \leq \lg(r+1)$. Thus α is at most $V^{\lg(r+1)} - 1$, which can be rewritten as $(r+1)^{\lg V} - 1$, so $I < (r+1)^{\lg V - 1} + 1$. This is only mildly superpolynomial in r and V. In addition, we can compute α given V and r. Figure 6.4 plots the computed value of I for each value of V from 3 to 10 and each value of r from 1 to 50. Note that, even though I may grow faster than we would like, it is not impractically large for even the worst case computed ($V = 10$ and $r = 50$).

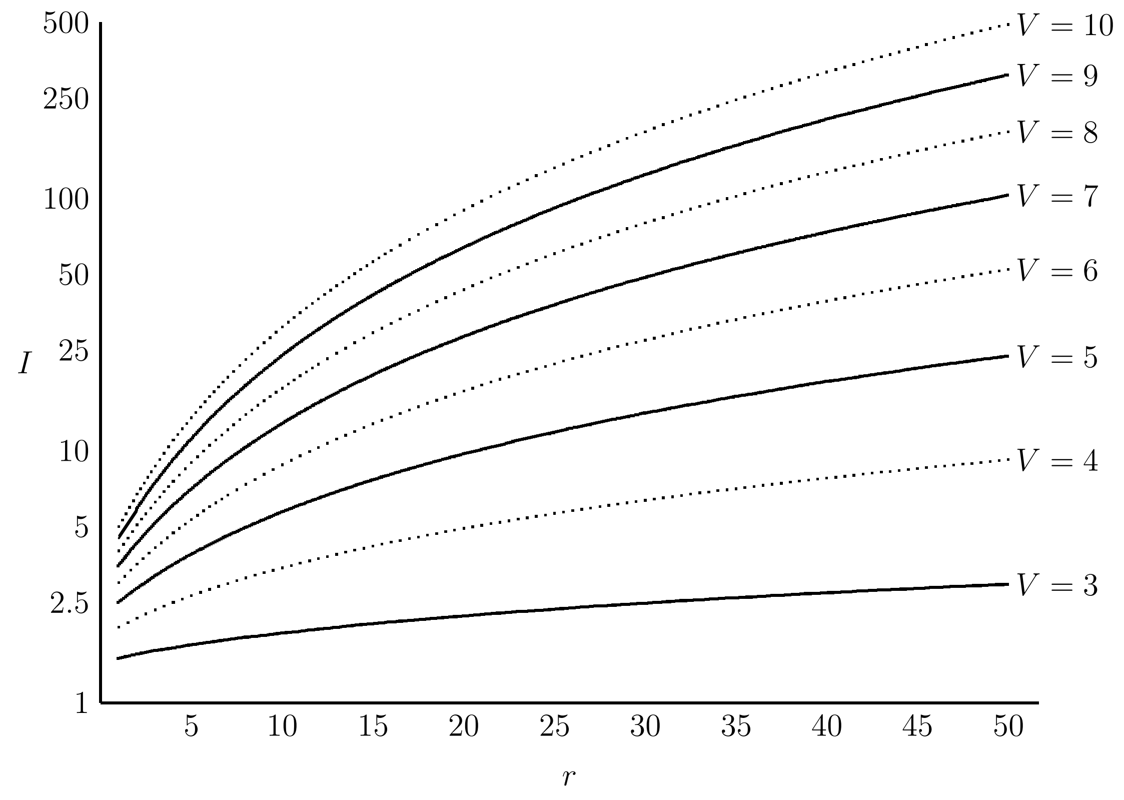

Figure 6.4: I as a function of r.

6.3.5 LRGEN

LRGEN may be thought of as a modification of BRGEN1 for discrete linear attributes. It takes the additional parameters r (number of non-default rules) and V (number of possible values for an attribute). The instance space is $X \overset{\text{def}}{=} \{1, \ldots, V\}^n$. The instance distribution $\mathcal{D}_X$ is just the uniform distribution on X. The primitive tests have the form $x_i \leq \theta$ or $x_i > \theta$ for $1 \leq i \leq n$ and $1 \leq \theta < V$ (a test $x_i \leq V$ would be satisfied by *all* elements of X). In generating the target hypothesis h, LRGEN chooses the rule classes, number of attributes to test for a rule, and which attributes to test, in exactly the same way as BRGEN1.

To generate the actual primitive tests in a rule's precondition, LRGEN first generates for each attribute j to be tested an integer φ_j such that $1 \leq \varphi_j \leq V - 1$. Then with equal probabilities LRGEN chooses the test $x_j \leq \varphi_j$ or $x_j > V - \varphi_j$. Thus we never have a two-sided test (i.e., $\theta_0 < x_j \leq \theta_1$), which is an unfortunate limitation, but it does considerably simplify the process of generating the rule list h.

We now discuss the choice of the φ_j. Let A be the set of attributes to be tested in the precondition of rule i. The test on attribute j covers a fraction φ_j/V of the instance space. For the same reasons as discussed with BRGEN1, we would like the precondition of rule i to cover a fraction $1/(i+1)$ of the instance space. Thus we would like $\prod_{j \in A}(\varphi_j/V) = 1/(i+1)$. This won't, in general, be possible, due to the fact that the φ_j take on only a finite set of values. Instead we choose the φ_j randomly and independently, with an expected value of Φ_j for φ_j, and enforce $\prod_{j \in A}(\Phi_j/V) = 1/(i+1)$. This assures that the geometric mean fraction of the instance space covered is $1/(i+1)$. To keep the variance low, we choose φ_j to be either $\lfloor \Phi_j \rfloor$ or $\lceil \Phi_j \rceil$, with the respective probabilities chosen to achieve an expected value of Φ_j.

Now only the choice of the Φ_j remains. Note that

$$\prod_{j \in A} \frac{\Phi_j}{V} = \frac{1}{i+1} \Leftrightarrow \sum_{j \in A} \log \Phi_j = |A| \log V - \log(i+1) \overset{\text{def}}{=} \Lambda.$$

Let $\Phi_j = \exp(\lambda_j)$. If $A = \{j\}$ then we simply set $\lambda_j = \Lambda$. Otherwise $|A| > 1$, and we randomly choose the vector $(\lambda_j)_{j \in A}$ from the uniform distribution over the set of such vectors satisfying $\sum_{j \in A} \lambda_j = \Lambda$ and $0 \leq \lambda_j \leq \log(V-1)$ for all j.

Let j_0 be any element of A, and let $A_0 \overset{\text{def}}{=} A - \{j_0\}$. Figure 6.5 gives the procedure that implements the above random choice of $(\lambda_j)_{j \in A}$. One might be concerned about the repeat-until loop of Figure 6.5; how many times, on average, does this loop iterate? I answered this question experimentally for all values of V from 3 to 16, and all values of i from 1 to 100. For each pair of values (V, i) I ran the above procedure 10,000 times and averaged the number of iterations of the repeat-until loop. This average generally increased as V or i increased, but never exceeded 30.

6.3.6 Irrelevant Attributes

In addition to what I have described, BRGEN1, NRGEN and LRGEN have an additional parameter $\tilde{n}$, which is the number of *relevant* attributes. If $\tilde{n} \neq n$, the only

```
repeat
    $s := 0$
    for $j \in A_0$ do
        randomly choose $\lambda_j$ from the interval $[0, \log(V-1)]$
        $s := s + \lambda_j$
until $\Lambda - \log(V-1) \leq s \leq \Lambda$
$\lambda_{j_0} := \Lambda - s$
```

Figure 6.5: Choice of $(\lambda_j)_{j \in A}$

thing that changes is the generation of the target rule list. In this case the problem generators first proceed as if there were only $\tilde{n}$ attributes; then $n - \tilde{n}$ additional, irrelevant attributes are added, and the set of all attributes is randomly permuted, so that the irrelevant attributes are randomly scattered throughout the attribute vector.

Chapter 7

Rule Induction

In this chapter I describe a heuristic approach to rule induction (learning using rule lists as the hypothesis space) called BBG. The approach uses a combination of greedy techniques (successively insert the best new rule into the existing rule list) with branch-and-bound techniques (to find the best new rule); hence the acronym "BBG". BBG takes an Occam's Razor approach to learning, in that it attempts to keep the rule-list size low. It also naturally handles noisy or stochastic learning situations. The current version of BBG is called BBG6.

An advantage of the BBG approach is that it allows the use of arbitrary loss functions. Other algorithms for tree induction or rule induction, such as CART [7], C4.5 [35], CN2 [9, 10] or PVM [44], are restricted to using the misclassification loss or a limited variation of it having the form $l(y', y) = w(y) \cdot (y' \neq y)$.

7.1 Overview of BBG

The BBG approach to rule induction consists of two independent parts:

1. An algorithm GREEDY which produces a sequence of rule lists of increasing size and decreasing empirical risk.

2. A method for trading off empirical risk and size to choose one of the rule lists output by GREEDY. (As we saw in Chapter 2, there is a tendency for learning algorithms to fit noise or statistical flukes of the data when excessively complex hypotheses are allowed.)

We now consider these separately.

7.1.1 Algorithm GREEDY

Algorithm GREEDY, shown in Figure 7.1, works by succesively inserting new rules into the current rule list. The rule to insert is chosen to maximize the ratio (decrease in empirical risk) / (increase in hypothesis size). We call this the *gain-cost ratio.*

```
**GREEDY**($\mathbf{z}, S$):
$i := -1$
$h := (\mathbf{true}, c)$, where $c$ is the most common class in $\mathbf{z}$
repeat
  $i := i + 1$; $h_i := h$
  $(\mathbf{t}, c, p) :=$ BESTRULE$(h, \mathbf{z})$
  insert rule $(\mathbf{t}, c)$ into $h$ just before rule $p$
until $(\hat{\mathbf{r}}(h; \mathbf{z}) \geq \hat{\mathbf{r}}(h_i; \mathbf{z})$ or siz$(h) > S)$
```

Figure 7.1: Procedure GREEDY

Definition 7.1 Suppose we are given a rule list $h = \lambda_r \ldots \lambda_1 \lambda_0$ and sequence of examples $\mathbf{z}$. Let λ be a rule and $0 \leq p \leq r$. Then the *gain* of (λ, p), written $\text{gain}(\lambda, p)$, is $\hat{\mathbf{r}}(h; \mathbf{z}) - \hat{\mathbf{r}}(h'; \mathbf{z})$, where h' is obtained from h by inserting λ immediately before rule p. (Recall from Def. 3.4 that $\hat{\mathbf{r}}(h; \mathbf{z})$ is the empirical risk of h on $\mathbf{z}$.) The *gain-cost ratio* of (λ, p), written $\text{gcr}(\lambda, p)$, is $\text{gain}(\lambda, p)/\text{siz}(\lambda)$. In addition, $\text{gain}(\lambda) \stackrel{\text{def}}{=} \max_p \text{gain}(\lambda, p)$ and $\text{gcr}(\lambda) \stackrel{\text{def}}{=} \max_p \text{gcr}(\lambda, p)$.

GREEDY makes use of a procedure BESTRULE$(h, \mathbf{z})$, which returns a pair (λ, p) with high (hopefully maximum) gain-cost ratio. BESTRULE is discussed in Section 7.2. GREEDY takes as input a sequence of training examples $\mathbf{z}$ and a size bound S. It then outputs a series of rule lists $h_0, h_1, \ldots$, stopping when it can no longer decrease the empirical risk, or when the size bound is exceeded.

7.1.2 Trading Off Empirical Risk and Hypothesis Size

The literature contains various methods for the second part of the BBG approach (trading off empirical risk and hypothesis size). These include cross-validation [7], hold-out sets [12], the minimum description-length principle [36], structural risk minimization [42], and heuristic error estimates [35]. BBG6 uses 10-fold cross-validation to choose the best size bound for algorithm GREEDY:

1. Partition the training sample $\mathbf{z}$ into ten (nearly) equal-sized parts $\mathbf{z}_1, \ldots, \mathbf{z}_{10}$. Let $\tilde{\mathbf{z}}_i$ be $\mathbf{z}$ with the subsequence $\mathbf{z}_i$ removed.

2. Run GREEDY ten times, each time using nine parts ($\tilde{\mathbf{z}}_i$) for training and the remaining part ($\mathbf{z}_i$) to evaluate the rule lists produced. In these runs use $S = |\tilde{\mathbf{z}}_i|$.

3. Let $h[i, s]$ be the largest hypothesis output by GREEDY when trained on $\tilde{\mathbf{z}}_i$, of size s or less. Since successive rule lists have increasing size and decreasing empirical risk, $h[i, s]$ is also the minimum-risk hypothesis output by GREEDY, of size s or less. Choose S_0 so as to minimize $\frac{1}{10} \sum_{i=1}^{10} \hat{\mathbf{r}}(h[i, S_0]; \mathbf{z}_i))$, the average empirical risk on the test part.

4. Now run GREEDY($\mathbf{z}, S_0$) and choose the last (lowest empirical risk) rule list output by GREEDY.

The use of cross-validation is a heuristic approximation to having a separate hold-out set on which to evaluate different size bounds. It has the practical advantage that *all* the examples available are used for training, and *all* the examples available are used for testing, but the training and test samples used in any one particular run are separate and independent. In this light, the above learning strategy may be seen as a heuristic approximation of the method analyzed in Chapter 3, based on the use of a loose ERM algorithm. (Of course, we can't really call G a loose ERM algorithm, because we have proven no approximation bounds.)

An early version of BBG set the size bound S of GREEDY to the number of training examples, and used a heuristic error estimate, based on the empirical risk and hypothesis size, to choose one of the h_i [40]. I abandoned this approach in favor of cross-validation because I found the latter to be more robust. Of course, there is a disadvantage to using 10-fold cross-validation: it increases the total run-time of the learning algorithm about 10-fold. But in any practical application the time and expense of collecting the training data is likely to exceed even that of running a learning algorithm for several hours. Thus the increased running time of cross-validation appears to be an acceptable price to pay for its advantages.

7.1.3 Instances and Primitive Tests

BBG6 handles nominal attributes and discrete linear attributes, but not continuous attributes. For the purpose of demonstrating the promise of the BBG approach of greedy rule insertion using the criterion of maximizing the gain-cost ratio, this is not a serious deficiency; however, a practical system should have some means of dealing with continuous attributes. The simplest approach is to do what other classification algorithms restricted to discrete attributes have done, and apply a preprocessor to discretize continuous attributes. Some methods of doing this are described by Kerber [26] and Catlett [8]. There is, however, no fundamental reason why one could not apply BBG's greedy rule-insertion strategy in the presence of continuous attributes.

The primitive tests are as described in Section 6.1, with one exception. It is common in many machine-learning tasks to have unknown attribute values. BBG6 treats "?" (don't know) as just another possible value for every attribute. For linear attributes the value "?" is considered incomparable to all the others. Thus the primitive tests $x_i \leq v$ and $x_i > v$ are false if $x_i = ?$, and we add additional primitive tests $x_i = ?$ and $x_i \neq ?$. If there are V possible values for discrete linear attribute i, there are a total of $2(V-1)+2 = 2V$ primitive tests for attribute i — there is no point in including the tests $x_i \leq V$, which is always true, or $x_i > V$, which is always false.

In the search for a rule of maximum gain-cost ratio, there are some combinations of primitive tests that can be immediately rejected.

Theorem 7.1 The gain-cost ratio of a rule is 0 if its precondition contains two mutually exclusive primitive tests P_1 and P_2. In particular, this holds if P_1 and P_2 have any of the following forms:

1. $x_i = v_1$ and $x_i = v_2$, where $v_1 \neq v_2$.
2. $x_i \leq v_1$ and $x_i > v_2$, where $v_1 \leq v_2$.
3. $x_i = ?$ and P, where P has the form $x_i \neq ?$, $x_i \leq v$, or $x_i > v$.

Proof. Obvious. □

Definition 7.2 We say that a predicate P_1 on X is *stronger* than P_2, and P_2 is *weaker* than P_1, if $P_1(x) \Rightarrow P_2(x)$ for all $x \in X$.

Theorem 7.2 The gain-cost ratio of a rule λ having positive gain may be increased by removing from the precondition any primitive test P_2 for which a stronger test $P_1 \neq P_2$ is also in the precondition. In particular, this holds if P_1 and P_2 have any of the following forms:

1. $x_i \leq v_1$ and $x_i \leq v_2$, where $v_1 < v_2$.
2. $x_i > v_1$ and $x_i > v_2$, where $v_1 > v_2$.

Proof. Removing the test P_2 produces an equivalent precondition, hence the gain is unchanged. But removing P_2 decreases the rule size by 1. If gain(λ) > 0 then decreasing the denominator of the gain-cost ratio while leaving the numerator unchanged increases the gain-cost ratio. □

Definition 7.3 A pair of primitive tests is *incompatible* if it has of any of the five forms given in Theorems 7.1 and 7.2. The pair is *compatible* if it is not incompatible. A rule $(\mathbf{t}, c)$ or conjunction $\mathbf{t}$ is *proper* if $\mathbf{t}$ contains no two incompatible primitive tests.

BESTRULE need only consider proper rules, since by Theorems 7.1 and 7.2, if a positive gain is achievable and (λ, p) maximizes the gain-cost ratio, then λ is proper.

BBG6 preprocesses the training sample, turning each instance into a bit vector with a position for every primitive test indicating whether or not the test evaluates true. The precondition of a rule can likewise be represented as a binary vector, with 1 indicating the presence of a primitive test in the conjunction and 0 its absence. Thus in the remainder of this chapter we will take the precondition of a rule to be an element of $\{0, 1\}^N$, where N is the total number of primitive tests.

7.2 Implementation of BESTRULE

The difficult part, of course, is implementing BESTRULE. BBG6 uses a branch-and-bound search algorithm with memory and time limits. Because of these limits there is no guarantee of optimality, but optimal or near-optimal results are often obtained (as determined by comparing the gain-cost ratio of the best solution found with an upper bound on the gain-cost ratio for unexplored portions of the search tree).

7.2.1 Parameters

Throughout Section 7.2 we use the following:

- l is the loss function. This may be implemented as a matrix of the values $l(y', y)$, for $y, y' \in C$.
- h is the current hypothesis, passed as an argument to BESTRULE; it has r non-default rules.
- $\mathbf{z} = (\mathbf{x}_1, y_1) \ldots (\mathbf{x}_m, y_m)$ is the sequence of training examples, passed as an argument to BESTRULE.
- N is the total number of primitive tests.

Actually, h itself is not passed to BESTRULE. All BESTRULE needs to know about h is $h(\mathbf{x}_i)$ and which rule of h captures $\mathbf{x}_i$, for each $1 \leq i \leq m$. In place of h, BESTRULE is passed the following:

- The number r of non-default rules in h.
- An array $\mathrm{cap}[0, \ldots, r+1]$ such that examples $\mathrm{cap}[p]$ through $\mathrm{cap}[p+1]-1$ are the examples captured by rule p, for all $0 \leq p \leq r$. Note that $\mathrm{cap}[0] = 0$ and $\mathrm{cap}[r+1] = m$.
- An array $\mathrm{hclass}[1, \ldots, m]$ such that $\mathrm{hclass}[i] = h(\mathbf{x}_i)$ for all $1 \leq i \leq m$.

Note that the examples in $\mathbf{z}$ are ordered according to the rule that captures them; i.e., the examples captured by rule 0 come first, followed by the examples captured by rule 1, etc. The last thing BESTRULE does before returning its answer (λ, p) is to update $\mathbf{z}$, cap and hclass to reflect the insertion of λ just before rule p in h, in preparation for the next call of BESTRULE.

7.2.2 Search Tree Structure

Branch-and-bound algorithms explore a search tree, the nodes of which correspond to partial solutions. Equivalently, the nodes can be thought of as sets of solutions (all completions of the partial solution). Leaf nodes are complete (single) solutions. The children of a node comprise a partition of the set of solutions corresponding to the node. The goal is to find a solution λ maximizing $\phi(\lambda)$, for some objective function ϕ. When a node ν is reached one computes an upper bound u on $\phi(\lambda)$ for λ an arbitrary completion of ν; if $u \leq \phi(\lambda^\star)$, where $\lambda^\star$ is the best solution found so far, there is no need to explore the subtree rooted at ν.

Definition 7.4 The search tree for BESTRULE has the following form:

- A non-root node is a tuple $\nu = (\mathbf{t}, c)$, where $\mathbf{t} \in \{*, 0, 1\}^N$, $c \in C$, and tests i and j are compatible whenever $t_i = 1$ and $t_j \in \{*, 1\}$. ν represents the set of proper rules that can be obtained by replacing each $*$ entry of $\mathbf{t}$ with 0 or 1.

- If $\mathbf{t}$ has no $*$ entries then ν is a leaf node.
- If $\mathbf{t}$ has some $*$ entry then ν has two children, $\text{child}(\nu, i, 0)$ and $\text{child}(\nu, i, 1)$, for some i s.t. $t_i = *$. We define $\text{child}(\nu, i, 0)$ to be the node obtained by setting t_i to 0, and we define $\text{child}(\nu, i, 1)$ to be the node obtained by setting t_i to 1 and t_j to 0 for all j incompatible with i.
- The children of the root node are the $|C|$ nodes of form $(\mathbf{t}, c)$, where $c \in C$ and $t_i = *$ for all i.
- The function we seek to maximize for a leaf λ is $\text{gcr}(\lambda)$.

Leaf nodes are rules. The insertion position is not part of the node information because the node evaluation procedure incrementally evaluates every possible position for inserting the rule in a single pass through the data. Some additional useful definitions follow.

Definition 7.5 We say that ν is *expanded on test* i if the children of ν are $\text{child}(\nu, i, 0)$ and $\text{child}(\nu, i, 1)$.

Definition 7.6 A *completion* of node $\nu = (\mathbf{t}, c)$ is any proper rule obtained by replacing each $*$ entry of $\mathbf{t}$ with 0 or 1. The *trivial completion* of ν, written ν°, is obtained by setting each $*$ entry to 0. A *unit completion* is obtained by replacing one $*$ entry with 1 and the remainder with 0.

The leaf descendants of a node ν are all the completions of ν. This holds regardless of how the test on which to expand a node is chosen. Note that the precondition of ν° is weaker than that of any other completion of ν.

Definition 7.7 Let $\mathbf{t} \in \{0, 1\}^N$. We write $\mathbf{t} \wedge [i]$ for the vector obtained from $\mathbf{t}$ by setting t_i to 1, or equivalently, the conjunction obtained from $\mathbf{t}$ by adding primitive test i. We define a *restriction* of $\mathbf{t}$ to be any proper $\mathbf{t}'$ obtained from $\mathbf{t}$ by changing any number of 0 entries to 1, or equivalently, by adding any number of additional primitive tests to the conjunction. A restriction of the rule $(\mathbf{t}, c)$ is any rule $(\mathbf{t}', c)$, where $\mathbf{t}'$ is a restriction of $\mathbf{t}$.

7.2.3 Good and Bad Examples

Let us consider how the insertion of a rule into h affects the empirical risk on $\mathbf{z}$. Since the empirical risk is the average loss on the examples, we look at how inserting a new rule affects the loss for a single example.

Definition 7.8 Let $z = (\mathbf{x}, y)$. Then $\delta(z, c) \overset{\text{def}}{=} l(c, y) - l(h(\mathbf{x}), y)$, and $\delta(z, \lambda, p) \overset{\text{def}}{=} \delta(z, h'(\mathbf{x}))$, where h' is the rule list obtained by inserting rule λ at position p in h (i.e., just before rule p of h).

Definition 7.9 $\kappa(\mathbf{x}) = j$ if $\mathbf{x}$ is captured by rule j of h.

Lemma 7.3 If $\lambda = (\mathbf{t}, c)$ and $z = (\mathbf{x}, y)$ then $\delta(z, \lambda, p) = (\mathbf{t}(\mathbf{x}) \wedge \kappa(\mathbf{x}) \leq p) \cdot \delta(z, c)$.

Proof. Write $K(\lambda, p, \mathbf{x})$ for "rule λ captures $\mathbf{x}$ when inserted at position p." If $K(\lambda, p, \mathbf{x})$ then $h'(\mathbf{x}) = c$ and so $\delta(z, \lambda, p) = \delta(z, c)$. Otherwise $h'(x) = h(x)$, so $\delta(z, \lambda, p) = 0$. Thus

$$\delta(z, \lambda, p) = (K(\lambda, p, \mathbf{x})) \cdot \delta(z, c).$$

But $K(\lambda, p, \mathbf{x})$ holds iff $\mathbf{t}(\mathbf{x})$ holds and $\mathbf{x}$ is not captured by any of the rules $p+1, \ldots, r$; i.e., $K(\lambda, p, \mathbf{x}) \Leftrightarrow \mathbf{t}(\mathbf{x}) \wedge \kappa(\mathbf{x}) \leq p$. $\square$

It is useful to split the gain into two parts: the contributions of examples whose loss decreases ("good examples"), and the contributions of the examples whose loss increases ("bad examples").

Definition 7.10 $G(\lambda, p) \overset{\text{def}}{=} \sum_{i=1}^{m} \max\{0, -\delta(z_i, \lambda, p)\}$ and
$B(\lambda, p) \overset{\text{def}}{=} \sum_{i=1}^{m} \max\{0, \delta(z_i, \lambda, p)\}$.

If we use the misclassification loss, then $G(\lambda, p)$ is just the number of misclassified examples which become correctly classified after inserting rule λ at position p, and $B(\lambda, p)$ is the number of correctly classified examples which become misclassified. G and B are useful for computing the gain and bounding it.

Lemma 7.4 $\text{gain}(\lambda, p) = \frac{1}{m}(G(\lambda, p) - B(\lambda, p))$.

Proof. Using the definitions of gain (Def. 7.1), empirical risk (Def. 3.4), and δ (Def. 7.8), we obtain

$$\begin{aligned} \text{gain}(\lambda, p) = \hat{\mathbf{r}}(h; \mathbf{z}) - \hat{\mathbf{r}}(h'; \mathbf{z}) &= \tfrac{1}{m} \textstyle\sum_{i=1}^{m} l(h(\mathbf{x}_i), y_i) - l(h'(\mathbf{x}_i), y_i) \\ &= \tfrac{1}{m} \textstyle\sum_{i=1}^{m} -\delta(z_i, \lambda, p) = \tfrac{1}{m}(G(\lambda, p) - B(\lambda, p)) \end{aligned}$$

$\square$

Lemma 7.5 If λ' is a restriction of λ and $p' \leq p$, then $G(\lambda', p') \leq G(\lambda, p)$ and $B(\lambda', p') \leq B(\lambda, p)$.

Proof. Follows directly from Lemma 7.3 and the definitions of G and B. $\square$

Corollary 7.6 $\frac{1}{m} G(\nu^\circ, r) \geq \text{gain}(\lambda, p)$ for any completion λ of ν.

Proof. Any completion λ of ν is a restriction of ν°. Since $p \leq r$, we then have by Lemma 7.5 that $G(\nu^\circ, r) \geq G(\lambda, p)$. The result then follows from Lemma 7.4 and the fact that $B(\lambda, p) \geq 0$. $\square$

We now present a procedure called goodAndBad, which is used in the evaluation of a node ν. This procedure does the following:

```
**goodAndBad**$(\nu, da, ga, ba)$:
let $(\mathbf{v}, c) = \nu$ and $(\mathbf{t}, c) = \nu^\circ$
$g := 0$; $b := 0$
for ($j := 1$ to $N$)
  $ga[j] := 0$; $ba[j] := 0$
$J := N + 1$
for ($p := 0$ to $r$)
  for ($i :=$ cap$[p]$ to cap$[p+1] - 1$) if ($\mathbf{t}(\mathbf{x}_i)$)
    for ($j := 1$ to $N$) if ($P_j(\mathbf{x}_i)$)
      $ga[j] := ga[j] + \max\{0, -\delta(z_i, c)\}$
      $ba[j] := ba[j] + \max\{0, \delta(z_i, c)\}$
    $g := g + \max\{0, -\delta(z_i, c)\}$
    $b := b + \max\{0, \delta(z_i, c)\}$
  $\rho := (g - b)/\mathrm{siz}(\mathbf{t}, c)$
  if ($\rho > \rho^\star$)
    $\rho^\star := \rho$; $p^\star := p$; $J := 0$
  for ($j := 1$ to $N$) if ($v_j = *$)
    $da[j] := \max\{da[j], ga[j] - ba[j]\}$
    $\rho := (ga[j] - ba[j])/(1 + \mathrm{siz}(\mathbf{t}, c))$
    if ($\rho > \rho^\star$)
      $\rho^\star := \rho$; $p^\star := p$; $J := j$
if ($J = 0$) $\lambda^\star := (\mathbf{t}, c)$
if ($1 \le J \le N$) $\lambda^\star := (\mathbf{t} \wedge [J], c)$
return $(g, b)$
```

Figure 7.2: Procedure goodAndBad

- It computes $G(\mathbf{t} \wedge [i], c, r)$ and $B(\mathbf{t} \wedge [i], c, r)$ for all i, where $\nu^\circ = (\mathbf{t}, c)$, storing the results in arrays ga and ba.
- It computes $m\mathrm{gcr}(\nu^\circ, p)$ and $m\mathrm{gcr}(\lambda, p)$ for all unit completions λ of ν and $0 \le p \le r$, updating as necessary $\lambda^\star$ and $p^\star$ (the current best rule and insertion position), and $\rho^\star = m\mathrm{gcr}(\lambda^\star, p^\star)$.
- It computes $m\mathrm{gain}(\lambda)$ for every unit completion λ of ν, storing the results in the array da.
- It computes and returns the values $G(\nu^\circ, r)$ and $B(\nu^\circ, r)$.

We will see later how these computed values are used in the evaluation of a node.

The procedure goodAndBad does all of the above in the same asymptotic time as needed to compute $\mathrm{gcr}(\nu^\circ, r)$, i.e., $\Theta(mN)$ time. It makes use of the array hclass to compute $\delta(z_i, c)$ in constant time, using $\delta(z_i, c) = l(c, y_i) - l(\mathrm{hclass}[i], y_i)$. It uses the array cap and the ordering of examples in $\mathbf{z}$ to compute $\kappa(\mathbf{x}_i)$, since $\kappa(\mathbf{x}_i) = p$ for cap$[p] \leq i <$ cap$[p+1]$. Figure 7.2 gives the algorithm. P_j stands for primitive test j. Note that because of the preprocessing of examples, mentioned in Section 7.1.3, the test $P_j(\mathbf{x})$ can be carried out in $O(1)$ time simply by checking bit j of (the preprocessed) $\mathbf{x}$.

7.2.4 Dominance

Besides the use of upper bounds, branch-and-bound algorithms often make use of *dominance relations* to identify subtrees that are known not to contain the optimal solution, and thus can be pruned.

Definition 7.11 A node ν *θ-dominates* node ν' of the search tree if for every completion of ν' with gain-cost ratio greater than θ, there exists a completion of ν that has a higher gain-cost ratio.

Suppose the best rule found so far has a gain-cost ratio of θ, and ν θ-dominates ν'. Then every leaf of the subtree rooted at ν' is known to be either nonoptimal or no better than the best rule found so far. Since an optimal rule does exist (there are only a finite number of rules to consider), there is no need to explore the subtree rooted at ν'.

A trivial case of θ-dominance occurs when we know that no completion of a node has a gain-cost ratio greater than θ.

Theorem 7.7 Suppose that $\mathrm{gcr}(\nu^\circ) \leq \theta$ and $G(\nu^\circ, r) \leq m\theta(1 + \mathrm{siz}(\nu^\circ))$. Then any node θ-dominates ν.

Proof. We prove the theorem by showing that no completion of ν has gain-cost ratio greater than θ. Certainly ν° does not. Any nontrivial completion λ of ν has $\mathrm{siz}(\lambda) \geq 1 + \mathrm{siz}(\nu^\circ) \overset{\text{def}}{=} S$. Using this together with Corollary 7.6 we have

$$\mathrm{gcr}(\lambda, p) \leq \frac{\mathrm{gain}(\lambda, p)}{S} \leq \frac{G(\nu^\circ, r)}{mS} \leq \frac{m\theta S}{mS} = \theta.$$

□

We shall find it useful to consider the quantity $B(\mathbf{t}, c, p) - B(\mathbf{t} \wedge [i], c, p)$, i.e., the sum of the weights of the "bad" examples ruled out by adding a new primitive test i. The following lemma shows that this quantity can only decrease for restrictions of $\mathbf{t}$.

Lemma 7.8 Let $\mathbf{t}'$ be a restriction of $\mathbf{t} \in \{0,1\}^N$, and let $t'_i = 0$. Then $B(\mathbf{t}', c, p) - B(\mathbf{t}' \wedge [i], c, p) \leq B(\mathbf{t}, c, r) - B(\mathbf{t} \wedge [i], c, r)$.

Proof. Let us write P_i for primitive test i. Then

$$\begin{aligned} & B(\mathbf{t}', c, p) - B(\mathbf{t}' \wedge [i], c, p) \\ = \; & \sum_j ((\mathbf{t}'(\mathbf{x}_j)) - (\mathbf{t}'(\mathbf{x}_j) \wedge P_i(\mathbf{x}_j))) \cdot (\kappa(\mathbf{x}_j) \leq p) \cdot \max\{0, \delta(z_j, c)\} \end{aligned}$$

$$
\begin{aligned}
&= \sum_j (\mathbf{t}'(\mathbf{x}_j)) \cdot (\neg P_i(\mathbf{x}_j)) \cdot (\kappa(\mathbf{x}_j) \leq p) \cdot \max\{0, \delta(z_j, c)\} \\
&\leq \sum_j (\mathbf{t}(\mathbf{x}_j)) \cdot (\neg P_i(\mathbf{x}_j)) \cdot \max\{0, \delta(z_j, c)\} \\
&= \sum_j ((\mathbf{t}(\mathbf{x}_j)) - (\mathbf{t}(\mathbf{x}_j) \wedge P_i(\mathbf{x}_j))) \cdot \max\{0, \delta(z_j, c)\} \\
&= B(\mathbf{t}, c, r) - B(\mathbf{t} \wedge [i], c, r)
\end{aligned}
$$

□

We are now ready to state and prove our main dominance theorem.

Theorem 7.9 Let $\theta \geq 0$, $\nu^\circ = (\mathbf{t}, c)$, $t_i = 0$, and $B(\mathbf{t} \wedge [i], c, r) + m\theta \geq B(\mathbf{t}, c, r)$. Then child$(\nu, i, 0)$ θ-dominates child$(\nu, i, 1)$.

Proof. It suffices to prove the theorem for $\theta > 0$. To see this, suppose that $\theta = 0$. Since there are only a finite number of rule-position pairs (λ, p) and at most $N + 1$ different rule sizes, there are only a finite number of different gain-cost ratios. Thus there is some $\epsilon > 0$ which is smaller than any positive gain-cost ratio. Since $\epsilon > 0 = \theta$ and $B(\mathbf{t} \wedge [i], c, r) + m\theta \geq B(\mathbf{t}, c, r)$, we have $B(\mathbf{t} \wedge [i], c, r) + m\epsilon \geq B(\mathbf{t}, c, r)$. Applying the theorem with ϵ in place of θ, we find that child$(\nu, i, 0)$ ϵ-dominates child$(\nu, i, 1)$. But $\text{gcr}(\lambda) > \theta = 0 \Leftrightarrow \text{gcr}(\lambda) > \epsilon$ for every rule λ, so we also have child$(\nu, i, 0)$ θ-dominates child$(\nu, i, 1)$.

Thus we now assume that $\theta > 0$. We prove the theorem by proving that $\text{gcr}(\mathbf{t}' \wedge [i], c, p) < \text{gcr}(\mathbf{t}', c, p)$ for every $0 \leq p \leq r$ and restriction $\mathbf{t}'$ of $\mathbf{t}$ s.t. $t'_i = 0$ and $\text{gcr}(\mathbf{t}' \wedge [i], c, p) > \theta$. In the following we write $\mathbf{t}''$ for $\mathbf{t} \wedge [i]$.

From Lemma 7.8 and the statement of the theorem, we have

$$B(\mathbf{t}', c, p) - B(\mathbf{t}'', c, p) \leq B(\mathbf{t}, c, r) - B(\mathbf{t} \wedge [i], c, r) \leq m\theta < m\,\text{gcr}(\mathbf{t}'', c, p). \tag{7.1}$$

From Lemma 7.5 we have $G(\mathbf{t}'', c, p) - G(\mathbf{t}', c, p) \leq 0$; adding this difference to the left-hand side of (7.1), applying Lemma 7.4, and dividing by m, we obtain

$$\text{gain}(\mathbf{t}'', c, p) - \text{gain}(\mathbf{t}', c, p) < \text{gcr}(\mathbf{t}'', c, p). \tag{7.2}$$

Letting $S \overset{\text{def}}{=} \text{siz}(\mathbf{t}', c)$, we have

$$\text{gain}(\mathbf{t}'', c, p) - \text{gcr}(\mathbf{t}'', c, p) = ((1 + S) - 1)\,\text{gcr}(\mathbf{t}'', c, p) = S\,\text{gcr}(\mathbf{t}'', c, p). \tag{7.3}$$

Rearranging (7.2), applying (7.3), and dividing by S, we then find that $\text{gcr}(\mathbf{t}'', c, p) < \text{gcr}(\mathbf{t}', c, p)$. □

We make use of dominance as follows. Given a node $\nu = (\mathbf{v}, c)$, we step through the tests i such that $v_i = *$. If $\tilde{\nu} \overset{\text{def}}{=} \text{child}(\nu, i, 1)$ is $(\rho^\star/m)$-dominated, then (conceptually) we expand ν on test i and immediately prune the subtree rooted at $\tilde{\nu}$. Operationally, we simply set v_i to 0. This is done by the dominate procedure, shown in Figure 7.3. It uses Theorems 7.7 and 7.9, and the values ga, ba, and b computed by procedure goodAndBad. The application of Theorem 7.7 is based on the following two facts:

1. $1 + \text{siz}(\tilde{\nu}^\circ) = 2 + \text{siz}(\nu^\circ)$.
2. $\text{gcr}(\tilde{\nu}^\circ) \le \rho^\star/m$, since the preceding call of procedure goodAndBad evaluated the rule $\tilde{\nu}^\circ$.

The procedure dominate runs in $O(N)$ time.

7.2.5 Choosing a test on which to expand

Part of evaluating a node is choosing a test on which to expand it. In general, because of time and space limits, it will not be feasible to carry the branch-and-bound search to completion; so we want the search to concentrate on finding good solutions early. To this end, our goal is to expand on that test whose addition seems most likely to lead to a good solution.

If $\nu = (\mathbf{v}, c)$ is the node to be expanded and $\nu^\circ = (\mathbf{t}, c)$, one possibility is to expand on the test i (with $v_i = *$) that maximizes $G(\mathbf{t} \wedge [i], c, r)$, since this gives us the greatest upper bound on the gain-cost ratio for completions of $\text{child}(\nu, i, 1)$. This was tried in an early version of BBG, but it has the disadvantage that it is attracted to weak tests, which can lead the search astray.

Another obvious choice is to expand on the test i (with $v_i = *$) that maximizes $\text{gain}(\mathbf{t} \wedge [i], c)$. This is in fact what BBG6 does when $\text{gain}(\mathbf{t}, c, r) > 0$. But there is a problem with this choice when $\text{gain}(\mathbf{t}, c, r) < 0$: the simplest way to increase the gain is to choose a test i sufficiently strong that $\text{gain}(\mathbf{t} \wedge [i], c, 0) = 0$, i.e., *no* good or bad examples at all are captured. This problem is especially severe when $\mathbf{v}$ has few 1 entries (as is the case early in the search) and h already has a low empirical risk. In this case we can expect $\text{gain}(\mathbf{t}, c, r)$ to be very negative, since there are few examples whose classification can be improved, and many whose classification can be worsened.

If we are using the misclassification loss, one possible solution to this problem is to use an information gain criterion. Information gain is a concept from information theory [30], and has been used in algorithms for learning decision trees [7, 35], as a criterion for choosing a test on which to partition a node. If U is a binary random variable with probability p_i of being i, then the *entropy* of U is

$$H(U) \stackrel{\text{def}}{=} H(p_0, p_1) \stackrel{\text{def}}{=} \sum_i -p_i \log_2 p_i,$$

```
**dominate**($\nu, ga, ba, b$):
Let $(\mathbf{v}, c) = \nu$
for ($j$ := 1 to $N$) if ($v_j = *$)
    if ($ga[j] \le \rho^\star \cdot (2 + \text{siz}(\nu^\circ)) \;\vee\; ba[j] + \rho^\star \ge b$)
        $v_j := 0$
```

Figure 7.3: Procedure to restrict search using dominance

where we let $0 \log_2 0 = 0$. $H(U)$ is a measure of the uncertainty about U's actual value. Suppose that there is another random variable V upon which U has some dependence. In particular, U has probability $p_{i,j}$ of being j when $V = i$. Given that $V = i$, U has an entropy of $H(p_{i,0}, p_{i,1})$. If V has probability q_i of being i, then the *conditional entropy* of U, given V, is

$$H(U|V) \overset{\text{def}}{=} \sum_i q_i H(p_{i,0}, p_{i,1}) = -\sum_{i,j} q_i p_{i,j} \log_2 p_{i,j};$$

this measures the expected amount of uncertainty remaining about U after V has been observed. Then the amount of information about U we can expect to gain from observing V is

$$I(U;V) \overset{\text{def}}{=} H(U) - H(U|V).$$

Recall that there is a target distribution $\mathcal{D}$ over $X \times C$ from which examples are drawn. Let $\mathcal{D}'$ be the conditional distribution obtained from $\mathcal{D}$ by restricting it to examples which are either good or bad for ν° (recall that some examples may be neither good nor bad). Let $z = (x, y)$ be a random variable distributed as $\mathcal{D}'$. Let $U \overset{\text{def}}{=} (z \text{ is good})$, and $V_i \overset{\text{def}}{=} P_i(x)$ (where P_i is primitive teste i). If $I(U; V_i)$ is high, then P_i would seem to be a good test to add to the precondition of ν°, *assuming that*

$$\Pr[U \mid P_i(x)] \geq Pr[U \mid \neg P_i(x)]; \tag{7.4}$$

otherwise, $\neg P_i$ is the test we really want. Let an *available* test be one for which $v_i = *$. Then our strategy is to pick an available test i satisfying (7.4) that maximizes $I(U; V_i)$, which is equivalent to maximizing $-H(U|V_i)$. If there is no available test i satisfying (7.4), then none of the available tests looks promising, and we choose among them arbitrarily.

In reality, we estimate (7.4) and $-H(U|V_i)$ by calculating the corresponding empirical quantitities on the training examples. Let g and b be the number of good and bad examples for $\nu^\circ = (\mathbf{t}, c)$, let g_i and b_i be the numbers of good and bad examples for $(\mathbf{t} \wedge [i], c)$, and let $\tilde{g}_i \overset{\text{def}}{=} g - g_i$ and $\tilde{b}_i \overset{\text{def}}{=} b - b_i$. Then the empirical equivalent of (7.4) is

$$\frac{g_i}{g_i + b_i} \geq \frac{\tilde{g}_i}{\tilde{g}_i + \tilde{b}_i}. \tag{7.5}$$

The empirical equivalent of $-H(U|V_i)$ is proportional to

$$\text{score}(g, b, g_i, b_i) \overset{\text{def}}{=} g_i \log_2 \frac{g_i}{g_i + b_i} + b_i \log_2 \frac{b_i}{g_i + b_i} + \tilde{g}_i \log_2 \frac{\tilde{g}_i}{\tilde{g}_i + \tilde{b}_i} + \tilde{b}_i \log_2 \frac{\tilde{b}_i}{\tilde{g}_i + \tilde{b}_i}.$$

From the fact that $H(p, 1-p)$ has a maximum possible value of 1, it is straightforward to show that $\text{score}(g, b, g_i, b_i)$ has a minimum possible value of $-(g + b)$ given that $0 \leq g_i \leq g$ and $0 \leq b_i \leq b$. If we let $s_i \overset{\text{def}}{=} \text{score}(g, b, g_i, b_i)$ when (7.5) holds, and $s_i \overset{\text{def}}{=} -2(g + b)$ otherwise, then choosing a test i maximizing s_i will implement the preference for tests satisfying (7.5).

```
**chooseTest**$(\nu, g, b, ga, ba, da)$:
let $\nu = (\mathbf{v}, c)$
if (there is at most one $j$ for which $v_j = *$)
   return 0
$S := -\infty$
for ($j := 1$ to $N$) if ($v_j = *$)
   if ($g > b$)
      $s := da[j]$
   else if ($ga[j]/ba[j] \geq g/b$)
      $s :=$ score$(g, b, ga[j], ba[j])$
   else
      $s := -2(g + b)$
   if ($s > S$)
      $S := s$; $J := j$
return $J$
```

Figure 7.4: Procedure to choose test on which to expand node

The above discussion assumed that we were using the misclassification loss. If this is not the case, then we can think of the examples as being weighted, each example z having a weight of $|\delta(z, c)|$, with z a good example if $\delta(z, c) < 0$ and a bad example if $\delta(z, c) > 0$. We can then continue to use equation (7.5) and the function score if we redefine g, b, g_i, and b_i as follows:

- $g \overset{\text{def}}{=} G(\mathbf{t}, c, r)$, $b \overset{\text{def}}{=} B(\mathbf{t}, c, r)$, $g_i \overset{\text{def}}{=} G(\mathbf{t} \wedge [i], c, r)$, and $b_i \overset{\text{def}}{=} B(\mathbf{t} \wedge [i], c, r)$.

Figure 7.4 gives the algorithm to pick a test on which to expand a node. It uses the values g, b, ga, ba, and da computed by procedure goodAndBad (recall that $da[j] = m\,\text{gain}(\mathbf{t} \wedge [j], c)$). If there is at most one j s.t. $v_j = *$ then all completions of ν have already been considered (in procedure goodAndBad), so there is no need to expand this node, and chooseTest returns 0. The running time of chooseTest is $O(N)$.

7.2.6 Computing the Upper Bound

Once a test J on which to expand $\nu = (\mathbf{v}, c)$ is chosen, we want to compute, for each of the two child nodes, an upper bound on $\text{gcr}(\lambda)$ for any completion λ of the child node. In doing so we may omit from consideration any completion whose gain-cost ratio is known not to exceed $\rho^\star/m$ (the gain-cost ratio of the best rule found so far). To see why, let u be an upper bound on $\text{gcr}(\lambda)$ for any completion λ whose gain-cost ratio exceeds $\rho^\star/m$, and let $u' = \max\{u, \rho^\star/m\}$. Then u' is an upper bound for *all* completions λ, and $u < u'$ only if $u' = \rho^\star/m$, in which case the node can be pruned anyway.

```
**saBnd0**$(\nu, i, J, ga)$:
let $(\mathbf{v}, c) = \nu$
$\mathbf{w} := \mathbf{v}$; $w_J := 0$
$a :=$ attr$(i)$
if ($a$ is a nominal attribute) return 0
if ($w[k < x_a] \neq *$ for all $k$) return 0
$k_1 :=$ the least $k$ such that $w[k < x_a] = *$
if ($w[x_a \leq k] \neq *$ for all $k > k_1$) return 0
$k_2 :=$ the greatest $k$ such that $w[x_a \leq k] = *$
return $ga[x_a \leq k_2] - (ga[x_a \neq ?] - ga[k_1 < x_a])$
```

Figure 7.5: Procedure to compute saBnd0

Recall that the procedure goodAndBad computes the gain-cost ratio for the trivial completion of ν and every unit completion of ν, updating $\lambda^\star$, $p^\star$ and $\rho^\star$ as necessary, hence after goodAndBad finishes the gain-cost ratios of these completions are known not to exceed $\rho^\star/m$. Thus in computing the upper bounds for the children of ν, we need only consider completions with at least two additional tests beyond those in ν°. These two additional tests may be on different attributes, or on the same attribute. In handling the latter case, we make use of the functions saBnd0 and saBnd1, which we now discuss.

Definition 7.12 We write attr(i) for the attribute tested by primitive test i.

Definition 7.13 We define saBnd0(ν, i, J) to be the maximum value of $G(\lambda, r)$, where λ ranges over all completions of ν obtained by adding two additional, distinct primitive tests $i_1, i_2 \neq J$ s.t. attr$(i_1) =$ attr$(i_2) =$ attr(i). If no such λ exists then saBnd0$(\nu, i) \overset{\text{def}}{=} 0$.

Definition 7.14 If $\mathbf{v}$ is a vector or array of N elements, we write $v[k < x_a]$ for v_j or $v[j]$, where "$k < x_a$" is test j. We define $v[x_a \leq k]$ similarly.

Figure 7.5 gives the algorithm used to compute saBnd0. It take $O(N)$ time, since there are at most N tests to examine.

Theorem 7.10 If $ga[j] = G(\mathbf{t} \wedge [j], c, r)$ for all j, where $\nu^\circ = (\mathbf{t}, c)$, then the algorithm of Figure 7.5 correctly computes saBnd0(ν, i, J).

Proof. The two added tests must be compatible, since a completion of a node is, by definition, proper. If a is a nominal attribute, then no two distinct tests on a are compatible, and saBnd0$(\nu, i, J) = 0$. If a is a linear attribute and tests i_1 and i_2 are distinct and compatible tests on attribute a, then they must be of the form $k_1 < x_a$ and $x_a \leq k_2$ for some $k_1 < k_2$. Furthermore, if we are to complete ν by adding tests

```
**saBnd1**$(\nu, J, ga)$:
let $(\mathbf{v}, c) = \nu$
$a := \text{attr}(J)$
if ($a$ is a nominal attribute) return 0
if (test $J$ is $x_a = ?$ or $x_a \neq ?$) return 0
if (test $J$ is of form $k < x_a$)
    $k_1 :=$ that $k$ such that test $J$ is $k < x_a$
    if ($v[x_a \leq k] \neq *$ for all $k > k_1$) return 0
    $k_2 :=$ the greatest $k$ such that $v[x_a \leq k] = *$
else
    $k_2 :=$ that $k$ such that test $i$ is $x_a \leq k$
    if ($v[k < x_a] \neq *$ for all $k < k_2$) return 0
    $k_1 :=$ the least $k$ such that $v[k < x_a] = *$
return $ga[x_a \leq k_2] - (ga[x_a \neq ?] - ga[k_1 < x_a])$
```

Figure 7.6: Procedure to compute saBnd1

$i_1, i_2 \neq J$ to $\mathbf{t}$, then we must have $w_{i_1} = w_{i_2} = *$. If no such tests i_1 and i_2 on a exist, then saBnd0$(\nu, i, J) = 0$. Otherwise, to maximize $G(\mathbf{t} \wedge [k_1 < x_a] \wedge [x_a \leq k_2], c, r)$, k_1 must be as small as possible and k_2 must be as large as possible, subject to the restriction that $w[k_1 < x_a] = w[x_a \leq k_2] = *$ (by Lemma 7.5).

To finish the proof, it remains only to show that

$$G(\mathbf{t} \wedge [k_1 < x_a] \wedge [x_a \leq k_2], c, r) = \\ G(\mathbf{t} \wedge [x_a \leq k_2], c, r) - (G(\mathbf{t} \wedge [x_a \neq ?], c, r) - G(\mathbf{t} \wedge [k_1 < x_a], c, r)). \quad (7.6)$$

Let $e_i \overset{\text{def}}{=} (\mathbf{t}(\mathbf{x}_i)) \max\{0, -\delta(z_i, c)\}$. Then using Lemma 7.3 we have $G(\mathbf{t}', c, r) = \sum_i (P(\mathbf{x}_i)) e_i$ for all $\mathbf{t}'$ and P s.t. $\mathbf{t}'(\mathbf{x}_i) = \mathbf{t}(\mathbf{x}_i) \wedge P(\mathbf{x}_i)$. Thus (7.6) holds if

$$(k_1 < x_a \leq k_2) = (x_a \leq k_2) - (x_a \neq ?) + (k_1 < x_a)$$

when $k_1 < k_2$. If $k_1 \leq x_a \leq k_2$, then the LHS above is 1 and the RHS is $1 - 1 + 1 = 1$. If $x_a = ?$, then the LHS is 0 and the RHS is $0 - 0 + 0 = 0$. If $x_a \leq k_1$ then the LHS is 0 and the RHS is $1 - 1 + 0 = 0$. Finally, if $k_2 < x_a$ then the LHS is 0 and the RHS is $0 - 1 + 1 = 0$. Thus the equality holds, and the theorem is proven. □

Definition 7.15 We define saBnd1(ν, J) to be the maximum value of $G(\lambda, r)$, where λ ranges over all completions of ν obtained by adding one additional primitive test $i \neq J$ s.t. attr(i) = attr(J). If no such λ exists then saBnd1$(\nu, J) \overset{\text{def}}{=} 0$.

Figure 7.6 gives the algorithm used to compute saBnd1. The algorithm takes $O(N)$ time, since there are at most N tests to examine.

```
**upperBound**$(\nu, J, ga)$:
let $(\mathbf{v}, c) = \nu$
$g_1 := \max\{ga[j] : v_j = *, j \neq J\}$
$j_1 :=$ the maximizing $j$ above
$g_2 := \max\{ga[j] : v_j = *, j \neq J, \mathrm{attr}(j) \neq \mathrm{attr}(j_1)\}$
$u_0 := \max\{g_2, \mathrm{saBnd0}(\nu, j_1, ga)\}/(2 + \mathrm{siz}(\nu^\circ))$
$g_3 := (\mathrm{attr}(j_1) = \mathrm{attr}(J)\,?\, g_2 : g_1)$
$g_4 := \min\{g_3, ga[J]\}$
$u_1 := \max\{g_4, \mathrm{saBnd1}(\nu, J, ga)\}/(2 + \mathrm{siz}(\nu^\circ))$
return $(u_0, u_1)$
```

Figure 7.7: Procedure to compute upper bound

Theorem 7.11 If $ga[j] = G(\mathbf{t} \wedge [j], c, r)$ for all j, where $\nu^\circ = (\mathbf{t}, c)$, then the algorithm of Figure 7.6 correctly computes $\mathrm{saBnd1}(\nu, J)$.

Proof. Any test i added must be compatible with J. There exists no test on attribute a compatible with test J if a is a nominal attribute, or if test J is $x_a = ?$ or $x_a \neq ?$, hence $\mathrm{saBnd1}(\nu, J) = 0$ in these cases. The remainder of the proof follows that of Theorem 7.10. □

Figure 7.7 gives the algorithm for computing the upper bounds for $\mathrm{child}(\nu, J, 1)$ and $\mathrm{child}(\nu, J, 0)$. We now prove its correctness.

Theorem 7.12 Let $(\mathbf{v}, c) = \nu$ and $\nu^\circ = (\mathbf{t}, c)$. Suppose that $v_J = *$, $v_j = *$ for some $j \neq J$, and $ga[j] = G(\mathbf{t} \wedge [j], c, r)$ for all j s.t. $v_j = *$. Then the value u_0 computed in Figure 7.7 is an upper bound on $m\,\mathrm{gcr}(\lambda)$ for any completion λ of $\mathrm{child}(\nu, J, 0)$ containing at least two additional primitive tests.

Proof. We first bound $G(\mathbf{t}', c, r)$ for any restriction $\mathbf{t}'$ of $\mathbf{t}$ containing exactly two additional tests $i_1, i_2 \neq J$ (i.e., $\mathbf{t}' = \mathbf{t} \wedge [i_1] \wedge [i_2]$). Let $a \overset{\mathrm{def}}{=} \mathrm{attr}(j_1)$. There are two cases to consider:

1. $\mathrm{attr}(i_1) \neq a$ or $\mathrm{attr}(i_2) \neq a$. Then $g_2 \geq ga[i_1]$ or $g_2 \geq ga[i_2]$. But $ga[i_1] = G(\mathbf{t} \wedge [i_1], c, r) \geq G(\mathbf{t}', c, r)$, and $ga[i_2] = G(\mathbf{t} \wedge [i_2], c, r) \geq G(\mathbf{t}', c, r)$ (using Lemma 7.5), hence $g_2 \geq G(\mathbf{t}', c, r)$.
2. $\mathrm{attr}(i_1) = \mathrm{attr}(i_2) = a$. Then $\mathrm{saBnd0}(\nu, j_1) \geq G(\mathbf{t}', c, r)$ by definition.

Thus $M \overset{\mathrm{def}}{=} \max\{g_2, \mathrm{saBnd0}(\nu, j_1)\}$ bounds $G(\mathbf{t}', c, r)$. But from Lemma 7.5 and the fact that $G(\lambda, p) \geq m\,\mathrm{gain}(\lambda, p)$, this implies that $M \geq m\,\mathrm{gain}(\lambda)$ for any completion λ of $\mathrm{child}(\nu, J, 0)$ containing at least two additional primitive tests. Since all such rules λ have $\mathrm{siz}(\lambda) \geq 2 + \mathrm{siz}(\nu^\circ)$, we then see that $u_0 = M/(2 + \mathrm{siz}(\nu^\circ))$ is an upper bound on $m\,\mathrm{gcr}(\lambda)$. □

```
**eval**($\nu$):
$(g, b)$ := goodAndBad($\nu, da, ga, ba$)
dominate($\nu, ga, ba, b$)
$J$ := chooseTest($\nu, g, b, ga, ba, da$)
if ($J = 0$)
   return $(0, 0, 0, 0, 0)$
$(u_0, u_1)$ := upperBound($\nu, J, ga$)
$(l_0, l_1)$ := lowerBound($\nu, J, g, b, ga, ba, da$)
return $(J, u_0, l_0, u_1, l_1)$
```

Figure 7.8: Node evaluation procedure

Theorem 7.13 Let $(\mathbf{v}, c) = \nu$ and $\nu^\circ = (\mathbf{t}, c)$. Suppose that $v_J = *$, $v_j = *$ for some $j \neq J$, and $ga[j] = G(\mathbf{t} \wedge [j], c, r)$ for all j s.t. $v_j = *$. Then the value u_1 computed in Figure 7.7 is an upper bound on $m\,\mathrm{gcr}(\lambda)$ for any completion λ of child$(\nu, J, 1)$ containing at least one additional primitive test (i.e., two more than ν°).

Proof. We first bound $G(\mathbf{t}', c, r)$ for any restriction $\mathbf{t}'$ of $\mathbf{t} \wedge [J]$ containing exactly one additional test $j \neq J$ (i.e., $\mathbf{t}' = \mathbf{t} \wedge [J] \wedge [j]$). Let $a \overset{\text{def}}{=} \mathrm{attr}(J)$. There are two cases to consider:

1. attr$(j) \neq a$. If attr$(j_1) = a$ then $g_3 = g_2 \geq ga[j]$, otherwise attr$(j_1) \neq a$ so $g_3 = g_1 \geq ga[j]$; in either case, $g_3 \geq ga[j]$. But $ga[j] = G(\mathbf{t} \wedge [j], c, r) \geq G(\mathbf{t}', c, r)$ and $ga[J] = G(\mathbf{t} \wedge [J], c, r) \geq G(\mathbf{t}', c, r)$ (using Lemma 7.5), hence $g_4 = \min\{g_3, ga[J]\} \geq G(\mathbf{t}', c, r)$.
2. attr$(j) = a$. Then saBnd1$(\nu, J) \geq G(\mathbf{t}', c, r)$ by definition.

Thus $M \overset{\text{def}}{=} \max\{g_4, \mathrm{saBnd1}(\nu, J)\} \geq G(\mathbf{t}', c, r)$. But from Lemma 7.5 and the fact that $G(\lambda, p) \geq m\,\mathrm{gain}(\lambda, p)$, this implies that $M \geq m\,\mathrm{gain}(\lambda)$ for any completion λ of child$(\nu, J, 1)$ containing at least one additional test. Since all such rules λ have size at least $2 + \mathrm{siz}(\nu^\circ)$, we then see that $u_1 = M/(2 + \mathrm{siz}(\nu))$ is an upper bound on $m\,\mathrm{gcr}(\lambda)$. □

7.2.7 Node Evaluation

Figure 7.8 gives the complete procedure for evaluating a node. The procedure lowerBound is discussed in Section 7.2.9. Since goodAndBad runs in $O(mN)$ time and the remaining procedure calls run in $O(N)$ time, the entire procedure runs in $O(mN)$ time.

In this approach the eval procedure is run on a node that is about to be expanded. An alternative approach would be to run eval on a node when it is created, taking $\max\{u_0, u_1\}$ as the upper bound, and storing with the node the test J on which it

```
**updateArrs**$(\lambda, p, \mathbf{z}, \text{cap}, \text{hclass})$:
let $(\mathbf{t}, c) = \lambda$
$j := 0$; $i := 1$
for ($q := 0$ to $p$)
    $k := i$
    for ($s := \text{cap}[q]$ to $\text{cap}[q+1] - 1$)
        if ($\mathbf{t}(\mathbf{x}_s)$)
            $j := j + 1$; $\text{buf}[j] := z_s$
        else
            $z_i := z_s$; $\text{hclass}[i] := \text{hclass}[s]$; $i := i + 1$
    $\text{cap}[q] := k$
for ($q := r + 1$ downto $p + 1$)
    $\text{cap}[q + 1] := \text{cap}[q]$
$\text{cap}[p + 1] := i$
for ($s := 1$ to $j$)
    $z_i := \text{buf}[s]$; $\text{hclass}[i] := c$; $i := i + 1$
```

Figure 7.9: Procedure to update cap, hclass and $\mathbf{z}$

is to be expanded, for future use when the node is selected for expansion. This has the advantage of giving a tighter upper bound, at the cost of twice as many calls of eval (eval is called for each child of a node after it is expanded, versus a single call to eval before the node is expanded). Experiments with this alternate approach showed results slightly inferior to the approach taken here, for the same number of node evaluations.

7.2.8 Updating cap, hclass, and $\mathbf{z}$

Recall from Section 7.2.3 that the examples are grouped together according to the rule that captures them, with examples $\text{cap}[p]$ through $\text{cap}[p+1]-1$ captured by rule p, and $\text{hclass}[i] = h(z_i)$. Once a rule λ and position p at which to insert it are chosen, the last thing BESTRULE must do before returning (λ, p) is to update the arrays $\mathbf{z}$, cap and hclass to reflect the insertion of the new rule. This is done in $O(mN)$ time by the procedure of Figure 7.9.

7.2.9 Search control

We are now ready to consider the branch-and-bound search itself. As previously mentioned, the search has a time bound τ and space bound μ, meaning that the search stops after τ nodes have been evaluated, and that the search tree may have at most μ unexpanded nodes at any time (which may require pruning some nodes to make room for more promising ones). BESTRULE uses the following variables:

- $\rho^\star$, $\lambda^\star$ and $p^\star$: already described.

```
insert($u, l, \nu, F$):
if ($u > \rho^\star$)
   if ($|F| < \mu$)
      $F := F \cup \{(u, l, \nu)\}$
   else
      let $(u', l', \nu')$ be an element of $F$ minimizing $u'$
      if ($u > u'$)
         $F := F - \{(u', l', \nu')\} \cup \{(u, l, \nu)\}$
```

Figure 7.10: Conditional insertion procedure

- k, k_i and k_d: the number of nodes evaluated so far, the number evaluated by the intensification strategy, and the number evaluated by the diversification strategy (details follow).
- F: the frontier of the search tree, i.e., the set of "open" or unexpanded nodes, with some associated information.

The elements of F have the form (u, l, ν), where ν is the node, u its associated upper bound, and l is an associated lower bound (to be discussed shortly).

When a new node is created and not immediately chosen for expansion, it is either placed in F or discarded. Whether it is discarded depends on whether its upper bound is greater than $\rho^\star$, whether there is room in F, and if not, whether its upper bound is better than that of the worst element of F. Figure 7.10 gives the conditional insertion procedure.

In the absence of time and space bounds, the optimal search strategy would be to always expand the node with the highest upper bound. However, the search will often have to be cut short before every part of the search tree has been explored or shown not to contain any improvement over the current best solution. Thus it is important that the search strategy quickly find good solutions. Ideally, the best (or nearly best) solution would be found early in the search, with the rest of the search only confirming (near) optimality, thus minimizing the effects of stopping the search early.

BBG6 uses two different search strategies in parallel. The first is to repeatedly remove from F the node with the highest upper bound and explore a linear path down from the node, adding a new primitive test at every step, until reaching a descendant whose upper bound does not exceed $\rho^\star$. The greedy, downward search tends to quickly find a reasonably good solution. By starting each downward search from the node with the highest upper bound, we concentrate on those areas of the search space that have the greatest potential of containing high-quality solutions. Note that the nodes $\nu = (\mathbf{v}, c)$ with high upper bounds tend to be high in the search tree — fewer 1's in $\mathbf{v}$ means a smaller denominator in the g.c.r. bound, and fewer "good" examples ruled out, while fewer 0's means more possible additional tests to choose from. This causes somewhat of a "diversification" effect, to use a term from the literature on

```
**DIVERSE-SEARCH**:
$(u, l, \nu)$ := remove from $F$ an element maximizing $u$
while $(u > \rho^\star)$
   $(J, u_0, l_0, u_1, l_1) := \mathrm{eval}(\nu)$
   $k := k + 1$; $k_d := k_d + 1$
   if $(J \neq 0)$
      $\mathrm{insert}(u_0, l_0, \mathrm{child}(\nu, J, 0), F)$
      $(u, l, \nu) := (u_1, l_1, \mathrm{child}(\nu, J, 1))$
   else
      $u := 0$
```

Figure 7.11: One downward search of diversification strategy

```
**INTENSE-STEP**:
$(u, l, \nu)$ := remove from $F$ an element maximizing $l$
if $(u > \rho^\star)$
   $(J, u_0, l_0, u_1, l_1) := \mathrm{eval}(\nu)$
   $k := k + 1$; $k_i := k_i + 1$
   if $(J \neq 0)$
      $\mathrm{insert}(u_0, l_0, \mathrm{child}(\nu, J, 0), F)$
      $\mathrm{insert}(u_1, l_1, \mathrm{child}(\nu, J, 1), F)$
```

Figure 7.12: One step of intensification strategy

tabu search [14, 15], in which different downward searches tend to explore dissimilar regions of the search space. Figure 7.11 gives the procedure for a single downard search. The variable k_d keeps count of the number of nodes evaluated by this first strategy.

The second search strategy complements the first by providing an "intensification" effect (again borrowing a term from the tabu search literature), in which the search is concentrated in the vicinity of known good solutions. Associated with each node ν is a lower bound for ν, i.e., (m times) the gain-cost ratio for some completion of ν. This lower bound may be either (m times) the gain-cost ratio of ν° or of the best unit completion of ν (or 0 in certain cases). The second search strategy is to repeatedly pick the node with the best lower bound and expand it. The second strategy is used only when there is some node ν in F with $\mathrm{gain}(\nu^\circ) > 0$. Figure 7.12 gives the procedure for a single step of this second strategy, and the variable k_i keeps count of the number of nodes it evaluates.

We compute the lower bounds for $\nu_1 = \mathrm{child}(\nu, J, 1)$ and $\nu_0 = \mathrm{child}(\nu, J, 0)$ when node ν is evaluated, after it is determined that ν will be expanded on test J. The lower bound for ν_1 is just $m\,\mathrm{gcr}({\nu_1}^\circ)$, or 0 if $\mathrm{gain}({\nu_1}^\circ) \leq 0$. We have more information about the potential of ν_0: not only do we know $\mathrm{gcr}({\nu_0}^\circ)$, but also $\mathrm{gcr}(\lambda)$ for every unit

```
**lowerBound**$(\nu, J, g, b, ga, ba, da)$:
let $(\mathbf{v}, c) = \nu$
$l_1 := (ga[J] \leq ba[J]\,?\,0 : da[J]/(1 + \text{siz}(\nu^\circ)))$
$s := \max\{da[j] : v_j = *, j \neq J\}$
$l_0 := (g \leq b\,?\,0 : s/(1 + \text{siz}(\nu^\circ)))$
return $(l_0, l_1)$
```

Figure 7.13: Procedure to compute lower bound

```
**BESTRULE**$(r, \text{cap}, \text{hclass}, \mathbf{z})$:
$F := \emptyset$; $k := 0$; $\rho^\star := 0$
for (each class $c \in C$)
    $\nu := ((*, \ldots, *), c)$
    $(J, u_0, l_0, u_1, l_1) := \text{eval}(\nu)$
    $k := k + 1$
    if $(J \neq 0)$
        insert$(u_0, l_0, \text{child}(\nu, J, 0), F)$
        insert$(u_1, l_1, \text{child}(\nu, J, 1), F)$
$k_i := 0$; $k_d := 0$
while $(F \neq \emptyset \wedge k < \tau)$
    while $(k_i < k_d \wedge F \neq \emptyset \wedge k < \tau \wedge \text{bestlb}(F) > 0)$
        INTENSE-STEP
    if $(F \neq \emptyset \wedge k < \tau)$
        DIVERSE-SEARCH
updateArrs$(\lambda^\star, p^\star, \mathbf{z}, \text{cap}, \text{hclass})$
return $(\lambda^\star, p^\star)$
```

Figure 7.14: Procedure BESTRULE

completion λ of ν_0. Thus we set the lower bound for ν_0 to be the maximum value of $m\,\text{gcr}(\lambda)$ for λ a unit completion of ν_0, or 0 if $\text{gain}(\nu_0{}^\circ) \leq 0$.

The procedure for computing the lower bounds is given in Figure 7.13. Recall that $da[j]$ is $m\,\text{gain}(\lambda)$, where λ is the unit completion of ν obtained by adding test j. The procedure runs in $O(N)$ time.

All the parts are put together in Figure 7.14, which gives the BESTRULE procedure. The function $\text{bestlb}(F)$ finds an element (u, l, ν) of F maximizing l, and returns l.

Note that the following are all the operations performed on F:

1. Find $|F|$.

2. Find the record $(u, l, \nu) \in F$ maximizing u, minimizing u, or maximizing l.

3. Remove an extremal record (a record $(u, l, \nu) \in F$ maximizing u, minimizing u, or maximizing l).

4. Insert a record.

We implement F with the following data structure:

1. An integer s giving the number of node records in F.

2. An array A of node records such that $F = \{A[1], \ldots, A[s]\}$.

3. Three priority queues Q_+, Q_-, and Q_l, whose elements are indices into A, with Q_+ ordered by best upper bound, Q_- ordered by worst upper bound, and Q_l ordered by best lower bound. The priority queues are implemented by heap structures.

4. Arrays I_+, I_-, and I_l such that $Q_+[I_+[i]] = i$, $Q_-[I_-[i]] = i$, and $Q_l[I_l[i]] = i$, for $1 \leq i \leq s$.

We can find $|F|$ in $O(1)$ time, as it is just s. We can find the record $(u, l, \nu) \in F$ maximizing u, minimizing u, or maximizing l in $O(1)$ time also using the appropriate priority queue. Inserting a record into F requires $O(N + \log |F|) = O(N + \log \mu)$ time: $O(1)$ time to increment s, $O(N)$ time to copy the record into $A[s]$, and $O(\log |F|)$ time to insert the index s into each of the three priority queues and simultaneously update I_+, I_-, and I_l as elements in their respective queues are swapped. Removing an extremal record requires $O(\log |F|) = O(\log \mu)$ time: $O(1)$ time to find the record and update s, and $O(\log |F|)$ time to delete it from each of the three priority queues and simultaneously update I_+, I_- and I_l.

We now analyze the time complexity of BESTRULE. It will simplify our analysis to consider the time-cost for removing a record to be incurred when the record is inserted. Since every record that is removed was previously inserted, but some inserted records may never be removed, this can only overestimate the total time. Thus we will consider the insertion of a record to take $O(N + \log \mu) + O(\log \mu) = O(N + \log \mu)$ time, and the removal to take 0 time.

For each call of eval we have $O(mN)$ time for the call itself, plus $O(N + \log \mu)$ time for creating the children of the evaluated node and inserting one or both into F, for a total of $O(mN + \log \mu)$ time per call of eval. The call to updateArrs takes $O(mN)$ time, returning the answer takes $O(N)$ time, and the remaining variable initializations take $O(1)$ time, for a total of $O(mN)$ time, which is less than the time for a single call to eval. There are $|C|$ calls of eval in the initialization. If $|C| < \tau$ then there are at most a total of $\tau + N - 1$ calls of eval. The term $N - 1$ arises from the fact that we do not check for $k < \tau$ in the body of DIVERSE-SEARCH, which calls eval at most N times, thus we may have as many as $N - 1$ extra calls of eval beyond our bound τ. Putting this all together, the total time for BESTRULE is $O((|C| + \tau + N)(mN + \log \mu))$. In practice, it is generally true that $\tau \geq N + |C|$ and $\mu \leq 2^{mN}$, in which case we can simplify the time bound to $O(\tau m N)$.

7.3 Evolution of BBG

I first tested the basic BBG strategy with an algorithm called BBG0 [40, 41]. This early version of BBG differs from BBG6 in the following ways:

1. It is restricted to Boolean attributes only.
2. It uses a heuristic algorithm, based on hypothesis size and empirical error, to choose one of the hypotheses output by GREEDY, instead of cross-validation.
3. It makes no use of dominance relations nor the intensification search strategy.
4. The search tree structure is somewhat different. A node is a pair $(\mathbf{t}, c, p)$, with $c \in C$, p the position at which to insert the rule, and $\mathbf{t} \in \{*, -, 0, 1\}^n$. If $t_i = v \in \{0, 1\}$, this means that the test $x_i = v$ is included in the rule's precondition; if $t_i = -$, this means that there is no test on attribute i. A node has *three* children, obtained by choosing some i for which $t_i = *$ and setting t_i to –, 0, or 1.
5. It expands a node ν on the test i which gives the highest upper bound on the gain-cost ratio of $(\text{child}(\nu, i, 1), p)$.
6. It uses a weaker upper bound.
7. The alternate node evaluation approach mentioned in Section 7.2.7 is used: eval is run on a node when it is created, and the test J on which it is to be expanded is stored with the node for future use when the node is selected for expansion.

Next came BBG1–4. These improved on BBG0 as follows:

1. Arbitrary nominal attributes and discrete linear attributes were allowed, using the preprocessing step mentioned in Section 7.1.3. This also necessitated a change in the search-tree structure to its present form, where an interior node has only two children.
2. The use of dominance relations was introduced.
3. The upper bound computation was strengthened.
4. The node evaluation was altered. Rather than choose the test on which to expand on the basis of the best upper bound, both the information-gain criterion (by itself) and the combined strategy discussed in Section 7.2.5 were investigated. I also tried both the alternate node evaluation approach used in BBG0 (and mentioned in Section 7.2.7) and the node evaluation approach used in BBG6 (when a node is evaluated, compute the upper bound for both of its children). These two choices gave a total of four possibilities, hence BBG1–4. BBG4 used the same node evaluation strategy as BBG6.

In comparing BBG1–4 with each other and with BBG0's node evaluation strategy I removed the second part of BBG (choice of an appropriate size bound), and compared them under the assumption that the best size bound was always chosen. My experiments indicated that BBG1–4 needed far fewer node evaluations than BBG0 to achieve the same level of performance. In comparing BBG1–4 with each other I found that BBG4 had a small but significant advantage over BBG1–3.

I then added to BBG4 the use of cross-validation to choose the best size bound. In comparing BBG4 with the learning algorithms described in Section 7.4 I found a great advantage on somewhat noisy problems with binary attributes, but a much smaller advantage on problems with discrete linear attributes. With BBG5 I added the intensification search strategy running in parallel with the diversification search strategy, and this improved performance somewhat on problems with linear attributes. With BBG6 I added the notion of incompatible tests, with an associated strengthening of the upper bound computation, and this gave a marked performance improvement on problems with linear attributes.

7.4 Results

To evaluate BBG, I compared it to two well-known machine-learning algorithms, C4.5 [35] and CN2 [9, 10] (version 5.1). I used both the tree induction and rule induction variants of C4.5, referred to here as C4.5T and C4.5R respectively. There are two variants of CN2: one in which rule ordering is important, and one in which rule ordering is irrelevant; I used both, calling them CN2O and CN2U respectively. I used the default parameter settings with both C4.5 and CN2, with one exception: I increased the star size (width of the beam search) for CN2 to 30 or 50 from its default size of 5, in order to allow a more thorough search. Since CN2 and C4.5 don't allow arbitrary loss functions, in all experiments herein reported I used the misclassification loss.

7.4.1 Overview of C4.5 and CN2

C4.5T is a tree induction algorithm. It works by starting with a single-node tree, and successively splitting leaf nodes, by choosing a primitive partition on which to split the node. The choice of primitive partition is related to the information gain criterion discussed in Section 7.2.5. After the tree has been grown as far as possible, it is pruned back in order to avoid overfitting the data. C4.5T uses a pessimistic heuristic estimate of the actual error at each leaf of the tree to estimate whether the actual error will increase or decrease by pruning a node.

C4.5R is a rule induction algorithm. It starts with the tree produced by C4.5T and turns it into a set of rules, one for each leaf. Then it simplifies the rules, taking each rule and generalizing it by successively removing what appear to be unnecessary primitive tests in the precondition, until its heuristic indicates that the remaining primitive tests are all necessary. A criterion based on the Minimum Description Length Principle [36] is then used to choose a subset of the rules thus produced.

CN2O is a rule induction algorithm that works by successively adding new rules to the *tail* of the list of rules. Thus only the examples not captured by any of the existing rules need be considered in constructing a new rule to add to the tail of the list. CN2O chooses a new rule by trying to maximize a heuristic estimate of a rule's accuracy. Note that this may be an inappropriate criterion for problems where there is not a deterministic relation between instance and class (e.g., in the presence of classification noise).

CN2U is a rule induction algorithm which produces an unordered list of rules. When used to classify an instance that is covered by more than one rule, the class frequency vectors for each of the covering rules are combined to produce a new class frequency vector, and the most frequent class is output. CN2U runs a modified version of the CN2O algorithm once for each class $c \in C$, looking only for rules with output class c, and outputs the union of the resulting sets of rules.

7.4.2 Results for BBG0

We begin with some results for BBG0, the earliest version of BBG. These results are of interest because they demonstrate what even an unsophisticated implementation of the BBG approach can achieve; they also demonstrate the use of the data-set generators BRGEN0 and BTGEN.

I tested BBG0 on 1050 synthetic data sets generated using BRGEN0 and BTGEN, comparing its performance with C4.5 (at that time I had not yet obtained CN2). Each data set consisted of 500 training examples and 30000 test examples for estimating the actual error of the hypothesis output by a learning algorithm.

For BBG0 I set the startsize ς to 1 (see Def. 6.7), and I set the memory limit μ and time limit τ as follows. Let m be the number of examples (500) and n the number of Boolean attributes. For the 500 data sets summarized in Table 7.1 I set μ to $2mn$. For the 550 data sets summarized in Table 7.2 I set μ to $\max(50000, 2mn)$. In both cases I set τ to $\max(250000, 10mn)$. (For $n = 50$ and $m = 500$ we get $2mn = 50000$ and $10mn = 250000$.) Average execution times on an HP 710 workstation were around 20 minutes, but could be as much as 50 minutes. Examination of execution traces revealed that often the optimum rule-position pair was found, resulting in BESTRULE terminating long before the time bound was reached.

I generated 50 data sets for each of 10 different parameter settings of BRGEN0 (a total of 500 data sets), and compared BBG0 with C4.5 on these. I always set $m = 500$, $m' = 30000$, and $r + l = 41$, giving a size of 42 for all of the target rule lists. Beyond this, I came up with the 10 parameter settings by choosing them at random.

The results are summarized in Table 7.1. The column labeled BBG0 gives the amount by which the average error of BBG0 exceeds the noise level η, and the columns labeled c4.5 and c4.5rules give the amounts by which the average errors of the corresponding algorithms exceed that of BBG0. Each of these amounts is in percent, with a 95% confidence interval computed. The noise level η is also shown in percent. For each of the ten settings of parameter values for rgenex, BBG0 outperforms both c4.5 and c4.5rules by wide margins.

I next generated 50 data sets for each of 11 different parameter settings of BTGEN

l	r	n	k	η	BBG0	c4.5	c4.5rules
29	12	38	3	3	1.85 ± 0.84	9.47 ± 1.79	8.31 ± 1.49
32	9	38	5	2	1.38 ± 0.51	12.20 ± 2.04	9.76 ± 1.57
30	11	45	5	0	0.45 ± 0.26	11.41 ± 1.58	8.30 ± 1.62
34	7	50	4	4	3.56 ± 0.77	13.22 ± 2.42	13.21 ± 2.20
34	7	52	2	5	4.05 ± 1.01	8.17 ± 1.42	7.44 ± 1.32
30	11	54	6	0	0.58 ± 0.30	12.57 ± 1.82	9.31 ± 1.58
32	9	61	4	2	2.19 ± 0.82	12.50 ± 1.78	10.99 ± 1.73
36	5	64	3	5	6.21 ± 0.93	6.75 ± 1.43	8.17 ± 1.41
31	10	67	3	4	2.97 ± 0.80	13.73 ± 2.11	12.10 ± 2.14
28	13	79	2	2	4.36 ± 1.02	6.39 ± 1.50	5.30 ± 1.42

Table 7.1: Comparison of BBG0 and C4.5 on rule-oriented problems

s	n	k	η	BBG0	c4.5	c4.5rules
20	38	3	3	2.33 ± 0.56	8.94 ± 1.94	3.25 ± 1.22
20	38	5	2	0.69 ± 0.28	2.87 ± 1.14	0.86 ± 0.67
20	45	5	0	0.58 ± 0.32	3.88 ± 1.30	0.09 ± 0.35
20	50	4	4	1.43 ± 0.49	6.33 ± 1.72	3.13 ± 1.16
20	52	2	5	10.69 ± 1.90	14.46 ± 2.49	12.84 ± 2.63
20	54	6	0	0.76 ± 0.43	1.63 ± 0.84	−0.18 ± 0.48
20	61	4	2	1.77 ± 0.54	7.01 ± 2.02	2.14 ± 1.38
20	64	3	5	4.68 ± 0.96	12.93 ± 2.78	8.40 ± 2.58
20	67	3	4	4.66 ± 1.36	12.22 ± 2.82	6.24 ± 2.42
20	79	2	2	12.95 ± 2.63	15.79 ± 2.78	12.98 ± 3.20
20	50	2	0	3.33 ± 0.93	21.31 ± 2.38	14.71 ± 2.49

Table 7.2: Comparison of BBG0 and C4.5 on tree-oriented problems

(a total of 550 data sets), and compared BBG0 with C4.5 on these. I set $s = 20$, and, except for the eleventh parameter setting, kept the same values for n, k, and η as used for BRGEN0.

The results are summarized in Table 7.2. In each case the average performance of BBG0 exceeds that of c4.5 and either is very close to or exceeds that of c4.5rules. BBG0's performance advantage is greater when there are few classes, and is quite striking when $k = 2$ and $\eta = 0$ (the last case).

7.4.3 Results for BBG6 on UC Irvine data sets

I compared BBG6 with C4.5 and CN2 on ten data sets taken from the UC Irvine machine learning repository [31], using a star size of 50 for CN2. These data sets were chosen because they had only nominal or discrete linear attributes. On all but

data set	abbrev	nom attr	lin attr	classes	trn examp
solar flare	SF	8	2	2	710.67
chess	CH	36	0	2	2130.67
tic-tac-toe	TTT	9	0	2	638.67
breast cancer	BRC	0	9	2	466
zoo	ZOO	16	0	7	60
led display	LED	24	0	10	500
audiology	AUD	66	3	24	150.67
soybean (large)	SB	29	6	19	455.33
mol. biology	MB	57	0	2	70.67
mushroom	MSH	22	0	2	508

Table 7.3: UCI data sets used for comparison

two data sets I randomly partitioned the data into three (nearly) equal parts and ran each learning algorithm three times, using two of the parts as the training sample and the remaining part as the test sample, and averaged the results. The data sets are summarized in Table 7.3, which gives the number of nominal attributes, linear attributes, output classes, and average number of training examples used.

The exceptions were the mushroom and LED display data sets. The mushroom data set does not discriminate well between learning algorithms, since many achieve perfect classification on it due to its size (8124 examples). To make the problem more difficult I randomly divided it into 3 training samples of 508 examples each, and one test sample of 6600 examples. The LED display data set is actually an example generator. It is of interest because it has a large number of irrelevant attributes (17) and 10% noise in each relevant attribute; thus it stresses the ability of a learning algorithm to separate actual regularities in the data from statistical fluctuations. I generated 10000 examples, which I divided into 3 training samples of 500 examples each, and one test sample of 8500 examples. For these two data sets I averaged the test-sample error over runs on the 3 training samples.

For both this experiment and the experiments in Section 7.4.4, I set the time limit τ for BBG6 to 3000 node evaluations and the space limit μ to 3000 nodes. I also set the start size ς to 2, based on some previous experiments indicating a slight advantage to this choice over other possible settings.

Table 7.4 gives the results. The columns labeled with algorithm names give the average classification error, in percent. BBG6's performance is usually near that of CN2U, CN2O, and C4.5R, and superior to that of C4.5T. The exceptions are the AUD and LED data sets. BBG6 does rather worse than all the other algorithms on the AUD data set. We can compute a confidence interval for the BBG6 misclassification rate using the formula $\sigma = \sqrt{p(1-p)/n}$ for the standard error (where p is the measured proportion of misclassifications and $n = 226$ the number of test examples); we find that the 95% confidence interval is $[21.61\%, 33.25\%]$, which contains the measured misclassification rates for all the other algorithms. On the LED data set BBG6

data set	time	bbg6	cn2u	cn2o	c4.5r	c4.5t
SF	31.92	18.15	20.92	21.01	21.39	19.61
CH	42.45	0.72	1.13	0.94	0.78	0.81
TTT	0.26	1.57	0.94	0.00	0.94	14.82
BRC	7.13	4.72	5.01	5.29	4.43	5.15
ZOO	0.03	8.89	7.78	8.89	8.89	11.11
LED	47.88	28.86	38.72	36.46	35.47	33.02
AUD	29.30	27.43	22.55	25.23	23.87	23.43
SB	41.46	10.25	11.71	9.81	9.67	11.13
MB	0.04	19.84	22.67	20.63	21.75	27.35
MSH	0.20	0.30	0.11	0.29	0.35	1.35
# best		5	4	4.5	4.5	1

Table 7.4: Comparison of algorithms on UCI data sets

does rather better than all the other algorithms. The 95% confidence interval for the best of these (C4.5T), with $n = 8500$, is $[32.02\%, 34.02\%]$, and the measured misclassification rate for BBG6 is an additional 6.2σ below the bottom of this interval. Thus the measured differences on the LED data set are clearly significant, whereas it is questionable whether the differences on the AUD data set are significant or merely due to chance.

The last row of Table 7.4 gives the number of times (out of 10) each algorithm performs better than BBG6, with ties counting for 1/2. On this basis BBG6 is perhaps a bit better than C4.5R, CN20 and CN2U, and substantially better than C4.5T.

The column "time" gives the average time used by BBG6 on an HP710 workstation, in minutes. This time includes the 10-fold cross-validation used to choose a size bound, and hence can probably be reduced by about a factor of 10 by replacing cross-validation with a less time-consuming sizing heuristic.

7.4.4 Results for BBG6 on Synthetic Data

I next compared BBG6 with C4.5 and CN2 in 13 experiments using synthetic data sets. Each experiment was carried out by selecting one of the data-set generators of Chapter 6, fixing all but one of its parameters, and varying the remaining parameter. The fixed parameter settings for each experiment are given in Table 7.5, which also gives the number of the figure reporting the results of the experiment, and a mnemonic for the experiment. A * indicates the parameter that is varied. Recall that r is the number of rules, V is the number of different values an attribute can have, n is the number of attributes, $\tilde{n}$ is the number of relevant attributes, η is the noise level, and m is the number of training examples. Omitted are parameters k (number of classes) and m' (number of test examples), which were set at $k = 2$ and $m' = 3000$ for all experiments. I fixed k at 2 because this maximizes the number of rules per class, with the intention of fragmenting as much as possible the portion of the instance space

fig	name	gen	r	V	n	$\tilde{n}$	η	m
7.15	NR	BRGEN1	*	(2)	20	20	0.05	500
7.16	NM	BRGEN1	8	(2)	20	20	0.05	*
7.17	NE	BRGEN1	8	(2)	20	20	*	500
7.18	LR	LRGEN	*	5	20	20	0.05	500
7.19	LM	LRGEN	6	5	20	20	0.05	*
7.20	LE	LRGEN	6	5	20	20	*	500
7.21	NE0R	BRGEN1	*	(2)	30	10	0	500
7.22	NE0M	BRGEN1	12	(2)	30	10	0	*
7.23	LE0R	LRGEN	*	5	30	10	0	500
7.24	LE0M	LRGEN	8	5	30	10	0	*
7.25	NV5R	NRGEN	*	5	30	10	0.05	500
7.26	NV5M	NRGEN	7	5	30	10	0.05	*
7.27	NV5E	NRGEN	7	5	30	10	*	500

Table 7.5: Settings of fixed parameters

corresponding to each class, thus presumably making the problem harder to learn.

For each of 8 different values of the variable parameter I generated 30 data sets, ran the various learning algorithms on these data-sets, and computed the average misclassification error. The results are plotted in the figures indicated in Table 7.5. The key for all of these plots is given in Figure 7.15. Note that in Figures 7.17, 7.20, and 7.27 I have plotted the *difference* between the average error and the noise level η.

I used a star size of 30 for CN2 on these problems, as CN2 sometimes ran out of memory and crashed when I used a star size of 40 or 50. Note that this is still substantially larger than the star size of 15 or 20 used by the developers of CN2 in their papers [9, 10].

Unlike the UC Irvine problems, these test problems were designed to be well-represented by rule lists, to require the full representational power of rule lists, and to be difficult induction problems in the sense that the portion of the input space corresponding to each class is fragmented, and often obscured by noise. The results show generally superior performance for BBG6, often by substantial amounts. BBG6 almost always has the best error rate; the major exception is when $\eta = 0$. In this case CN20 sometimes has superior performance, but it is interesting to note that the performance of both CN20 and CN2U degrade rapidly as the the noise level η rises; this may be for the reason discussed in Section 7.4.1, i.e., that estimated rule accuracy is an inappropriate criterion for choice of a rule in the presence of noise. C4.5R has average error slightly better than BBG6 in the LE0R experiment when the number of rules is small, but as the number of rules r increases BBG6 gains the advantage.

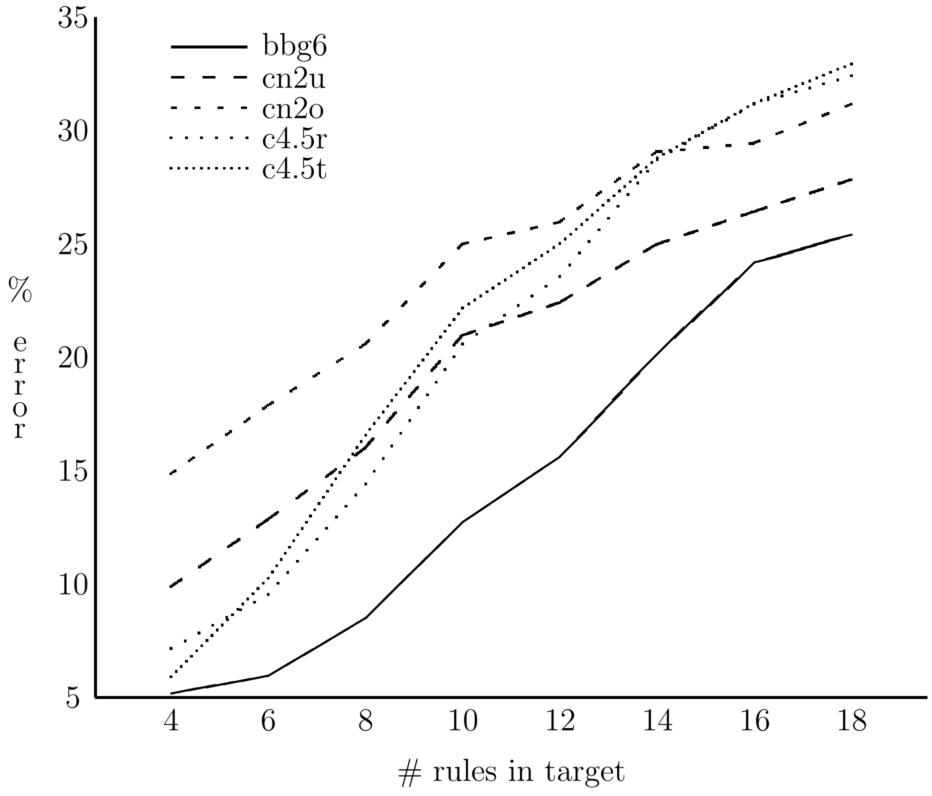

Figure 7.15: Average error as # rules varied (NR)

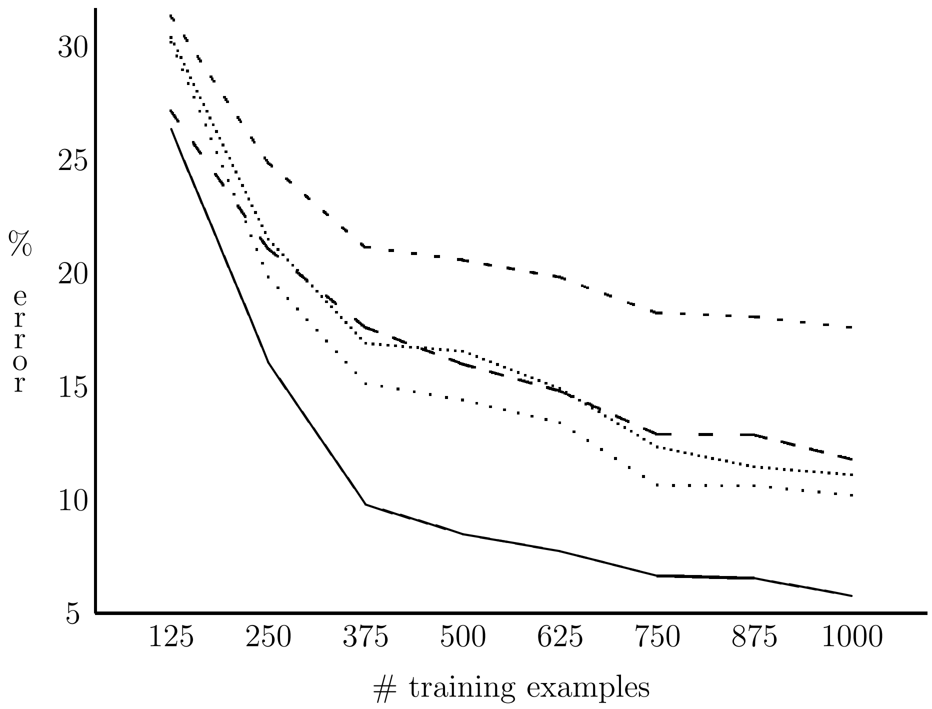

Figure 7.16: Average error as # examples varied (NM)

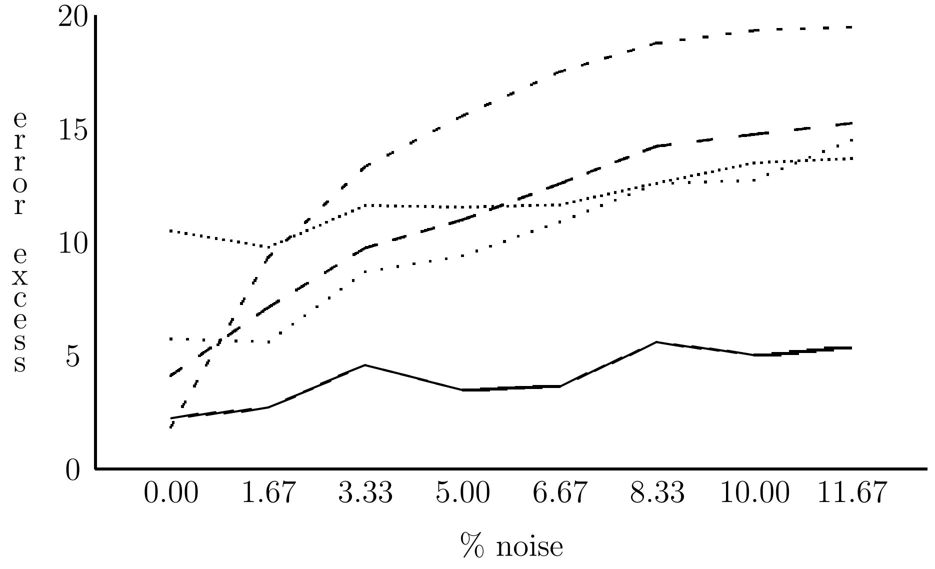

Figure 7.17: Average excess error as noise varied (NE)

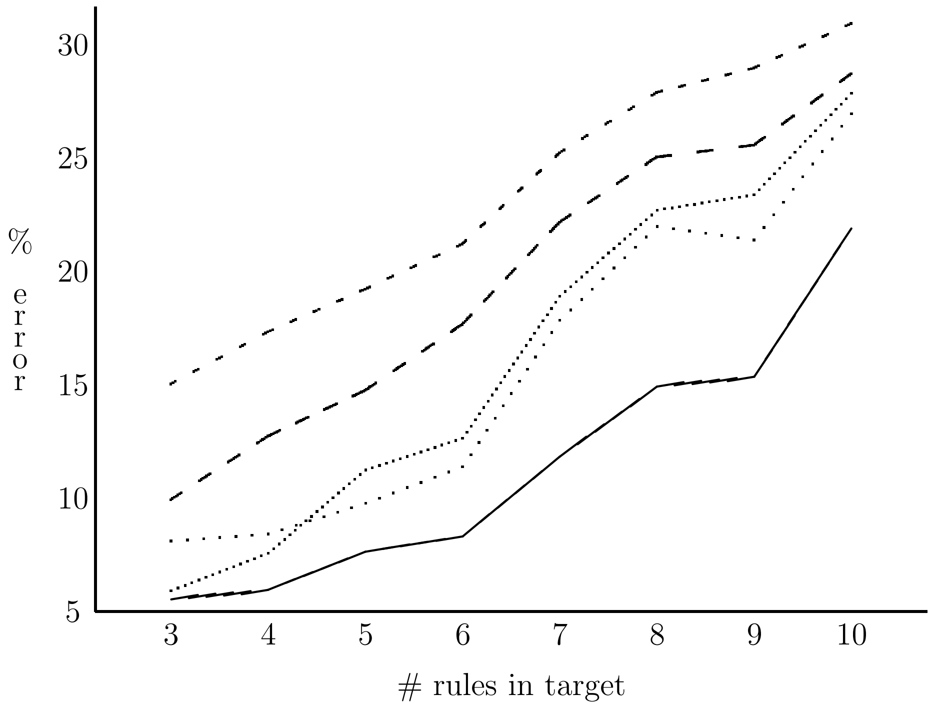

Figure 7.18: Average error as # rules varied (LR)

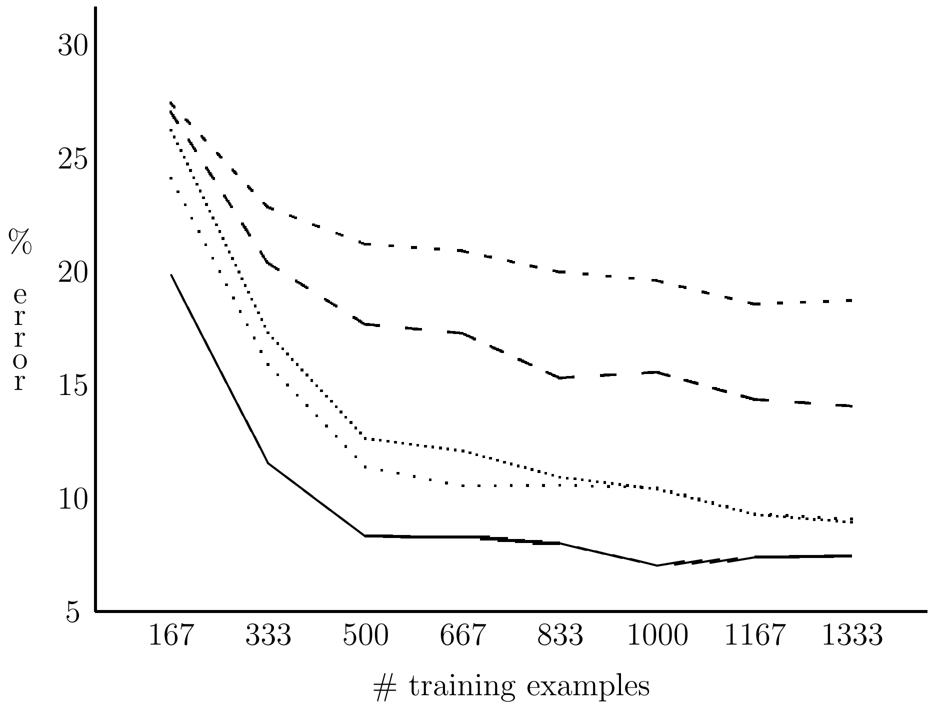

Figure 7.19: Average error as # examples varied (LM)

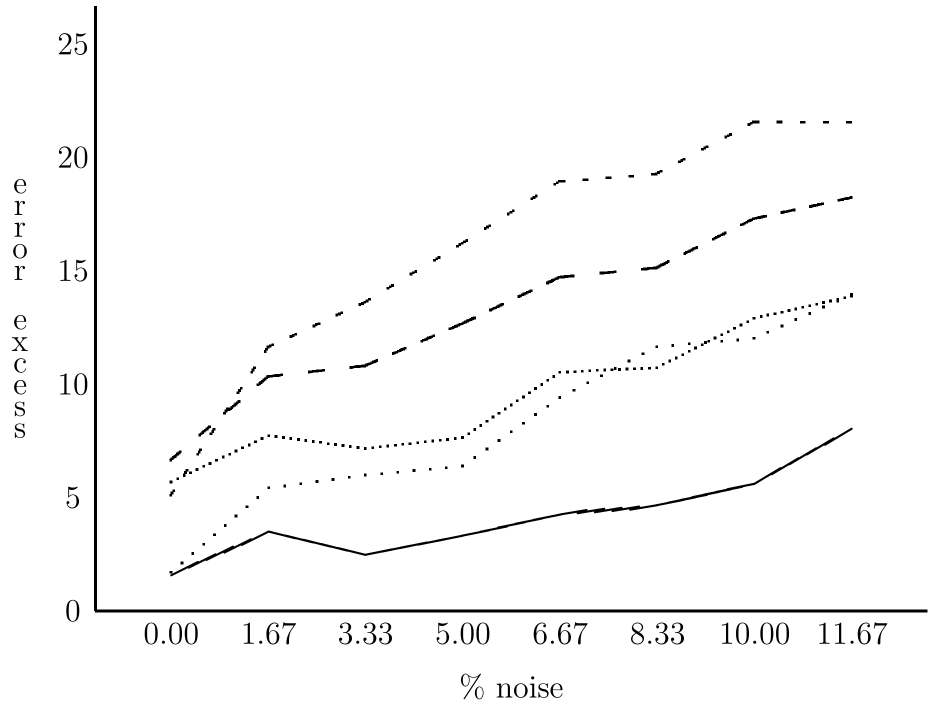

Figure 7.20: Average excess error as noise varied (LE)

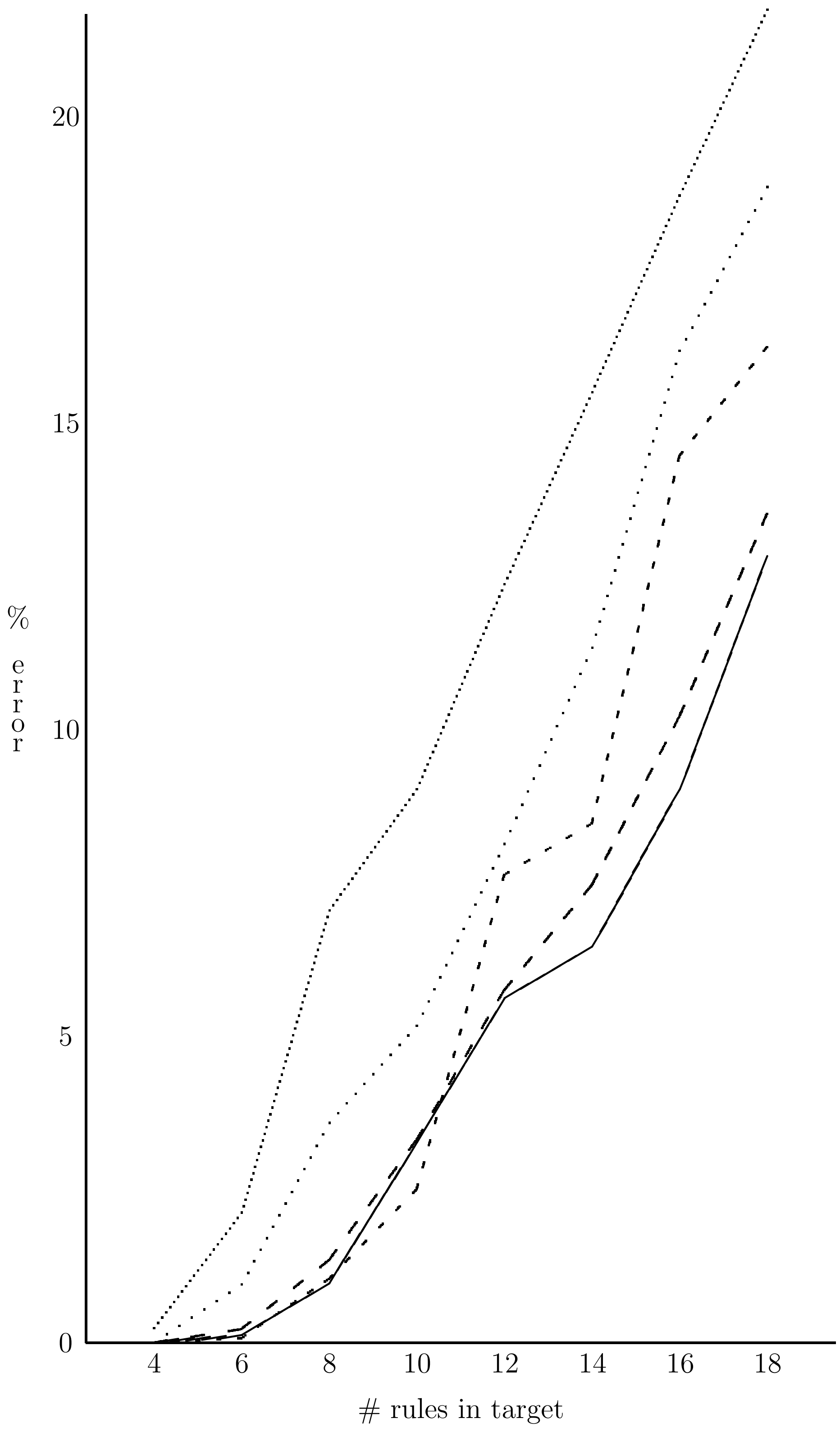

Figure 7.21: Average error as # rules varied (NE0R)

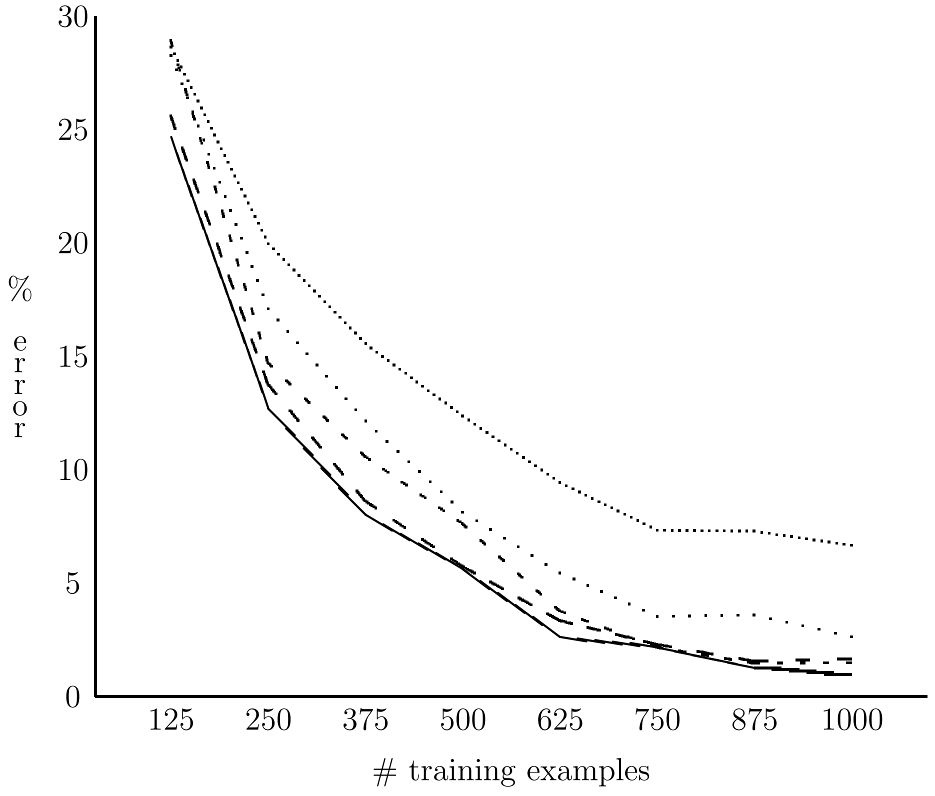

Figure 7.22: Average error as # examples varied (NE0M)

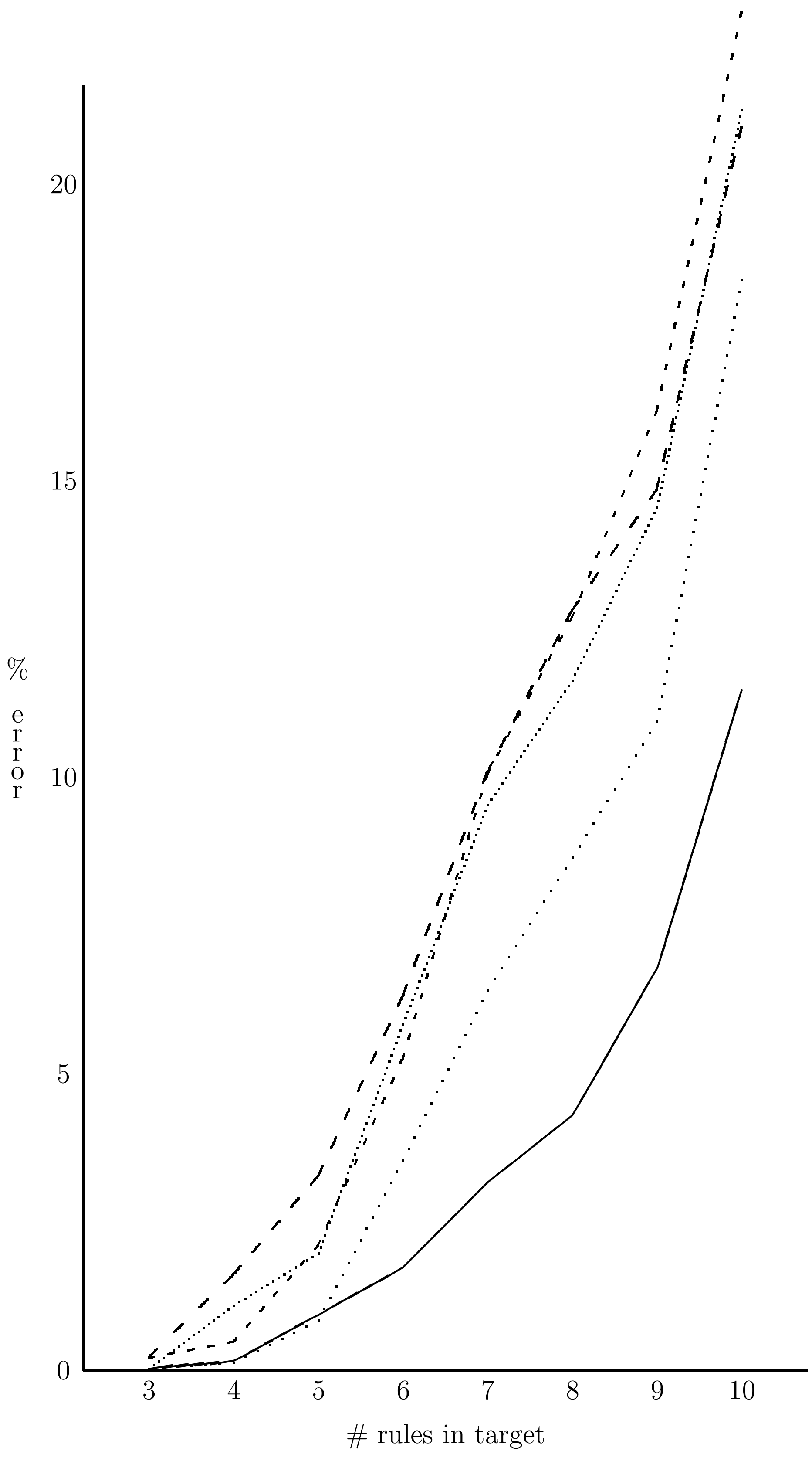

Figure 7.23: Average error as # rules varied (LE0R)

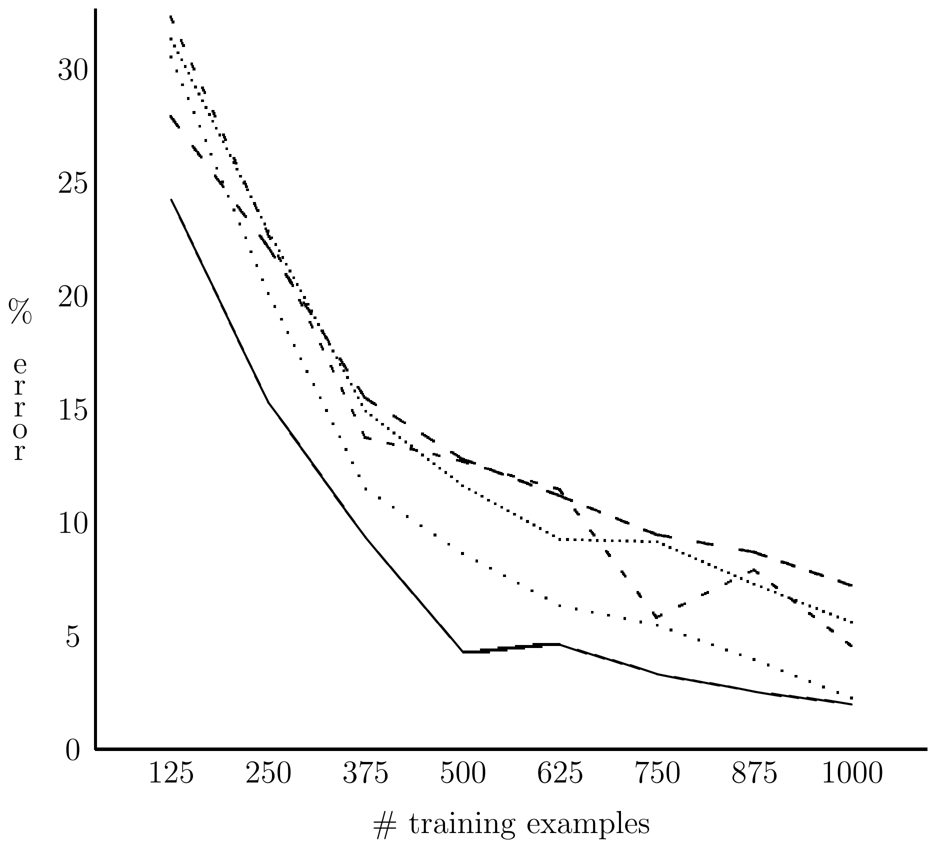

Figure 7.24: Average error as # examples varied (LE0M)

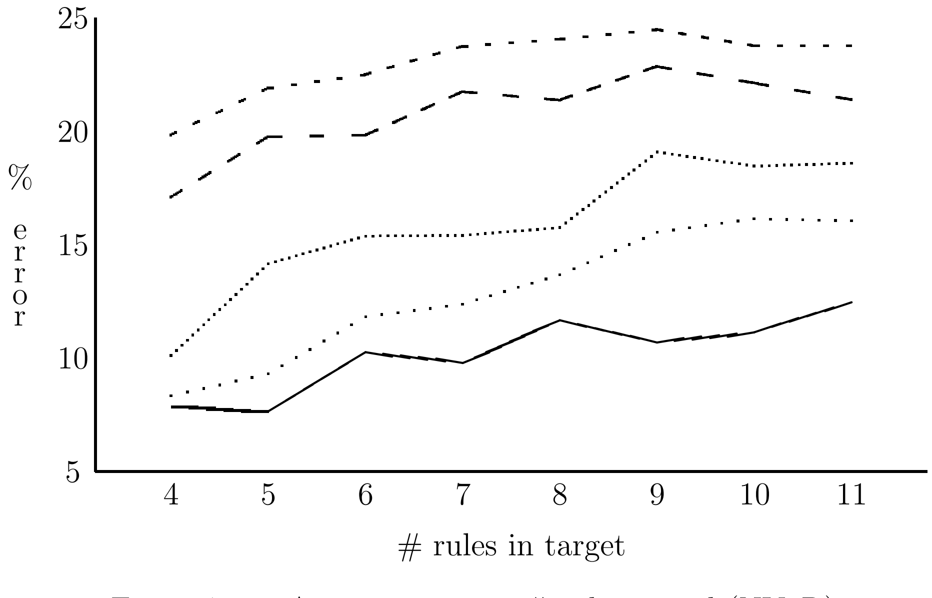

Figure 7.25: Average error as # rules varied (NV5R)

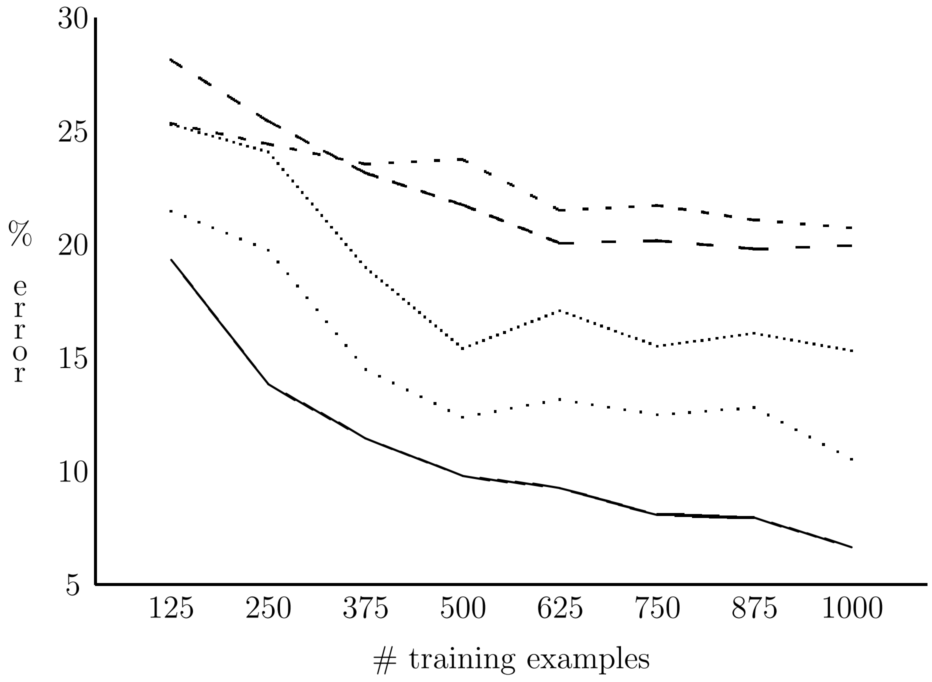

Figure 7.26: Average error as # examples varied (NV5M)

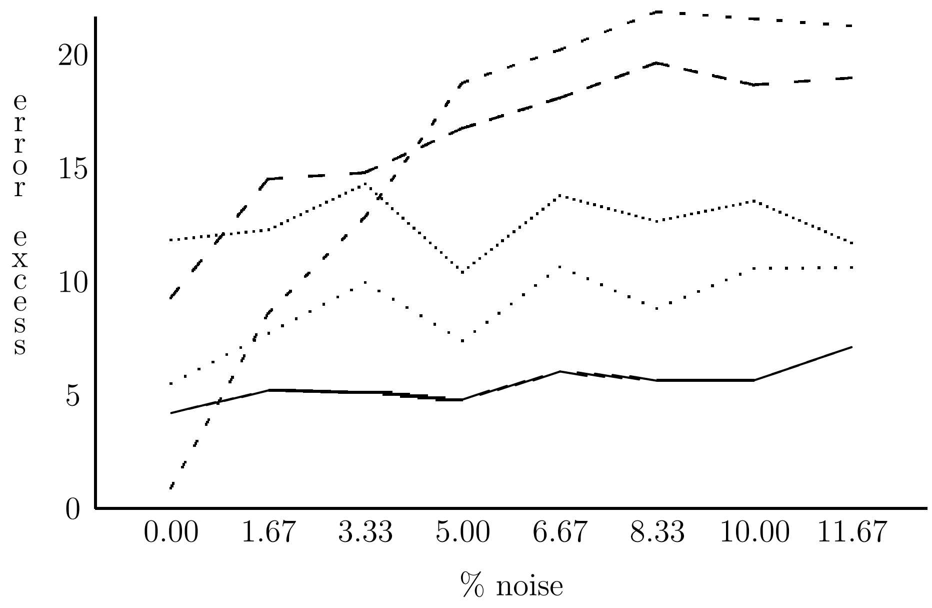

Figure 7.27: Average excess error as noise varied (NV5E)

Chapter 8

Conclusion

8.1 Summary

The following summarizes the accomplishments of this dissertation.

1. Chapters 3 and 4 give some new results relating learning and optimization:

   (a) Theorem 4.2 quantifies how minimizing the number of non-bias weights used in PAC learning a linear threshold function reduces the VC dimension of the effective hypothesis space. Since the sample complexity bound of Theorem 2.1 is linear in the VC dimension of the hypothesis space, this gives an improved sample complexity bound for learning linear threshold functions when only some of the input features are actually needed.

   (b) Chapter 3 defines loose ERM algorithms and the associated "hold-out" algorithms as an analog, for Haussler's very general learning model, of Blumer et al.'s Occam algorithms (which are for learning under the restrictive PAC model). Theorem 3.5 gives sample-complexity results under Haussler's model that are similar to the sample-complexity results of Blumer et al. for their Occam algorithms. This shows quantitatively how Occam's Razor applies in a very general learning setting.

   (c) Chapter 3 also outlines an alternative way of applying Occam's Razor, in terms of loose Occam Algorithms. Theorems 3.6 3.7 show that, under some weak assumptions, a polynomial-time loose Occam with bound $p(s)m^a$ exists for a learning problem iff a polynomial-time loose ERM algorithm with the same bound $p(s)m^a$ exists for the problem. This shows that two different strategies for applying Occam's Razor — using a loose ERM algorithm or using a loose Occam algorithm — have equivalent computational complexities.

2. Chapters 4 and 5 demonstrates the extreme computational difficulty of apparently simple machine learning problems:

   (a) Theorem 4.3 and Corollary 4.4 show that minimizing the number of non-zero non-bias weights needed to produce a linear threshold function consistent with a training sample is very hard: not only is this problem NP-complete, but it cannot even be approximated to within a constant factor unless P = NP, and it cannot be approximated to within an $o(\log m)$ factor (m the number of examples) unless NP $\subseteq$ DTIME$[n^{\log \log n}]$.

(b) Chapter 5 gives a series of results culminating in Corollaries 5.15 and 5.16. These results show that no polynomial learning algorithm can even approximate to within a constant factor the minimum classification error achievable with linear threshold functions (unless RP = NP); variations of this result are also proven for monomials and decision lists, and for the problem of using linear threshold functions to approximate the minimum error achievable with monomials. But note that it is possible to PAC learn a linear threshold function in polynomial time and sample complexity. The difference is that for PAC learning it is assumed that some hypothesis with zero error exists. Removing this one artificial assumption turns a polynomial-time problem into one that cannot even be approximated to within a constant factor in polynomial time.

3. Chapter 6 examines the expressive power of decision trees and rule lists, and quantifies the oft-reported expressive advantage of rule lists. In particular, Theorem 6.3 shows that, under reasonable definitions of the size of a rule list and the size of a decision tree, for any decision tree there exists an equivalent rule list whose size is at most log-linear in the size of the decision tree. Theorem 6.1, on the other hand, shows that there exists a class of rule lists for which the size of the smallest equivalent decision tree grows exponentially in the size of the defining rule list. Thus rule lists are more expressive than decision trees, since any function expressible with a decision tree is expressible with a rule list that is not much larger, while the reverse is not true.

4. Chapter 6 describes the random generation of learning problems whose target distribution is defined by a decision tree or rule list. Several generation algorithms are described, with parameters such as the kind of attributes, number of attribute values, number of attributes, size of hypothesis or number of rules, noise level, etc. These algorithms are designed so that every rule (resp. leaf) of the defining rule list (resp. decision tree) is likely to capture a significant fraction of the instance space. This is so that if one specifies a certain hypothesis size, the generated problem will (most likely) actually require a hypothesis of nearly that size in order to approach the minimum possible error.

   Chapter 7 demonstrates the use of the problem generators. In contrast to the use of standard machine-learning data sets, running learning algorithms on synthetic data sets and varying the parameters shows how the average error of the learning algorithms depends on the different parameters. A number of such error curves are plotted in Figures 7.4.4–7.27, comparing BBG6 with CN2O, CN2U, C4.5T, and C4.5R. Note that these curves demonstrate a weakness of CN2O and CN2U that is not shared by BBG6: although CN2O and CN2U are quite good on problems with nominal attributes and no noise, their performance rapidly degrades with the addition of even a little noise.

5. Chapter 7 describes a greedy rule-insertion approach to learning rule lists, describing in detail a particular implementation of this strategy, BBG6. The high-level BBG strategy was inspired by the results of Chapter 3 on applying the hold-out method with a loose ERM algorithm. BBG6 has the advantage over other rule induction algorithms that it handles arbitrary loss functions. Although Table 7.4 shows only a small advantage for BBG6 over other rule induction algorithms on the UC Irvine problems, Figures 7.4.4–7.27 show that BBG6 has a marked advantage on problems that were specifically designed to be difficult learning problems requiring the full representational power of rule lists. Thus BBG6 shows superior performance on the kinds of problems for which it was designed. Furthermore, Figures 7.17, 7.20, and 7.27, as well as the results on the LED data set, indicate that BBG6 is much more robust in the face of noise than CN2O or CN2U.

8.2 Further Research

Let us conclude by looking at some possible avenues for further research:

1. Continue the work of Chapter 5 by examining other hypothesis spaces known to have polynomial learning algorithms under the PAC model, to see if they also suffer from unlimited degradation when we move to PAO learning. In particular, it might be interesting to examine the hypothesis space of k-DNF formulae (Boolean functions representable by a disjunction of terms, with each term being the conjunction of at most k Boolean variables) or k-CNF formulae (the dual of k-DNF, switching the roles of conjunction and disjunction).

2. Extend the work of Chapter 6 on random generation of learning problems.

   (a) The noise parameter η could be replaced by some means of generating nonuniform noise, or even a second hypothesis that is sufficiently complex that it looks like noise. In the latter case we could reduce the misclassification error to η using a moderately-sized hypothesis, but reducing the error to zero (or even significantly less than η) would require a much larger hypothesis.

   (b) It could also be interesting to provide means of introducing various kinds of correlations between attributes, including the case of some attributes having a functional dependence on others.

   (c) Finally, it might be useful to carefully investigate some statistical properties of the problem generators. For example, one could experimentally verify, for a wide variety of parameter settings, that instances do in fact tend to be equally distributed among the rules; and one could check for any significant systematic bias favoring the capture of instances at one rule position over another. (I did some informal experiments of this sort in developing and debugging the problem generators.)

3. Improve BBG6. Sections 8.2.1 and 8.2.2 discuss two possible modifications of BBG6: handling inconsistent training samples, and handling continuous attributes.

8.2.1 Inconsistent Training Samples

An inconsistent training sample $\mathbf{z}$ is one which contains examples $z_i = (\mathbf{x}_i, y_i)$ and $z_j = (\mathbf{x}_j, y_j)$ such that $\mathbf{x}_i = \mathbf{x}_j$ but $y_i \neq y_j$. If the instance space is large and the probability distribution over instances is not too concentrated in any region of the instance space, then it is unlikely that the training sample will be inconsistent, simply because it is unlikely that the same instance will be drawn twice. However, these assumptions do not always hold, and thus it is useful to consider what happens when we have an inconsistent training sample.

The difficulty with an inconsistent training sample is that it may cause the computed upper bound on the gain-cost ratio to be overly optimistic. The upper-bound computation made use of $G(\lambda, p)$ to compute the gain that could be obtained if we could somehow restrict λ to capture only the "good" examples and none of the "bad" examples. But if the training sample contains two examples z_i and z_j with $\mathbf{x}_i = \mathbf{x}_j$ and $\delta(z_i, \lambda, p) > 0 > \delta(z_j, \lambda, p)$, then we know that we cannot capture the good example z_j without also capturing the bad example z_i; similarly, we cannot rule out the bad example z_i without also ruling out the good example z_j.

Thus in computing G and B it may be advantageous to consider maximal groups of examples with the same instance — call these "clans" — instead of just looking at individual examples. Let us suppose that the examples in $\mathbf{z}$ have been ordered such that each clan occupies a contiguous subsequence of $\mathbf{z}$. Let $\chi(j)$ be the set of indices i such that z_i is a member of clan j. Then we can redefine G and B as

1. $G(\lambda, p) \overset{\text{def}}{=} \sum_j \max\{0, -\sum_{i \in \chi(j)} \delta(z_i, \lambda, p)\}$;

2. $B(\lambda, p) \overset{\text{def}}{=} \sum_j \max\{0, \sum_{i \in \chi(j)} \delta(z_i, \lambda, p)\}$.

8.2.2 Continuous Attributes

As mentioned in Section 7.1.3, the simplest approach to handling continuous attributes is to discretize them, using methods such as those discussed by Kerber [26] and Catlett [8]. Both of their methods take the classes of examples into account when discretizing. A refinement of this approach would be to re-discretize for every call of BESTRULE, with a different discretization for every class, using the categorization of each example z_i as "good" or "bad" to drive the discretization (and perhaps taking into account the magnitude of $\delta(z_i, c)$).

It would be preferable, however, to handle continuous attributes directly. The current implementation of BESTRULE makes much use of the assumption that we have relatively few possible primitive tests per attribute, and hence would not be easily-modifiable to handle continuous attributes, where there may be upwards of m (the number of examples) thresholds v to consider for a primitive test of the form $x_j \leq v$. A more promising approach might be to use tabu search [14, 15] to implement BESTRULE. Tabu search is a general optimization strategy which extends local search in a number of ways, among them allowing the search to escape from a local minimum. Local search is a strategy of starting with some possible solution, and repeatedly evaluating a set of small changes to the current solution, choosing that change which most improves the objective function, until no further improvement can be found. These small changes are called *moves*. For maximizing the gain-cost ratio of a rule, possible moves would be the removal of a primitive test or the addition of a primitive test. I do not include changing the rule's class, since this would be a major change, unlikely to result in improvement; thus one would run a separate tabu search for each class.

We would like to evaluate all possible moves in not much more than $O(mN)$ time; I outline how to do this in $O(mN + mn \log m)$ time. I describe in detail only the evaluation of all ways of adding a new test of the form $x_j \leq v$ on a continuous attribute (tests of form $x_j > v$ are handled similarly). Evaluating all possible ways of removing a test or adding a test on a discrete attribute can be done in a manner similar to the computation of *ga*, *ba* and *da* in the procedure goodAndBad, with one added twist: we want to also consider examples for which all but one of the current primitive tests hold. This takes $O(mN)$ time.

Here is how to evaluate all possible new tests of the form $x_j \leq v$ on a continuous attribute. Let the sequence $\mathbf{z}$ be those training examples for which the precondition of the current solution holds; this could be determined when we evaluate all ways of adding a new test on a discrete attribute. Assume that we have stored with each example z_i the rule $\kappa(z_i)$ which captures z_i. Assume also that we have a data structure S which has the following operations:

1. initialize(S, r). Initialize S to have r bins, each containing the value 0; this is done in $O(r)$ time.

2. insert(S, w, R). Add w to bin R of S. This is done in $O(\log r)$ time.

3. best(S). Return a pair (R, W) such that $W = f(R)$ and $f(R) \geq f(R')$ for all $0 \leq R' \leq r$, where

   $f(R)$ is the sum of the values in bins 0 through R.

   This is done in $O(1)$ time.

We then evaluate all possible additions of a new test of form $x_j \leq v$ on a continuous attribute, as follows. For a given continous attribute j, we sort the examples in $\mathbf{z}$ on the value of attribute j. Then we call initialize(S, r) and step through the examples one by one, starting with the lowest attribute value. For each example z_i we call insert($S, \delta(z_i, c), \kappa(z_i)$), where c is the current rule's class. After each insertion we compute $(R, W) = \text{best}(S)$; if W is better than the best gain we have seen so far, then we update the best new test to be $x_j \leq v$, where v is the value of attribute j for example z_i. This whole process is repeated for every continuous attribute. The total sorting time is $O(mn \log m)$, the initialization takes $O(r)$ time, and the calls to insert and best take a total of $O(mn \log r)$ time; since $r \leq m$, the total time is $O(mn \log m)$.

S is implemented using a binary tree of depth $d = \lceil \log_2(r+1) \rceil$, with each leaf i being associated with bin i of S. Thus the leaves descended from the a-th node at level b of the tree are associated with bins $2^{d-b}a$ through $2^{d-b}(a+1) - 1$. Each node ν contains five pieces of information:

1. ν.sum. This is the sum of the values in the bins associated with ν.
2. ν.bestR. This is the bin R maximizing $f(R)$.
3. ν.bestf. This is $f(\nu.\text{bestR})$.
4. ν.left and ν.right. The left and right children of ν. Note that the bins associated with ν.left all precede the bins associated with ν.right.

The procedure initialize(S, r) constructs the tree and initializes ν.sum and ν.bestf to 0 for every node ν. ν.bestR is initialized to the lowest-numbered bin associated with ν. Since the tree has $O(r)$ nodes, this takes $O(r)$ time.

The procedure best(S) simply returns the pair $(\nu.\text{bestR}, \nu.\text{bestf})$, where ν is the root of the tree. This takes $O(1)$ time.

The procedure insert(S, w, R) does the following:

1. It works its way down from the root of the tree to the leaf associated with bin R, adding w to ν.sum for each node ν encountered.
2. Upon reaching the desired leaf node λ and updating λ.sum, it sets $\lambda.\text{bestf} := \lambda.\text{sum}$. It leaves λ.bestR unchanged from its initial value of R.
3. It then works its way back up to the root of the tree, recomputing ν.bestf and ν.bestR for every non-leaf node ν it encounters, as follows:

```
$f_0 := \nu$.left.bestf
$f_1 := \nu$.left.sum $+ \nu$.right.bestf
if ($f_0 \geq f_1$)
  $\nu$.bestf $:= f_0$; $\nu$.bestR $:= \nu$.left.bestR
else
  $\nu$.bestf $:= f_1$; $\nu$.bestR $:= \nu$.right.bestR
```

This runs in $O(d) = O(\log r)$ time. It is straightforward to verify that the above procedure maintains ν.sum, ν.bestf, and ν.bestR at their proper values.

Bibliography

[1] N. Abe & M. K. Warmuth (1990). On the computational complexity of approximating distributions by probabilistic automata. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, 52–66.

[2] T. M. Apostol (1969). *Calculus*, second edition, volume II. New York: John Wiley & Sons.

[3] E. B. Baum & D. Haussler (1989). What size net gives valid generalizations? *Neural Computation* 1, 151–160.

[4] M. Bellare, et al. (1993). Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 294–304.

[5] A. Blum (1992). Personal communication.

[6] A. Blumer, et al. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM* 36, 929–965.

[7] L. Breiman, et al. (1984). *Classification and Regression Trees.* Belmont, CA: Wadsworth.

[8] J. Catlett (1991). On changing continuous attributes into ordered discrete attributes. In *Machine Learning — EWSL-91* (*Lecture Notes in Artificial Intelligence 482*), 164–178.

[9] P. Clark & R. Boswell (1991). Rule induction with CN2: some recent improvements. In *Machine Learning — EWSL-91* (*Lecture Notes in Artificial Intelligence 482*), 151–163.

[10] P. Clark & T. Niblett (1989). The CN2 induction algorithm. *Machine Learning* 3, 261–283.

[11] T. H. Cormen, C. E. Leiserson, & R. Rivest (1990). *Introduction to Algorithms.* Cambridge, MA: MIT Press.

[12] L. Devroye (1988). Automatic pattern recognition: a study of the probability of error. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10, 530–543.

[13] M. R. Garey & D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY: W. H. Freeman and Company.

[14] F. Glover & M. Laguna (1993). Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems*, Colin R. Reeves (Ed.), 70–150. Oxford, MA: Blackwell Scientific Publications.

[15] F. Glover & D. de Werra (Eds.) (1993). *Annals of Operations Research, vol. 41: Tabu Search.*

[16] R. L. Graham, D. E. Knuth, & O. Patashnik (1989). *Concrete mathematics: a foundation for computer science.* Reading, MA: Addison-Wesley.

[17] D. Haussler (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence* 36, 177–221.

[18] D. Haussler (1992). Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation* 100, 78–150.

[19] Haussler, D., et al. (1988). Equivalence of models for polynomial learnability. In *Proceedings of the 1988 Workshop on Computational Learning Theory.*

[20] K.-U. Höffgen & H.-U. Simon (1992). Robust trainability of single neurons. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theorey*, 429–439. New York: Association for Computing Machinery.

[21] K.-U. Höffgen H.-U. Simon, & K. S. Van Horn (1994). Robust trainability of single neurons. To appear, *Journal of Computer and System Sciences.*

[22] V. Kann (1992). *On the Approximability of NP-complete Optimization Problems.* Ph.D. thesis, Royal Institute of Technology, Dept. of Numerical Analysis and Computing Science, Stockholm, Sweden.

[23] M. Kearns (1990). *The Computational Complexity of Machine Learning.* Cambridge, MA: MIT Press.

[24] M. Kearns, et al. (1987). On the learnability of boolean formula. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, 285–295.

[25] M. J. Kearns, R. E. Schapire, & L. M. Sallie (1992). Toward efficient agnostic learning. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, 341–353.

[26] R. Kerber (1992). ChiMerge: discretization of numerica attributes. In *Proceedings of the 10th National Conference on Artificial Intelligence*, 123–127.

[27] P. G. Kolaitis & M. N. Thakur (1991). Approximation properties of NP minimization classes. In *Proceedings of the 6th Annual Conference on Structures in Complexity Theory*, 353–366.

[28] J. van Leeuwen (editor) (1990). *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity.* Cambridge, MA: The MIT Press.

[29] N. Littlestone (1988). Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning* 2, 285–318.

[30] R. J. McEliece (1977). *The Theory of Information and Coding.* Reading, MA: Addison-Wesley.

[31] P. M. Murphy & D. W. Aha (1992.) *UCI Repository of machine learning databases.* Irvine, CA: University of California at Irvine, Dept. of Information and Computer Sciences.

[32] B. K. Natarajan (1991). *Machine Learning: A Theoretical Approach.* San Mateo, CA: Morgan Kaufmann.

[33] G. Pagallo & D. Haussler (1990). Boolean feature discovery in empirical learning. *Machine Learning* 5, 71–99.

[34] L. Pitt & L. Valiant (1988). Computational limitations on learning from examples. *J. Assoc. Comput. Mach.* 35, 965–984.

[35] J. R. Quinlan (1993). *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

[36] J. R. Quinlan & R. L. Rivest (1989). Inferring decision trees using the Minimum Description Length Principle. *Information and Computation* 80, 227–248.

[37] R. L. Rivest (1987). Learning decision lists. *Machine Learning* 2, 229–246.

[38] J. T. Tou & R. C. Gonzalez (1974). *Pattern Recognition Principles.* Reading, MA: Addison-Wesley.

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

[40] K. S. Van Horn & T. R. Martinez (1993). The Design and Evaluation of a Rule Induction Algorithm. Technical Report BYU-CS-93-11, Computer Science Department, Brigham Young University.

[41] K. S. Van Horn & T. R. Martinez (1993). The BBG rule induction algorithm. In *Proceedings of the 6th Australian Joint Conference on Artificial Intelligence*, Melbourne, Australia.

[42] V. N. Vapnik (1982). *Estimation of Dependences Based on Empirical Data.* New York: Springer-Verlag.

[43] V. N. Vapnik (1989). Inductive principles of the search for empirical dependences. In *Proceedings of the 2nd Annual Workshop on Computational Learning Theory.* San Mateo, CA: Morgan Kaufmann.

[44] S. M. Weiss, R. S. Galen, & P. V. Tadepalli (1990). Maximizing the predictive value of production rules. *Artificial Intelligence* 45, 47–71.