Machine Learning 2: 285–318, 1988

Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm

NICK LITTLESTONE (NICKL@SATURN.UCSC.EDU)
Department of Computer and Information Sciences, University of California,
Santa Cruz, CA 95064, U.S.A.

(Received: September 15, 1987)
(Revised: December 29, 1987)

Keywords: Learning from examples, prediction, incremental learning, mistake bounds, learning Boolean functions, linear-threshold algorithms

Abstract. Valiant (1984) and others have studied the problem of learning various classes of Boolean functions from examples. Here we discuss incremental learning of these functions. We consider a setting in which the learner responds to each example according to a current hypothesis. Then the learner updates the hypothesis, if necessary, based on the correct classification of the example. One natural measure of the quality of learning in this setting is the number of mistakes the learner makes. For suitable classes of functions, learning algorithms are available that make a bounded number of mistakes, with the bound independent of the number of examples seen by the learner. We present one such algorithm that learns disjunctive Boolean functions, along with variants for learning other classes of Boolean functions. The basic method can be expressed as a linear-threshold algorithm. A primary advantage of this algorithm is that the number of mistakes grows only logarithmically with the number of irrelevant attributes in the examples. At the same time, the algorithm is computationally efficient in both time and space.

1. Introduction

In this paper, we consider learning from examples in a situation in which the goal of the learner is simply to make few mistakes. The task is to induce a concept that can be described by a Boolean function; that is, the information received in each example is a list of Boolean attributes and the correct response is a Boolean function of the attributes. We are interested in cases where the correct-response function depends on only a small proportion of the attributes present in each example. For example, this case may occur in pattern recognition tasks; feature detectors may extract a large number of features for the learner's consideration, not knowing which few will prove useful. For another example, consider an environment in which the learner builds new concepts as Boolean functions of old concepts (Banerji, 1985; Valiant, 1984). Here the learner may need to sift through a large library of available concepts to find the suitable ones to use in expressing each new concept. In a special case of this situation, one may design a library of concepts specifically to ease learning of a certain class of complex functions. In this case one chooses concepts for the library that allow representation of any function in the class as a simple function of the library concepts. In the context of this paper, the concepts in the library will just be Boolean functions themselves. For example, consider k-DNF, the class of Boolean functions that can be represented in disjunctive normal form with no more than k literals per term (Valiant, 1985). If one has available intermediate concepts that include all conjunctions of no more than k literals, then any k-DNF function can be represented as a simple disjunction of these concepts. We will return to this idea at the end of the paper, presenting an algorithm for learning k-DNF.

Our main result is an algorithm that deals efficiently with large numbers of irrelevant attributes. If desired, it can be implemented within a neural net framework (Rumelhart & McClelland, 1986) as a simple linear-threshold algorithm. The method learns certain classes of functions that can be computed by a one-layer linear-threshold network; these include, among other functions, disjunctions, conjunctions, and r-of-k threshold functions (Hampson & Volper, 1986; Kearns, Li, Pitt, & Valiant, 1987a). (The latter functions are true if at least r out of k designated variables are true.) Preprocessing techniques can be used to extend the algorithm to classes of Boolean functions that are not linearly separable, such as k-DNF (for fixed k). When our algorithm is applied to k-DNF formulas with few terms, it makes significantly fewer mistakes than the algorithm presented by Valiant (1984, 1985). The algorithm is similar to classical perceptron algorithms, but it uses a multiplicative weight-update scheme that permits it to do much better than classical perceptron training algorithms when many attributes are irrelevant.

We study learning in an on-line setting. By this we mean that there is no separate set of training examples. The learner attempts to predict the appropriate response for each example, starting with the first example received. After making this prediction, the learner is told whether the prediction was correct, and then uses this information to improve its hypothesis. The learner continues to learn as long as it receives examples; that is, it continues to examine the information it receives in an effort to improve its hypothesis. In this setting, it is advantageous to use an algorithm that computes successive hypotheses incrementally, saving work that would be required to calculate every hypothesis from scratch from stored input examples. Our algorithm is incremental in this sense.

We evaluate the algorithm's learning behavior by counting the worst-case number of mistakes that it will make while learning a function from a specified class of functions. We also consider computational complexity. We will prove that the mistake bound of our algorithm is within a constant factor of optimal when the algorithm is applied to certain classes of functions. The method is also computationally time and space efficient.

Before we present the algorithm we will discuss some properties of mistake bounds for concept classes, including general lower bounds. We will also demonstrate a close relationship between exact identification with equivalence queries, as presented by Angluin (1987), and learning with a bounded number of mistakes.

The mistake bounds that we present are strong in the sense that they do not depend on any assumption about which examples the learner sees or the order in which it sees them: the selection and ordering can be done by an adversary. However, due to the freedom given the adversary, we cannot say how early the learner will make the mistakes. For example, a single instance could be repeated arbitrarily many times at the beginning of the sequence of trials and then followed by other instances for which the learner does not yet know how to respond.

One can adapt mistake-bounded algorithms to work well according to criteria that are useful in other settings. For example, consider a setting in which the learning process is separated into two phases: a training phase and a subsequent working phase. Learning occurs only during the training phase; mistakes are counted only during the working phase. Thus the only important hypothesis is the one formed by the learner at the conclusion of the training phase. One useful model in this context is the probabilistic model introduced by Valiant (1984) and discussed by Blumer, Ehrenfeucht, Haussler, and Warmuth (1987a, 1987b) and Angluin (1987). Starting with a mistake-bounded algorithm, one can derive an algorithm that does well under the criteria of this probabilistic model. We mention one indirect way to do this, using Angluin's (1987) results. Kearns, Li, Pitt, and Valiant (1987b) have mentioned a related technique.

Another change that one might make to the learning model involves keeping the on-line setting, but analyzing it with probabilistic instead of worst-case assumptions. One can use the probabilistic model mentioned above to this end. Haussler, Littlestone, and Warmuth (1987) discuss a related model developed particularly for this setting.

It is interesting to compare our main algorithm to similar classical methods for perceptron training. Hampson and Volper (1986) present empirical evidence that, for one classical perceptron algorithm, the number of mistakes grows linearly with the number of irrelevant attributes. This is in keeping with theoretical bounds from the perceptron convergence theorem

(Hampson & Volper, 1986; Duda & Hart, 1973; Nilsson, 1965). We know of no evidence that any other standard perceptron algorithm does better. In contrast, we will prove that the number of mistakes that our algorithm makes grows only logarithmically with the number of irrelevant attributes.

Others have looked at the problem of dealing efficiently with irrelevant attributes in the context of learning Boolean functions. Haussler (1986) mentions two algorithms for learning disjunctive functions in the context of Valiant's learning model. One of them is designed to learn rapidly in the presence of irrelevant attributes. However, that algorithm is not naturally incremental, and thus is significantly less time and space efficient than ours when used in an on-line setting. Valiant (1984, 1985) introduces a mechanism by which a friendly and knowledgeable teacher can help the learner by indicating which attributes are relevant. Hampson and Volper (1986), in addition to their study of classical perceptron algorithms, have experimented with new algorithms that use conditional probabilities in an effort to reduce the cost of irrelevant attributes. They do not present theoretical bounds for these algorithms.

The mistake-counting model that we use is essentially the same as a model discussed in Barzdin and Freivald (1972). See Angluin and Smith (1983) for a survey that compares a number of learning models.

2. The setting

In this section we will describe in more detail the learning environment that we consider and the classes of functions that our algorithm can learn. We assume that learning takes place in a sequence of trials. The order of events in a trial is as follows:

(1) The learner receives some information about the world, corresponding to a single example. This information consists of the values of n Boolean attributes, for some n that remains fixed. We think of the information received as a point in $\{0,1\}^n$. We call this point an *instance* and we call $\{0,1\}^n$ the *instance space.*

(2) The learner makes a response. The learner has a choice of two responses, labeled 0 and 1. We call this response the learner's *prediction* of the correct value.

(3) The learner is told whether or not the response was correct. This information is called the *reinforcement.*

Each trial begins after the previous trial has ended.

We assume that for the entire sequence of trials, there is a single function $f : \{0,1\}^n \rightarrow \{0,1\}$ which maps each instance to the correct response to that instance. We call this function the *target function* or *target concept.*

We call an algorithm for learning in this setting an *algorithm for on-line learning from examples.* When we speak of learning algorithms without further qualification we refer to algorithms for on-line learning from examples. For this paper we restrict our attention to deterministic algorithms.

We will present mistake bounds as worst case bounds over some class of possible target functions, which we will call the *target class.*

3. The nature of absolute mistake bounds

In this section we give some general results about mistake bounds for online learning from examples. We present upper and lower bounds on the number of mistakes in the case where one ignores issues of computational efficiency. The instance space can be any finite space X, and the target class is assumed to be a collection of functions, each with domain X and range $\{0, 1\}$. The results also apply to infinite X, provided that the target class remains finite. However, computability issues may arise in this case, and we do not consider them here.

For any learning algorithm A and any target function f, let $M_A(f)$ be the maximum over all possible sequences of instances of the number of mistakes that algorithm A makes when the target function is f. For any learning algorithm A and any non-empty target class C, let $M_A(C) = \max_{f \in C} M_A(f)$.[1] Define $M_A(C) = -1$ if C is empty. Any number greater than or equal to $M_A(C)$ will be called a *mistake bound* for algorithm A applied to class C.

Definition 1 The *optimal mistake bound* for a target class C, denoted $opt(C)$, is the minimum over all learning algorithms A of $M_A(C)$. This minimum is taken over all algorithms regardless of their computational efficiency. An algorithm A is called *optimal* for class C if $M_A(C) = opt(C)$. Thus $opt(C)$ represents the best possible worst case mistake bound for any algorithm learning C.

If computational resources are no issue, there is a straightforward learning algorithm that has excellent mistake bounds for many classes of functions. This algorithm uses the idea of repeated halving of the set of plausible hypotheses. This idea appears in various forms in Barzdin and Freivald (1972), Mitchell (1982), and Angluin (1987). We restate it in the current context because it gives an upper limit on the mistake bound and because it suggests strategies that one might explore in searching for computationally efficient algorithms.

[1] Some algorithms that we will describe are general algorithms whose functioning depends on knowledge of the particular target class for which they are being used. For such an algorithm A, we will use $M_A(C)$ to denote $\max_{f \in C} M_A(f)$ when A is told that the target class is C.

Algorithm 1 (halving algorithm)

The halving algorithm can be applied to any finite class C of functions taking values in $\{0,1\}$. It maintains a list of all of the functions in the class that agree with the target function on all past instances. We will call the functions on this list the *consistent* functions. In the terminology of Mitchell (1982), the consistent functions form the current *version space* of the algorithm. Initially the list contains all of the functions in the class. To respond to a new instance, the algorithm computes the values of all consistent functions at the new instance, and makes the prediction that agrees with the majority (or either possibility in case of a tie). Following each trial, the algorithm updates the list of consistent functions.

We will now give a second description of the halving algorithm to introduce notation that we will use later. Given a target class C and a point x in the associated instance space X, let $\xi_0(C,x)$ denote the subset of C containing those functions that are 0 at x, and let $\xi_1(C,x)$ denote those functions in C that are 1 at x.

The halving algorithm maintains a variable *CONSIST* whose value is a set containing all functions in C that are consistent with all past instances. Initially $\mathit{CONSIST} = C$. When the halving algorithm receives an instance, it determines the sets $\xi_0(\mathit{CONSIST},x)$ and $\xi_1(\mathit{CONSIST},x)$. If $|\xi_1(\mathit{CONSIST},x)| > |\xi_0(\mathit{CONSIST},x)|$ then the algorithm predicts 1; otherwise it predicts 0. When the algorithm receives the reinforcement, it sets *CONSIST* accordingly: if the correct response to x is 0 then it sets *CONSIST* to $\xi_0(\mathit{CONSIST},x)$; otherwise it sets *CONSIST* to $\xi_1(\mathit{CONSIST},x)$. Let $M_{HALVING}(C)$ denote the maximum number of mistakes that the algorithm will make when it is run for the target class C (i.e., its initial list of functions consists of C) and the target function in fact comes from C.

Theorem 1 *For any non-empty target class C, $M_{HALVING}(C) \leq \log_2 |C|$.*
PROOF: Since there are only two possible predictions, the learner will always be able to choose a prediction agreed to by at least half of the current list of consistent functions. Whenever a mistake occurs, those functions that agree with the prediction of the learner will be eliminated from the list of consistent functions; these functions constitute at least half of the list. Thus at each mistake the size of the list will be divided by at least two. Since we have assumed that the target function is in the initial class of functions, there will always be at least one consistent function. Thus the method can make at most $\log_2 |C|$ mistakes. ■

The theorem above also holds for a modified version of the halving algorithm in which *CONSIST* is only changed following trials in which mistakes occur. The same proof applies in this case. The halving algorithm immediately gives us the following theorem:

Theorem 2 *For any finite target class C, $opt(C) \leq \log_2 |C|$.* ■

Example 1 Note that for some classes of functions this bound is not tight. For example, for $x \in \{0,1\}^n$ let $g_x : \{0,1\}^n \to \{0,1\}$ be the function that is 1 at x and 0 elsewhere. Then one can easily verify that the halving algorithm applied to the class of functions $\{g_x\}_{x \in \{0,1\}^n}$ will make at most one mistake.

Now we will study $opt(C)$ more closely. To do this we need the following definitions.

Definition 2 A *mistake tree* for a target class C over an instance space X is a binary tree each of whose nodes is a non-empty subset of C and each of whose internal nodes is labeled with a point of X, which satisfies the following:

(1) The root of the tree is C.

(2) Given any internal node C' labeled with x, the left child of C', if present, is $\xi_0(C', x)$, and the right child, if present, is $\xi_1(C', x)$.

For example, Figure 1 shows the mistake tree for C when $X = \{0,1\}^5$ and C consists of the functions $f_i(x_1, \ldots, x_5) = x_i$, for $i = 1, \ldots, 5$.

A *complete k-mistake tree* is a mistake tree that is a complete binary tree of height k. We define the height of a tree to be the length in edges of the longest path from the root. The tree above is a complete 2-mistake tree. These trees provide a way to characterize the number of mistakes made by an optimal learning algorithm. We will present an optimal algorithm, and then discuss the number of mistakes that it makes.

For any non-empty finite target class C, let $K(C)$ equal the largest integer k such that there exists a complete k-mistake tree for C. The definition of mistake trees guarantees a finite upper bound to k. Let $K(\emptyset) = -1$.

Algorithm 2 (standard optimal algorithm)

The standard optimal algorithm is similar to the halving algorithm. It maintains the variable *CONSIST* in the same manner, and like the halving algorithm examines $\xi_0(CONSIST, x)$ and $\xi_1(CONSIST, x)$ to determine its prediction. The only difference from the halving algorithm lies in the rule it uses to choose its prediction. Instead of predicting according to which of these sets of functions is larger, it compares $K(\xi_0(CONSIST, x))$ with $K(\xi_1(CONSIST, x))$. If $K(\xi_1(CONSIST, x)) > K(\xi_0(CONSIST, x))$ then the algorithm responds 1; otherwise it responds 0. Thus whenever a mistake occurs, the remaining consistent functions have the smaller maximal complete mistake tree.

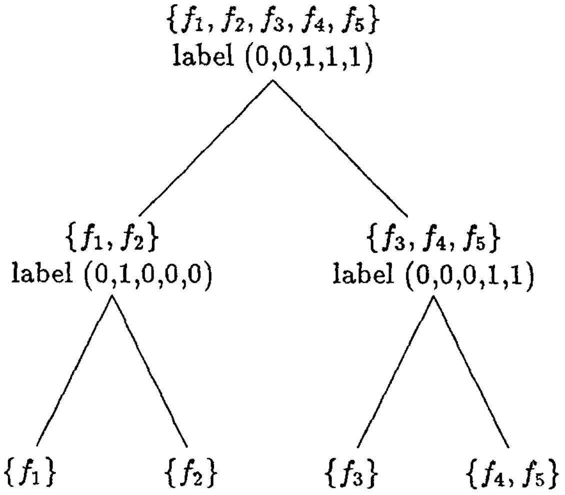

Figure 1. A complete 2-mistake tree.

Theorem 3 *Let X be any instance space. Let SOA denote the standard optimal algorithm defined above, and let C be any finite class of functions with domain X and range $\{0, 1\}$. Then*

$$opt(C) = M_{SOA}(C) = K(C).$$

We will prove this theorem using the following two lemmas:

Lemma 1 *For any target class C,*

$$opt(C) \geq K(C).$$

PROOF: This follows trivially from the definition if C is empty. Assume C is non-empty, and let $k = K(C)$. Saying that $opt(C) \geq k$ is equivalent to saying that for any deterministic learning algorithm A, there exists a function $f \in C$ and a sequence of instances such that A makes at least k mistakes when presented with that sequence of instances. Given an algorithm A, we will show how an adversary can choose a function and a sequence of instances such that A makes at least k mistakes. The adversary keeps track of a current mistake tree. Initially this is a complete k mistake tree for C. If $k = 0$, the lemma follows trivially. Otherwise, the first instance chosen by the adversary is the label of the root of the tree. Whatever the algorithm predicts, the adversary tells the algorithm that its prediction is wrong. This response of the adversary eliminates some functions as possible target functions. The remaining candidate functions are either the class $\xi_0(C, x)$ or the class $\xi_1(C, x)$, depending on the algorithm's prediction and the adversary's response to it. One of the two subtrees of the root of the adversary's current mistake tree is a complete $k-1$ mistake tree for the remaining candidate functions. The adversary sets its current mistake tree to that subtree. It chooses the next instance to be the label of the root of the new current tree. The adversary continues in this manner, forcing the algorithm to be wrong at each instance. After j mistakes, the adversary's current tree is a complete $k-j$ mistake tree for the remaining candidate target functions. As long as $j < k$, the root of the current tree has two children corresponding to non-empty subclasses of C; thus the adversary can choose a point (the label of the root) at which it can force the algorithm to make a mistake. When $j = k$, k mistakes have been made, as desired. The target function chosen by the adversary can be any candidate remaining after the last mistake was made. ∎

Lemma 2 *Let C be a finite non-empty target class. Suppose that SOA is run to learn some function in C and that the sequence of instances it receives is $x_1, \ldots, x_t$. Consider the variable CONSIST maintained by SOA. Let $CONSIST_i$ denote the value of CONSIST at the start of trial i. For any $k \geq 0$ and i in $\{1, \ldots, t\}$, if $K(CONSIST_i) = k$, then SOA will make at most k mistakes during trials $i, \ldots, t$.*

PROOF: We prove this by induction on k, taking $k = 0$ to be the base case. By the construction of *SOA*, the target function will always be in $CONSIST_i$. If $K(CONSIST_i) = 0$ then $CONSIST_i$ can contain only the target function. (If there are two functions in $CONSIST_i$, then any instance on which they differ is the label of the root of a complete 1-mistake tree for $CONSIST_i$.) The definition $K(\emptyset) = -1$ ensures that *SOA* will always respond correctly when $CONSIST_i$ contains only the target function. This proves the base case of the induction.

Now we will prove the lemma for arbitrary $k > 0$, assuming that it holds for $k - 1$. If *SOA* makes no mistakes during trials $i, \ldots, t-1$ then we are done. Otherwise, let j be the number of the first trial among trials $i, \ldots, t-1$ at which *SOA* makes a mistake. If there are complete k-mistake trees for both $\xi_0(CONSIST_j, x_j)$ and $\xi_1(CONSIST_j, x_j)$, then we can combine them into a complete $k+1$ mistake tree for $CONSIST_j$; we add a root node labeled with x_j. Since $CONSIST_j \subseteq CONSIST_i$ it is easy to transform this into a complete $k+1$-mistake tree for $CONSIST_i$. But we have assumed that there does not exist a complete $k+1$-mistake tree for $CONSIST_i$. Thus at least one of $K(\xi_0(CONSIST_j, x_j))$ and $K(\xi_1(CONSIST_j, x_j))$ must be less than k. Since the response of *SOA* corresponded to the larger of these two values for K, and since *SOA* was wrong, $CONSIST_{j+1}$ will have the property that $K(CONSIST_{j+1}) < k$. By the induction hypothesis, *SOA*

Table 1. Values of nine functions in Example 2.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
a_1	1	0	0	0	0	0	0	0	0
a_2	0	1	0	0	0	0	0	0	0
a_3	0	0	1	0	0	0	0	0	0
a_4	0	0	0	1	0	0	0	0	0
a_5	0	0	0	0	1	0	0	0	0
a_6	0	0	0	0	0	1	1	1	1
a_7	0	0	0	0	0	0	0	1	1
a_8	0	0	0	0	0	0	1	0	1

will make at most $k-1$ mistakes during trials $j+1,\ldots,t$. This gives the desired result. ■

PROOF OF THEOREM 3: If we set $k = K(C)$ and $i = 1$ in Lemma 2 we get $M_{SOA}(C) \leq K(C)$. Lemma 1 states $K(C) \leq opt(C)$. From the definition of $opt(C)$ we have $opt(C) \leq M_{SOA}(C)$. The theorem follows. ■

One of the consequences of this theorem is that we could use *opt* instead of K in the description of *SOA* and obtain the same algorithm.

Note that Example 1 shows that there are arbitrarily large target classes C for which $opt(C) = 1$. Using this, one can construct a target class C for which there is some point x such that

$$|\xi_0(C,x)| > |\xi_1(C,x)|$$

but

$$opt(\xi_0(C,x)) < opt(\xi_1(C,x)).$$

For such a target class, if the point x is the first instance, then the standard optimal algorithm and the halving algorithm will make different predictions for x. Let us consider an example of such a target class for which the halving algorithm is not optimal.

Example 2 Let the instance space X be an eight element set $\{a_1,\ldots,a_8\}$. Let the target class C consist of nine functions $f_1,\ldots,f_9$, with values shown in Table 1. If the first three instances received by the halving algorithm are a_6, a_7, a_8 in that order, then there is some target function for which the halving algorithm will make three mistakes. (If we use the version of the halving algorithm that chooses 0 in case of a tie, then the halving algorithm will make three mistakes for target function f_9.) On the other hand, there is no sequence of points and target function for which *SOA* will make more than 2 mistakes. One can see this by considering each point of the instance space in turn. For every $x \in X$ either $opt(\xi_0(C,x)) \leq 1$

or $opt(\xi_1(C,x)) \leq 1$. Thus no matter on which instance *SOA* makes its first mistake, its prediction will have been chosen so that the remaining consistent functions have an optimal mistake bound of at most one. Hence the halving algorithm is not optimal for this target class.

Now we give a lower bound for $opt(C)$ in terms of the Vapnik-Chervonenkis (Vapnik & Chervonenkis, 1971) dimension of C, which is a combinatorial parameter that has proven useful in other studies of learning (Vapnik, 1982; Blumer et al., 1987a; Haussler, Littlestone, & Warmuth, 1987).[2] To define the Vapnik-Chervonenkis dimension, we use the notion of a shattered set.

Definition 3 A set $S \subseteq X$ is *shattered* by a target class C if for every $U \subseteq S$ there exists a function $f \in C$ such that f is 1 on U and 0 on $S - U$.

Definition 4 The *Vapnik-Chervonenkis dimension* of a non-empty target class C is the cardinality of the largest set that is shattered by C. We will denote this $VCdim(C)$. We will define $VCdim(\emptyset) = -1$.

Theorem 4 *For any target class C, $VCdim(C) \leq opt(C)$.*

PROOF: Let $k = VCdim(C)$. Choose any set $\{v_1, \ldots, v_k\} \subseteq X$ that is shattered by C. Then we can construct a complete k-mistake tree for C with all internal nodes at depth j labeled with v_{j+1} for $j = 0, 1, \ldots, k-1$. The nodes are chosen to be subclasses of C as required in the definition of a mistake tree. These subclasses will be all non-empty (as required by the definition) by virtue of the fact that $\{v_1, \ldots, v_k\}$ is shattered by C. ■

The Vapnik-Chervonenkis dimension will prove to be a useful lower bound on $opt(C)$ for concept classes that we will consider in later sections of the paper. However, there are also concept classes for which the Vapnik-Chervonenkis dimension is a very weak lower bound. In fact, as the following example shows, $opt(C)$ can be arbitrarily large for classes for which $VCdim(C) = 1$.

Example 3 For $n > 0$, take $X = \{1, \ldots, 2^n - 1\}$. For each $j \in \{1, \ldots, 2^n\}$ let $f_j : X \rightarrow \{0, 1\}$ be the function such that $f_j(x) = 1$ if and only if $x < j$. Let $C = \{f_j : 1 \leq j \leq 2^n\}$. Then $VCdim(C) = 1$ but $opt(C) = n$. To see this, first note that for any $f \in C$ if $f(x) = 1$ then for all $y < x$, $f(y) = 1$. Thus no set of size 2 is shattered and $VCdim(C) = 1$. Also, by Theorem 2, $opt(C) \leq \log_2 |C| = n$. To see that $opt(C) \geq n$ we can construct a complete n-mistake tree. Label the root with the point 2^{n-1}. We have $\xi_0(C, 2^{n-1}) = \{f_1, \ldots, f_{2^{n-1}}\}$ and $\xi_1(C, 2^{n-1}) = \{f_{2^{n-1}+1}, \ldots, f_{2^n}\}$. Each of these two subclasses is similar to the original class but half as large.

[2] In Vapnik '1982), the Vapnik-Chervonenkis dimension is called the *capacity*.

It is easy to see that points can be found to be the labels of the children of the root that will split each of the subclasses exactly in two. This line of reasoning can be formalized to yield an inductive construction of the mistake tree.

4. General transformations

There is a close relationship between learning algorithms of the type that we have been considering and those that exactly identify a target function using a bounded number of *equivalence queries*, as described by Angluin (1987). An equivalence query is a request by an algorithm that asks if the target function matches some function described in the query. Whenever an algorithm receives a negative answer to an equivalence query, it also receives a counterexample, i.e., a point at which the target function and the proposed function disagree. The equivalence query algorithms that we consider here receive no examples as input other than the counterexamples to the queries. In this section, we will use the term "query algorithm" to refer to an algorithm that learns using equivalence queries, and the terms "on-line learning algorithm", "mistake-bounded algorithm", and "algorithm for learning from examples" to refer to algorithms of the type discussed elsewhere in this paper.

To describe the relationship between equivalence query algorithms and our model, we must define the notion of the *current hypothesis* of an algorithm for on-line learning from examples. The current hypothesis is defined initially and between trials, and is a function from the instance space to $\{0, 1\}$. Its value at any instance x is defined to be the response that the algorithm would give at the next trial if the instance received in the next trial were x. This is well-defined for any deterministic algorithm. If we copy the state of an algorithm at the conclusion of a trial, then we can use the copy of the state to determine (by simulating the algorithm) what prediction the algorithm would make for any new instance, without sending that instance to the running version of the algorithm. Thus the state can be considered a representation of the current hypothesis of the algorithm. (Often a portion of the state will suffice.) Using this representation to represent the functions appearing in queries, an algorithm that learns from examples can be transformed into a query algorithm. We will show that the number of queries needed will be at most one more than the number of mistakes that the learning-from-examples algorithm would make.[3]

[3] Note that for most of Angluin's results, the queries are restricted to use only functions from the target class in question. For the conversion here, the functions used in the queries must be allowed to come from the class of functions that the original algorithm uses for its hypotheses. Also note that with this transformation, the functions used in

The inverse transformation is also possible: a query algorithm can be transformed into an algorithm that learns from examples making a bounded number of mistakes. The efficiency of the transformed algorithm will depend on the difficulty of evaluating the functions given in the queries. The number of mistakes made by the transformed algorithm is bounded by the number of queries used by the query algorithm. We now give the details of these transformations.

Algorithm transformation 1 Given a mistake-bounded learning algorithm A, this transformation yields a query algorithm B for the same target class. The first query of the derived algorithm B is the initial hypothesis of algorithm A. Algorithm B waits for a response to this query and then repeats the following for the first response and the response to each subsequent query: if the response indicates that the query specified the correct target function, then algorithm B halts and reports the correct target function; otherwise, the response to the latest query includes a counterexample. The derived algorithm gives this instance to algorithm A. After receiving A's prediction, it tells A that the prediction was incorrect. (Algorithm B knows that A will be wrong here, since the last query was just the current hypothesis of A, and by definition the current hypothesis tells how A will respond to the next instance.) Algorithm B takes the new hypothesis of algorithm A and uses it as the next query, continuing in this fashion until it determines the correct target function.

Since every query after the first results from a mistake of A, we have the following theorem:

Theorem 5 *The number of queries needed by the derived algorithm to exactly identify target function f is bounded by $M_A(f) + 1$.* ■

Algorithm transformation 2 Now suppose we are given a query algorithm A that achieves exact identification of every function in some target class C with a bounded number of queries. This transformation yields a mistake-bounded learning algorithm B for the same target class. The initial hypothesis of algorithm B is the hypothesis output by the query algorithm as its initial query. Algorithm B uses this hypothesis to respond to all instances that are received until it is told that it has made a mistake. Until the first mistake, algorithm A receives no response to its first query. At the time of the first mistake, algorithm B gives algorithm A a response to its query: it tells A that its hypothesis was wrong, and reports that the

the queries will not necessarily be given a compact symbolic representation. However, if the query algorithm is derived from a computationally efficient algorithm for on-line learning from examples, then the query functions will be represented in a form that can be efficiently evaluated.

instance at which a mistake was made is a counterexample. Algorithm B now waits to make any further predictions until A either makes another query or halts and reports the correct target function. Since A achieves exact identification, one of these events will occur. The hypothesis given in the Pnew query (or the reported target function) becomes the new current hypothesis of algorithm B. The derived algorithm B proceeds in this manner indefinitely.

The next theorem follows immediately.

Theorem 6 *For any target function $f \in C$, the number of mistakes made by the derived algorithm in learning f is bounded by the number of queries needed by algorithm A to exactly identify f.* ■

One can also convert a mistake-bounded algorithm into an algorithm that learns effectively in the probabilistic model introduced by Valiant (1984) and described by Blumer et al. (1987a, 1987b). Angluin (1987) refers to this model as *pac*-learning, where *pac* stands for "probably approximately correct." One way to perform the conversion essentially follows a method discussed by Kearns, Li, Pitt, and Valiant (1987b) for using failure bounds to derive probabilistic learning results. Alternatively, one can use an indirect route: one can convert a mistake-bounded algorithm into an algorithm for exact identification using equivalence queries, and then use a conversion described by Angluin (1987) to obtain an algorithm for the probabilistic setting.

Other general algorithm transformations are possible. Sometimes it is useful to have an algorithm that changes its hypothesis only when a mistake occurs; Haussler (1985) has referred to such methods as *conservative.* One can transform a mistake-bounded algorithm into a conservative algorithm with the same mistake bound. Haussler (1985) has referred to such methods as *failure-bounded.* One way to convert a mistake-bounded algorithm to a conservative algorithm is to use the above transformations to convert it first to an equivalence query algorithm and thence back to a mistake-bounded algorithm. The mistake bound increases by one if the above theorems about the transformations are applied as they stand. With more careful analysis of the double conversion, the increase disappears. The conversion to a conservative algorithm is also straightforward to perform directly.

5. The linear-threshold algorithm

Now we describe our main algorithm, first describing the classes of target functions. We will consider *linearly-separable* Boolean functions, which are those functions that can be computed by a one-layer linear-threshold network such as a perceptron. A function from $\{0,1\}^n$ to $\{0,1\}$ is said to be linearly separable if there is a hyperplane in $\mathbf{R}^n$ separating the points on which the function is 1 from the points on which it is 0. Monotone disjunctions constitute one class of linearly-separable functions.

Definition 5 A *monotone disjunction* is a disjunction in which no literal appears negated, that is, a function of the form

$$f(x_1, \ldots, x_n) = x_{i_1} \vee \cdots \vee x_{i_k}.$$

The hyperplane given by $x_{i_1} + \cdots + x_{i_k} = 1/2$ is a separating hyperplane for $f(x_1, \ldots, x_n) = x_{i_1} \vee \cdots \vee x_{i_k}$. We will present two variants of our algorithm. The first variant, which we now present, is specialized for learning monotone disjunctions. We will later describe a simple transformation to remove the monotone restriction.

Algorithm 3 (WINNOW1)

We call this algorithm "WINNOW" because it has been designed for efficiency in separating relevant from irrelevant attributes. We will present the algorithm as a linear-threshold algorithm. The instance space is $X = \{0, 1\}^n$. The algorithm maintains non-negative real-valued weights $w_1, \ldots,$ w_n, each having 1 as its initial value. The algorithm also makes use of a real number θ, which we call the *threshold*. When the learner receives an instance $(x_1, \ldots, x_n)$, the learner responds as follows:

- If $\sum_{i=1}^{n} w_i x_i > \theta$, then it predicts 1;
- If $\sum_{i=1}^{n} w_i x_i \leq \theta$, then it predicts 0.

The choice of prediction when $\sum_{i=1}^{n} w_i x_i = \theta$ is not critical for our results.

The weights are changed only if the learner makes a mistake, and then only the weights corresponding to non-zero x_i are changed. The amount by which the weights are changed depends on a fixed parameter $\alpha > 1$. Good bounds are obtained if θ is set to $n/2$ and α is set to 2. We will say more about the values of α and θ later. Table 2 describes the changes made to the weights in response to different combinations of prediction and reinforcement. The threshold is left fixed.

Note in Table 2 that we have given each type of update action a name; each mistake corresponds to a single *promotion* step or to a single *elimination* step. The space needed (without counting bits per weight) and the sequential time needed per trial are both clearly linear in n. Note that the non-zero weights are powers of α. We will prove that the weights are at most $\alpha\theta$. Thus if the logarithms (base α) of the weights are stored, only

Table 2. WINNOW1's response to mistakes.

learner's prediction	*correct response*	*update action*	*update name*
1	0	$w_i := 0$ if $x_i = 1$ w_i unchanged if $x_i = 0$	elimination step
0	1	$w_i := \alpha \cdot w_i$ if $x_i = 1$ w_i unchanged if $x_i = 0$	promotion step

$O(\log_2 \log_\alpha \theta)$ bits per weight are needed. The running time needed to calculate predictions and changes to the weights could be reduced greatly by parallel implementation, such as with an appropriately constructed neural net. For a mistake bound we give the following theorem.

Theorem 7 *Suppose that the target function is a k-literal monotone disjunction given by $f(x_1, \ldots, x_n) = x_{i_1} \vee \cdots \vee x_{i_k}$. If* WINNOW1 *is run with $\alpha > 1$ and $\theta \geq 1/\alpha$, then for any sequence of instances the total number of mistakes will be bounded by $\alpha k(\log_\alpha \theta + 1) + \frac{n}{\theta}$.*

For example, if $\theta = n$ and $\alpha = 2$ then the mistake bound is $2k(\log_2 n + 1) + 1$. If we set $\theta = \frac{n}{\alpha}$, the bound simplifies to $\alpha k \log_\alpha n + \alpha$. For $\alpha = 2$ this gives a bound of $2k \log_2 n + 2$. The dominating first term is minimized for $\alpha = e$; the bound then becomes $\frac{e}{\log_2 e} k \log_2 n + e < 1.885 k \log_2 n + e$.

We will prove this theorem by finding bounds on the number of promotion and elimination steps that occur. First we give three lemmas used in the proof.

Let u be the number of promotion steps that have occurred by the end of some sequence of trials and let v be the number of elimination steps that have occurred by the end of the same sequence of trials.

Lemma 3 $v \leq \frac{n}{\theta} + (\alpha - 1)u$.

PROOF: Consider how the sum $\sum_{i=1}^{n} w_i$ changes over time. Initially the sum is n; promotion and elimination steps cause it to change. Each promotion steps increases this sum by at most $(\alpha - 1)\theta$, since when a promotion step occurs we have $\sum_{i|x_i=1} w_i \leq \theta$. Each elimination step decreases $\sum_{i=1}^{n} w_i$ by at least θ. Since the sum is never negative we have

$$0 \leq \sum_{i=1}^{n} w_i \leq n + \theta(\alpha - 1)u - \theta v,$$

giving the desired result. ■

Lemma 4 *For all i, $w_i \leq \alpha\theta$.*

PROOF: Since $\theta \geq 1/\alpha$, the weights are initially less than or equal to $\alpha\theta$. For any j, the value of w_j is only increased during a trial in which $x_j = 1$ and $\sum_i^n w_i x_i \leq \theta$. These conditions can only occur together if $w_j \leq \theta$ immediately prior to the promotion. Thus $w_j \leq \alpha\theta$ after the promotion. ■

Lemma 5 *After u promotion steps and an arbitrary number of elimination steps, there exists some i for which $\log_\alpha w_i \geq u/k$.*

PROOF: Let $R = \{i_1, \ldots, i_k\}$. We look at how the product $\prod_{i \in R} w_i$ is changed by elimination and promotion steps. Note that $f(x_1, \ldots, x_n) = 0$ if and only if $x_i = 0$ for all $i \in R$. Elimination steps occur only when $f(x_1, \ldots, x_n) = 0$; promotion steps occur only when $f(x_1, \ldots, x_n) = 1$. Thus $\prod_{i \in R} w_i$ is unchanged by elimination steps and is increased by a factor of at least α by each promotion step. Initially $\prod_{i \in R} w_i = 1$. Thus after u promotion steps $\prod_{i \in R} w_i \geq \alpha^u$, giving $\sum_{i \in R} \log_\alpha w_i \geq u$. Since $|R| = k$, for some $i \in R$ we have $\log_\alpha w_i \geq u/k$, as desired. ■

Note that only the last of these lemmas depends on the form of the target function and it is only there that k appears.

PROOF OF THEOREM 7: The total number of mistakes made during a run of the algorithm is equal to the number of promotion steps, u, plus the number of elimination steps, v. We bound u using the last two lemmas and then use the first lemma to bound v. Combining lemmas 4 and 5 we see that

$$u/k \leq \log_\alpha w_i \leq \log_\alpha \theta + 1,$$

or

$$u \leq k(\log_\alpha \theta + 1).$$

Lemma 3 now gives

$$v \leq \frac{n}{\theta} + (\alpha - 1)k(\log_\alpha \theta + 1).$$

Adding the bounds on u and v leads to the desired bound on the total number of mistakes. ■

Note that the above algorithm does not depend on k. Thus the algorithm can learn the entire target class of monotone disjunctions without modification. The mistake bound depends on the number of literals in the actual target concept. Now for $1 \leq k \leq n$, let $\tilde{C}_k$ denote the class of k-literal monotone disjunctions, and let C_k denote the class of all those monotone disjunctions that have at most k-literals. Suppose one wants to specialize the algorithm to learn the target class C_{k_0} efficiently for a particular k_0. If one chooses $\theta = \frac{n}{\alpha k_0}$, then the mistake bound becomes

$$\alpha k \log_\alpha \frac{n}{k_0} + \alpha k_0 \leq \alpha k_0 (1 + \log_\alpha \frac{n}{k_0})$$

when the target function is a k-literal monotone disjunction in C_{k_0}. For $\alpha = 2$ this gives a bound of $2k_0(1 + \log_2 \frac{n}{k_0})$. For $\alpha = e$ we obtain the bound $k_0(e + 1.885 \log_2 \frac{n}{k_0})$.

We now give a lower bound on the number of mistakes needed to learn $\tilde{C}_k$ and C_k.

Theorem 8 (lower bound) *For $1 \leq k \leq n$, $opt(C_k) \geq opt(\tilde{C}_k) \geq k\lfloor \log_2 \frac{n}{k} \rfloor$. For $n > 1$ we also have $opt(C_k) \geq \frac{k}{8}(1 + \log_2 \frac{n}{k})$.*

The second form gives a formula directly comparable to the upper bound above. When the above algorithm is specialized for a particular C_k and when $n > 1$, the algorithm is within a constant factor of being optimal.

PROOF: Since $\tilde{C}_k \subseteq C_k$, it is clear that $opt(C_k) \geq opt(\tilde{C}_k)$. By Theorem 4, any algorithm to learn a concept class will have a mistake bound at least equal to the Vapnik-Chervonenkis dimension of the concept class. In the following lemma we show that the Vapnik-Chervonenkis dimension of $\tilde{C}_k$ is bounded below by $k\lfloor \log_2 \frac{n}{k} \rfloor$. This gives the first part of the theorem. We will split the derivation of the second formula from the first into two cases, depending on whether or not $k \leq \frac{n}{2}$. If $k \leq \frac{n}{2}$ then $\log_2 \frac{n}{k} \geq 1$. Thus

$$k\lfloor \log_2 \frac{n}{k} \rfloor \geq \frac{k}{3}(2\lfloor \log_2 \frac{n}{k} \rfloor + \log_2 \frac{n}{k} - 1) \geq \frac{k}{3}(2 + \log_2 \frac{n}{k} - 1) \geq \frac{k}{8}(1 + \log_2 \frac{n}{k})$$

as desired. If $n \geq k > \frac{n}{2}$, then $opt(C_k) \geq opt(\tilde{C}_{\lfloor \frac{n}{2} \rfloor})$ since $\tilde{C}_{\lfloor \frac{n}{2} \rfloor} \subseteq C_k$. (Here we use the assumption that $n > 1$.) We have

$$opt(\tilde{C}_{\lfloor \frac{n}{2} \rfloor}) \geq \lfloor \frac{n}{2} \rfloor \lfloor \log_2 \frac{n}{\lfloor \frac{n}{2} \rfloor} \rfloor \geq \lfloor \frac{n}{2} \rfloor \lfloor \log_2 2 \rfloor = \lfloor \frac{n}{2} \rfloor \geq \frac{n}{2} - \frac{1}{2}.$$

We also have

$$\frac{k}{8}(1 + \log_2 \frac{n}{k}) \leq \frac{n}{8}(1 + \log_2 2) = \frac{n}{4}.$$

For $n \geq 2$, this is less than or equal to $\frac{n}{2} - \frac{1}{2}$, giving the desired result. ∎

We will prove a more general lemma than is needed here, since it will give us results that will be useful later. Note that k-literal monotone conjunctions are just 1-term monotone k-DNF formulas. Also l-literal monotone disjunctions are l-term monotone 1-DNF formulas.

Lemma 6 *For $1 \leq k \leq n$ and $1 \leq l \leq \binom{n}{k}$, let C be the class of functions expressible as l-term monotone k-DNF formulas and let m be any integer, $k \leq m \leq n$ such that $\binom{m}{k} \geq l$. Then $VCdim(C) \geq kl\lfloor\log_2 \frac{n}{m}\rfloor$.*

Note in particular that if l is 1 then we can take m to be k and if k is 1 then we can take m to be l.

PROOF: Let $r = \lfloor\log_2 \frac{n}{m}\rfloor$. If $r = 0$, then the theorem follows trivially, since C is non-empty. Assume $r > 0$. Let $s = 2^r$. Note that $ms \leq n$. We will construct a set $S \subseteq \{0,1\}^n$ containing klr points that is shattered by C. To describe the construction we will need to refer to an enumeration of the $\binom{m}{k}$ ways to choose k distinct integers from the set $\{1, \ldots, m\}$. Let $\{(\kappa_{j1}, \ldots, \kappa_{jk})\}$, where j runs from 1 through $\binom{m}{k}$, be such an enumeration. (The values of the κ_{ji} are the chosen integers.) We will construct S as the union of sets S_{ji} for $i = 1, \ldots, k$ and $j = 1, \ldots, l$. Each set S_{ji} contains r points and each point has n coordinates. Split these coordinates into groups so that there are m disjoint groups of s coordinates each. There may be some coordinates left over; we will not make them a part of any group. Number the groups from 1 through m. Fix attention on some i and j. Let all coordinates of each point in S_{ji} be 0 except for coordinates in the groups numbered κ_{j1} through κ_{jk}. Let the coordinates in groups κ_{j1} through κ_{jk} be 1 except for those in group κ_{ji}. The coordinates in group κ_{ji} are used to distinguish the points within set S_{ji}. Set the coordinates in this group to 0 or 1 in such a manner that for each subset $V \subseteq S_{ji}$, there is a corresponding coordinate in group κ_{ji} that is 1 at points in V and 0 at points in $S_{ji} - V$. This is possible since there are 2^r subsets of S_{ji} and there are 2^r coordinates in the group. For example, suppose $n = 24$, $k = 2$, and $l = 3$. If we take $m = 3$, then we get $r = 3$. Picking $(1,2), (1,3), (2,3)$ as the enumeration of the three ways to choose two integers from $\{1,2,3\}$, we can take the sets S_{ji} as shown in Table 3.

Now we show how to construct an l-term k-DNF formula that is 1 exactly on some arbitrary subset $U \subseteq S$. Each of the l terms of this formula will have length exactly k, and no literal will be negated. Let $U_{ji} = U \cap S_{ji}$. We will express the formula in terms of n variables, with one variable corresponding to each coordinate. This gives m groups of variables, corresponding to the m groups of coordinates. The jth term will contain one variable from each of the groups $\kappa_{j1}, \ldots, \kappa_{jk}$. We choose the ith variable in the jth term from group κ_{ji} so that it is 1 at all of the points in U_{ji} and 0 at points in $S_{ji} - U_{ji}$. This is possible due to the way the sets S_{ji} were constructed.

To see that this formula is 1 on U and 0 on $S - U$, consider any point $x \in S$. This point will be in S_{ji} for some i and j. The coordinates of the point will be 0 except in groups $\kappa_{j1}, \ldots, \kappa_{jk}$. Thus every term but the

Table 3. An example of the sets S_{ji}.

$$
\begin{aligned}
S_{11} = \{&(0,0,0,0,1,1,1,1,\ 1,1,1,1,1,1,1,1,\ 0,0,0,0,0,0,0,0),\\
&(0,0,1,1,0,0,1,1,\ 1,1,1,1,1,1,1,1,\ 0,0,0,0,0,0,0,0),\\
&(0,1,0,1,0,1,0,1,\ 1,1,1,1,1,1,1,1,\ 0,0,0,0,0,0,0,0)\}\\
S_{12} = \{&(1,1,1,1,1,1,1,1,\ 0,0,0,0,1,1,1,1,\ 0,0,0,0,0,0,0,0),\\
&(1,1,1,1,1,1,1,1,\ 0,0,1,1,0,0,1,1,\ 0,0,0,0,0,0,0,0),\\
&(1,1,1,1,1,1,1,1,\ 0,1,0,1,0,1,0,1,\ 0,0,0,0,0,0,0,0)\}\\
S_{21} = \{&(0,0,0,0,1,1,1,1,\ 0,0,0,0,0,0,0,0,\ 1,1,1,1,1,1,1,1),\\
&(0,0,1,1,0,0,1,1,\ 0,0,0,0,0,0,0,0,\ 1,1,1,1,1,1,1,1),\\
&(0,1,0,1,0,1,0,1,\ 0,0,0,0,0,0,0,0,\ 1,1,1,1,1,1,1,1)\}\\
S_{22} = \{&(1,1,1,1,1,1,1,1,\ 0,0,0,0,0,0,0,0,\ 0,0,0,0,1,1,1,1),\\
&(1,1,1,1,1,1,1,1,\ 0,0,0,0,0,0,0,0,\ 0,0,1,1,0,0,1,1),\\
&(1,1,1,1,1,1,1,1,\ 0,0,0,0,0,0,0,0,\ 0,1,0,1,0,1,0,1)\}\\
S_{31} = \{&(0,0,0,0,0,0,0,0,\ 0,0,0,0,1,1,1,1,\ 1,1,1,1,1,1,1,1),\\
&(0,0,0,0,0,0,0,0,\ 0,0,1,1,0,0,1,1,\ 1,1,1,1,1,1,1,1),\\
&(0,0,0,0,0,0,0,0,\ 0,1,0,1,0,1,0,1,\ 1,1,1,1,1,1,1,1)\}\\
S_{32} = \{&(0,0,0,0,0,0,0,0,\ 1,1,1,1,1,1,1,1,\ 0,0,0,0,1,1,1,1),\\
&(0,0,0,0,0,0,0,0,\ 1,1,1,1,1,1,1,1,\ 0,0,1,1,0,0,1,1),\\
&(0,0,0,0,0,0,0,0,\ 1,1,1,1,1,1,1,1,\ 0,1,0,1,0,1,0,1)\}
\end{aligned}
$$

jth will contain at least one variable that is 0 at x. Therefore the formula will be 1 if and only if the jth term is 1. The coordinates of x will be 1 in groups $\kappa_{j1}, \ldots, \kappa_{jk}$, except possibly in group κ_{ji}. Hence all variables in the jth term will be 1, except possibly for the ith variable. Therefore the value of the formula at x will match the value of the ith variable of the jth term. This variable will be 1 if $x \in U_{ji}$ and 0 if $x \in S_{ji} - U_{ji}$, as desired. ∎

The algorithm can be modified to work on larger classes of Boolean functions. For any instance space $X \subseteq \{0,1\}^n$, and for any δ satisfying $0 < \delta \leq 1$ let $F(X, \delta)$ be the class of functions from X to $\{0,1\}$ with the following property: for each $f \in F(X, \delta)$ there exist $\mu_1, \ldots, \mu_n \geq 0$ such that for all $(x_1, \ldots, x_n) \in X$

$$\sum_{i=1}^{n} \mu_i x_i \geq 1 \quad \text{if } f(x_1, \ldots, x_n) = 1 \tag{1}$$

and

$$\sum_{i=1}^{n} \mu_i x_i \leq 1 - \delta \quad \text{if } f(x_1, \ldots, x_n) = 0. \tag{2}$$

In other words, the inverse images of 0 and 1 are linearly separable with a minimum separation that depends on δ. We will present a second variant of WINNOW that can handle target classes of this form.

Table 4. WINNOW2's response to mistakes.

learner's prediction	*correct response*	*update action*	*update name*
1	0	$w_i := w_i/\alpha$ if $x_i = 1$ w_i unchanged if $x_i = 0$	demotion step
0	1	$w_i := \alpha \cdot w_i$ if $x_i = 1$ w_i unchanged if $x_i = 0$	promotion step

The mistake bound that we derive will be practical only for those linearly-separable functions for which δ is sufficiently large. For example, these include the Boolean r-of-k threshold functions. Let $X = \{0,1\}^n$. An r-of-k threshold function $f(x_1, \ldots, x_n)$ is defined by selecting a set of k significant variables. The value of f is 1 whenever at least r of these k variables are 1. If the k selected variables are $x_{i_1}, \ldots, x_{i_k}$, then f is 1 exactly when $x_{i_1} + \cdots + x_{i_k} \geq r$. Equivalently, f is 1 when

$$\frac{1}{r}x_{i_1} + \cdots + \frac{1}{r}x_{i_k} \geq 1.$$

The value of f is 0 when no more than $r - 1$ of the selected variables are 1. In this case

$$\frac{1}{r}x_{i_1} + \cdots + \frac{1}{r}x_{i_k} \leq 1 - \frac{1}{r}.$$

Thus the r-of-k threshold functions are contained in $F(\{0,1\}^n, \frac{1}{r})$.

There exist other classes of linearly-separable Boolean functions for which $\frac{1}{\delta}$ grows exponentially with n when the instance space is $\{0,1\}^n$ (Muroga, 1971; Hampson & Volper, 1986). One example of a set of functions with exponentially small δ consists of

$$f(x_1, \ldots, x_n) = x_1 \vee (x_2 \wedge (x_3 \vee (x_4 \wedge \cdots x_n)) \cdots)$$

as n varies. For such functions, the mistake bound that we will derive grows exponentially with n. We now give a description of the second variant of WINNOW.

Algorithm 4 (WINNOW2)

The only change to WINNOW1 involves the amount by which the weights are changed when a mistake is made. In a promotion step, as before, we multiply the weights by a fixed $\alpha > 1$. But now, instead of setting weights to zero in an elimination step, we divide them by α. (We now call this a *demotion* step.) We must now be more careful in our choice of α. For the mistake bound that we derive below, we use $\alpha = 1 + \delta/2$ for learning a target function in $F(X, \delta)$.

Table 4 describes WINNOW2's responses to different types of mistakes. Space and time requirements for WINNOW2 are similar to those for WINNOW1. However, more bits will be needed to store each weight, perhaps as many as the logarithm of the mistake bound. The following theorem gives a mistake bound for WINNOW2.

Theorem 9 *For $0 < \delta \leq 1$, if the target function is in the class $F(X, \delta)$ for $X \subseteq \{0,1\}^n$, if $\mu_1, \ldots, \mu_n$ have been chosen so that the target function satisfies the inequalities (1) and (2), and if Algorithm 4 is run with $\alpha = 1 + \frac{\delta}{2}$ and $\theta \geq 1$ and the algorithm receives instances from X, then the number of mistakes will be bounded by*

$$\frac{8}{\delta^2}\frac{n}{\theta} + (\frac{5}{\delta} + \frac{14 \ln \theta}{\delta^2})\sum_{i=1}^{n} \mu_i.$$

Before proving this theorem, we will state and prove three lemmas analogous to the lemmas used to prove Theorem 7. We define u and v in the same manner as for those lemmas. The current lemmas do not depend on the particular choice of α given in Theorem 9.

Lemma 7

$$v \leq \frac{\alpha}{\alpha - 1}\frac{n}{\theta} + \alpha u.$$

PROOF: We will examine how the weights are changed by promotion and demotion steps. We will use $w_{i,\text{bef}}$ to denote weights at the beginning of a trial in which a promotion or demotion occurs, and $w_{i,\text{aft}}$ to denote the weights resulting from the promotion or demotion. For a promotion step, we can write the update rule as

$$w_{i,\text{aft}} = w_{i,\text{bef}} + (\alpha - 1)x_i w_{i,\text{bef}} \qquad \text{for } i = 1, \ldots, n.$$

Since a promotion step only occurs when $\sum_{i=1}^{n} w_{i,\text{bef}} x_i \leq \theta$, we have

$$\sum_{i=1}^{n} w_{i,\text{aft}} \leq \sum_{i=1}^{n} w_{i,\text{bef}} + (\alpha - 1)\theta$$

for a promotion step. For a demotion step, we have

$$w_{i,\text{aft}} = w_{i,\text{bef}} - (1 - \frac{1}{\alpha})x_i w_{i,\text{bef}} \qquad \text{for } i = 1, \ldots, n.$$

A demotion step only occurs when $\sum_{i=1}^{n} w_{i,\text{bef}} x_i > \theta$. Thus

$$\sum_{i=1}^{n} w_{i,\text{aft}} \leq \sum_{i=1}^{n} w_{i,\text{bef}} - (1 - 1/\alpha)\theta.$$

Initially, the sum of the weights is n; hence after u promotions and v demotions,

$$\sum_{i=1}^{n} w_i \leq n + u(\alpha - 1)\theta - v(1 - 1/\alpha)\theta.$$

Since the weights are never negative, we must have $n + u(\alpha - 1)\theta - v(1 - 1/\alpha)\theta \geq 0$. Thus $v(1 - 1/\alpha) \leq \frac{n}{\theta} + u(\alpha - 1)$, giving $v \leq \frac{\alpha}{\alpha - 1}\frac{n}{\theta} + \alpha u$, as desired. ■

Lemma 8 *For all i, $w_i \leq \alpha\theta$.*

PROOF: Since $\theta \geq 1$ and $\alpha > 1$, the weights are initially less than or equal to $\alpha\theta$. For any j, the value of w_j is only increased during a trial in which $x_j = 1$ and $\sum_i^n w_i x_i \leq \theta$. These conditions can only occur together if $w_j \leq \theta$ immediately prior to the promotion. Thus $w_j \leq \alpha\theta$ after the promotion. ■

Lemma 9 *After u promotion steps and v elimination steps, there exists some i for which*

$$\log w_i \geq \frac{u - (1 - \delta)v}{\sum_{i=1}^{n} \mu_i} \log \alpha.$$

PROOF: We will use the symbols $w_{i,\text{bef}}$ and $w_{i,\text{aft}}$ as in the proof of Lemma 7. This time we look at what happens to $\sum_{i=1}^{n} \mu_i \log w_i$. We can write the promotion update rule as

$$w_{i,\text{aft}} = \alpha^{x_i} w_{i,\text{bef}}.$$

Taking the logarithm and multiplying by μ_i, we get

$$\mu_i \log w_{i,\text{aft}} = \mu_i \log w_{i,\text{bef}} + \mu_i x_i \log \alpha.$$

A promotion step only occurs when $\sum_{i=1}^{n} \mu_i x_i \geq 1$. Thus, at a promotion step we have

$$\sum_{i=1}^{n} \mu_i \log w_{i,\text{aft}} \geq \sum_{i=1}^{n} \mu_i \log w_{i,\text{bef}} + \log \alpha.$$

At a demotion step we have

$$w_{i,\text{aft}} = \alpha^{-x_i} w_{i,\text{bef}}.$$

Thus

$$\mu_i \log w_{i,\text{aft}} = \mu_i \log w_{i,\text{bef}} - \mu_i x_i \log \alpha.$$

For a demotion step to occur, we must have $\sum_{i=1}^{n} \mu_i x_i \leq 1 - \delta$. Thus, at a demotion step we have

$$\sum_{i=1}^{n} \mu_i \log w_{i,\text{aft}} \geq \sum_{i=1}^{n} \mu_i \log w_{i,\text{bef}} - (1-\delta) \log \alpha.$$

Initially, $\sum_{i=1}^{n} \mu_i \log w_i = 0$. After u promotion steps and v demotion steps, we have

$$\sum_{i=1}^{n} \mu_i \log w_i \geq u \log \alpha - (1-\delta) v \log \alpha.$$

Since the μ_i are non-negative, we get

$$(\max_{i=1,\ldots,n} \log w_i) \sum_{i=1}^{n} \mu_i \geq [u - (1-\delta)v] \log \alpha,$$

and dividing by $\sum \mu_i$ gives the desired result. ■

PROOF OF THEOREM 9: From lemmas 8 and 9 we get

$$\frac{u - (1-\delta)v}{\sum_{i=1}^{n} \mu_i} \log \alpha \leq \log \alpha + \log \theta.$$

Since $\alpha > 1$ and the μ_i are non-negative, we can rewrite this inequality as

$$u - (1-\delta)v \leq (1 + \frac{\log \theta}{\log \alpha}) \sum_{i=1}^{n} \mu_i.$$

A second inequality involving $u - (1-\delta)v$ results from using Lemma 7 to eliminate v from the expression. This gives us

$$u - (1-\delta)v \geq u - (1-\delta)(\frac{\alpha}{\alpha - 1}\frac{n}{\theta} + \alpha u),$$

and using the value for α given in the theorem, we get

$$u-(1-\delta)v \geq u-(1-\delta)\frac{2+\delta}{\delta}\frac{n}{\theta}-(1-\delta)(1+\frac{\delta}{2})u = (\frac{\delta}{2}+\frac{\delta^2}{2})u-\frac{(1-\delta)(2+\delta)}{\delta}\frac{n}{\theta}$$

Combining the two inequalities involving $u - (1-\delta)v$, we get

$$(\frac{\delta}{2} + \frac{\delta^2}{2})u - \frac{(1-\delta)(2+\delta)}{\delta}\frac{n}{\theta} \le (1 + \frac{\log\theta}{\log(1+\frac{\delta}{2})})\sum_{i=1}^{n}\mu_i,$$

and therefore

$$\frac{\delta}{2}u \le \frac{(1-\delta)(2+\delta)}{\delta}\frac{n}{\theta} + (1 + \frac{\log\theta}{\log(1+\frac{\delta}{2})})\sum_{i=1}^{n}\mu_i.$$

From Taylor's formula with remainder, we get

$$\ln(1+\frac{\delta}{2}) \ge \frac{\delta}{2} - \frac{(\delta/2)^2}{2} = \frac{\delta}{2}(1 - \frac{\delta}{4}),$$

and since $\delta \le 1$ we get

$$\ln(1+\frac{\delta}{2}) \ge 3\delta/8.$$

Thus, since we have assumed that $\theta \ge 1$,

$$\frac{\delta}{2}u \le \frac{(1-\delta)(2+\delta)}{\delta}\frac{n}{\theta} + (1 + \frac{\ln\theta}{3\delta/8})\sum_{i=1}^{n}\mu_i.$$

From Lemma 7, we have a bound on the total number of mistakes:

$$u + v \le \frac{\alpha}{\alpha-1}\frac{n}{\theta} + (\alpha+1)u.$$

Thus

$$u + v \le \frac{2+\delta}{\delta}\frac{n}{\theta} + \frac{4+\delta}{2}\frac{2}{\delta}[\frac{(1-\delta)(2+\delta)}{\delta}\frac{n}{\theta} + (1 + \frac{\ln\theta}{3\delta/8})\sum_{i=1}^{n}\mu_i]$$

$$= (\frac{(2+\delta)\delta + (4+\delta)(1-\delta)(2+\delta)}{\delta^2})\frac{n}{\theta} + \frac{4+\delta}{\delta}(1 + \frac{8\ln\theta}{3\delta})\sum_{i=1}^{n}\mu_i.$$

Using $0 < \delta \le 1$, we can simplify the upper bound to get

$$u + v \le \frac{8}{\delta^2}\frac{n}{\theta} + (\frac{5}{\delta} + \frac{14\ln\theta}{\delta^2})\sum_{i=1}^{n}\mu_i,$$

as desired. ∎

For the earlier example involving r-of-k threshold functions, we have $\delta = \frac{1}{r}$ and $\sum_{i=1}^{n} \mu_i = \frac{k}{r}$. Thus we get a mistake bound for r-of-k threshold functions for $\alpha = 1 + \frac{1}{2r}$ and $\theta = n$ of $8r^2 + 5k + 14kr \ln n$. We do not know lower bounds for this concept class which are comparable to this upper bound. Note that 1-of-k threshold functions are just k-literal monotone disjunctions. Thus if $\alpha = 3/2$, WINNOW2 will learn monotone disjunctions. The mistake bound is similar to the bound for WINNOW1, though with larger constants.

6. Transformations to other target classes

Various transformations are possible that let one apply the above algorithms to other classes of functions. One can think of these transformations as letting one derive a new learning algorithm from an existing one. The transformations that we will describe here take the form of mappings applied to the instances and predictions. If the instance space of the derived algorithm is X_1 and that of the original algorithm is X_2, then the transformations will take the form of functions $T_i : X_1 \rightarrow X_2$ and $T_p : \{0,1\} \rightarrow \{0,1\}$. We will always take T_p to be either the identity or the function that interchanges 0 and 1 (negation); thus T_p will be invertible. When the derived algorithm receives an instance $x \in X_1$, it sends the instance $T_i(x)$ to the original algorithm, which generates the prediction y. The derived algorithm then generates the prediction $T_p(y)$. Finally, to conclude the trial, when a reinforcement is received, the derived algorithm sends it to the original algorithm. (The reinforcement is passed along without transformation since we view it as a message saying "right" or "wrong" rather than as a message containing the value of the correct response).

Suppose we start with an original algorithm A and we want to derive an algorithm to learn some target class C. What we seek is a target class C_0 that can be learned by A and mappings T_i and T_p such that for every $g \in C$, there exists an $f \in C_0$ such that $T_p \circ f \circ T_i = g$. We have the following theorem.

Theorem 10 *Suppose we are given transformation $T_i : X_1 \rightarrow X_2$, invertible transformation $T_p : \{0,1\} \rightarrow \{0,1\}$, an original algorithm A that can accept instances from X_2, and a derived algorithm B constructed from these as described above. Suppose that we wish algorithm B to learn a target function $g : X_1 \rightarrow \{0,1\}$. If $f : X_2 \rightarrow \{0,1\}$ is a function that can be learned by A with a bounded number of mistakes, and if $T_p \circ f \circ T_i = g$, then algorithm B will learn g making at most $M_A(f)$ mistakes.*

PROOF: Let y be the prediction that the derived algorithm B makes in response to some instance x. For algorithm B to make this prediction, algorithm A must have made the prediction ${T_p}^{-1}(y)$ in response to the instance $T_i(x)$. We have $T_p^{-1}(y) = f(T_i(x))$ if and only if $y = g(x)$. Algorithm A is told that it has made a mistake when the derived algorithm makes a mistake. From the above we see that this happens exactly when the response of A to an instance $T_i(x)$ is not equal to $f(T_i(x))$. This can happen at most $M_A(f)$ times. ■

These transformations are similar in effect to the substitutions described by Kearns, Li, Pitts, and Valiant (1987a).

Now we consider some examples of ways that these transformations can be used to extend the classes of functions learnable using WINNOW1 and WINNOW2. For each example, we show that the transformation satisfies the condition given in Theorem 10, namely that for any desired target function g, there exists a function f in a target class that can be learned by WINNOW1 or WINNOW2 and for which $T_p \circ f \circ T_i = g$. Note that in any case in which we use WINNOW1, WINNOW2 could also be used.

Example 4 *Learning arbitrary disjunctions.* This is an example of one way to learn disjunctions that are not necessarily monotone. Arbitrary disjunctions are also special cases of the classes discussed in Examples 6 and 7 below. We will use WINNOW1, but the learner does not send the first instances to WINNOW1. Instead, the learner just responds 1 until the first mistake is made. This will be an extra mistake, not counted in the bound for WINNOW1. Then the learner starts using WINNOW1, using transformations defined as follows. Suppose $(z_1, \ldots, z_n)$ is the first instance on which a mistake is made. Then we let $T_i : \{0,1\}^n \to \{0,1\}^n$ be the function given by

$$T_i(x_1, \ldots, x_n) = (x_1 + z_1, \ldots, x_n + z_n),$$

where the addition is modulo 2. We let T_p be the identity.

To construct the function f of Theorem 10, write the target function g as

$$g(x_1, \ldots, x_n) = x_{i_1} \vee \cdots \vee x_{i_l} \vee \bar{x}_{j_1} \vee \cdots \vee \bar{x}_{j_m}$$

for some l and m. Since $g(z_1, \ldots, z_n) = 0$ we must have $z_{i_1} = \cdots = z_{i_l} = 0$ and $z_{j_1} = \cdots = z_{j_m} = 1$. Let

$$f(x_1, \ldots, x_n) = x_{i_1} \vee \cdots \vee x_{i_l} \vee x_{j_1} \vee \cdots \vee x_{j_m}.$$

Then

$$f \circ T_i(x_1, \ldots, x_n) = x_{i_1} \vee \cdots \vee x_{i_l} \vee (x_{j_1} + 1) \vee \cdots \vee (x_{j_m} + 1) = g(x_1, \ldots, x_n),$$

as desired. The mistake bound for learning non-monotone disjunctions with this method is one more than the corresponding mistake bound for monotone disjunctions.

Example 5 *Learning k-literal monotone conjunctions.* We use WINNOW1. Let $T_i(x_1,\ldots,x_n) = (1-x_1,\ldots,1-x_n)$ and $T_p(r) = 1-r$. If one thinks of 0 and 1 as false and true, then the transformations T_i and T_p just negate all of their arguments. Thus if the target function $g(x_1,\ldots,x_n) = x_{i_1}\cdots x_{i_k}$ (i.e., the conjunction of these k variables) and if we let $f(x_1,\ldots,x_n) = x_{i_1} \vee \cdots \vee x_{i_k}$, then $T_p \circ f \circ T_i = g$ by de Morgan's law. Using WINNOW1 with $\theta = \frac{n}{2}$ and $\alpha = 2$, the number of mistakes will be bounded by $2k\log_2 n + 2$.

Example 6 *Learning linearly-separable Boolean functions with weights that vary in sign.* For $X \subseteq \{0,1\}^n$ and $0 < \delta \leq 1$, let $G(X,\delta)$ be the class of functions $g : X \rightarrow \{0,1\}$ for which there exist $\nu_1,\ldots,\nu_n \geq 0$ and $\tilde{\nu}_1,\ldots,\tilde{\nu}_n \geq 0$ depending on g such that for all $(x_1,\ldots,x_n) \in X$,

$$\sum_{i=1}^{n}(\nu_i x_i + \tilde{\nu}_i(1-x_i)) \geq 1 \quad \text{if } g(x_1,\ldots,x_n) = 1$$

and

$$\sum_{i=1}^{n}(\nu_i x_i + \tilde{\nu}_i(1-x_i)) \leq 1-\delta \quad \text{if } g(x_1,\ldots,x_n) = 0.$$

We will first give a transformation to learn $G(X,\delta)$, and then demonstrate that any linearly-separable Boolean function with domain X is in $G(X,\delta)$ for some δ. To learn functions in $G(X,\delta)$, we use WINNOW2 and the transformation $T_i : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ given by

$$T_i(x_1,\ldots,x_n) = (x_1,x_2,\ldots,x_n,1-x_1,1-x_2,\ldots,1-x_n).$$

We let T_p be the identity. For any function $g \in G(X,\delta)$ we can find a function $f \in F(T_i(X),\delta)$ for which $T_p \circ f \circ T_i = g$, satisfying the condition of Theorem 10. To define f, let $\nu_1,\ldots,\nu_n$ and $\tilde{\nu}_1,\ldots,\tilde{\nu}_n$ be as above. Let $\mu_i = \nu_i$ for $i = 1,\ldots,n$, and let $\mu_i = \tilde{\nu}_{i-n}$ for $i = n+1,\ldots,2n$. Then the function f that is 1 if and only if $\sum_{i=1}^{2n}\mu_i x_i \geq 1$ is the desired function. The mistake bound of Theorem 9 applies, except that n must be replaced with $2n$, and the sum $\sum_{i=1}^{n}\mu_i$ with $\sum_{i=1}^{n}(\nu_i + \tilde{\nu}_i)$.

Now we show that any linearly-separable Boolean function f is in $G(X,\delta)$ for some δ. To see this, first observe that the function that is identically 1 on X is in $G(X,1)$; we can take $\nu_i = \tilde{\nu}_i = 1$ for $i = 1,\ldots,n$. Now take g to be any linearly-separable Boolean function which is not identically 1. We can find $\mu_1,\ldots,\mu_n$, θ and $\theta' < \theta$ such that for all $(x_1,\ldots,x_n) \in X$,

$$\sum_{i=1}^{n}\mu_i x_i \geq \theta \quad \text{if } g(x_1,\ldots,x_n) = 1$$

and

$$\sum_{i=1}^{n} \mu_i x_i \leq \theta' \quad \text{if } g(x_1, \ldots, x_n) = 0.$$

Here we allow the μ_i to vary in sign. Now for each i choose $\mu_i^+, \mu_i^- \geq 0$ such that $\mu_i = \mu_i^+ - \mu_i^-$ and either μ_i^+ or μ_i^- is 0. Then

$$\sum_{i=1}^{n} \mu_i x_i = \sum_{i=1}^{n} (\mu_i^+ x_i + \mu_i^-(1 - x_i)) - \sum_{i=1}^{n} \mu_i^- .$$

Thus

$$\sum_{i=1}^{n} (\mu_i^+ x_i + \mu_i^-(1 - x_i)) \geq \theta + \sum_{i=1}^{n} \mu_i^- \quad \text{if } g(x_1, \ldots, x_n) = 1$$

and

$$\sum_{i=1}^{n} (\mu_i^+ x_i + \mu_i^-(1 - x_i)) \leq \theta' + \sum_{i=1}^{n} \mu_i^- \quad \text{if } g(x_1, \ldots, x_n) = 0.$$

We will next divide each of these inequalities by $\theta + \sum_{i=1}^{n} \mu_i^-$. Note that since g is not identically 1, we have

$$\sum_{i=1}^{n} -\mu_i^- \leq \min_{(x_1, \ldots, x_n) \in X} \sum_{i=1}^{n} \mu_i x_i \leq \theta' < \theta.$$

Hence $\theta' + \sum_{i=1}^{n} \mu_i^- \geq 0$ and $\theta + \sum_{i=1}^{n} \mu_i^- > 0$. We obtain the inequalities

$$\sum_{i=1}^{n} \frac{\mu_i^+}{\theta + \sum_{i=1}^{n} \mu_i^-} x_i + \frac{\mu_i^-}{\theta + \sum_{i=1}^{n} \mu_i^-} (1 - x_i) \geq 1$$

if $g(x_1, \ldots, x_n) = 1$, and

$$\sum_{i=1}^{n} \frac{\mu_i^+}{\theta + \sum_{i=1}^{n} \mu_i^-} x_i + \frac{\mu_i^-}{\theta + \sum_{i=1}^{n} \mu_i^-} (1 - x_i) \leq \frac{\theta' + \sum_{i=1}^{n} \mu_i^-}{\theta + \sum_{i=1}^{n} \mu_i^-}$$

if $g(x_1, \ldots, x_n) = 0$. Thus g is in $G(X, \frac{\theta - \theta'}{\theta + \sum_{i=1}^{n} \mu_i^-})$.

Example 7 *Learning k-DNF for fixed k.* This transformation demonstrates the use of WINNOW1 to learn functions that are not linearly separable. The class k-DNF consists of functions that can be expressed in disjunctive normal form with at most k literals per term. Valiant (1985) and Kearns et al. (1987a) have studied this class. To learn k-DNF, we let $n_2 = \sum_{i=0}^{k} 2^i \binom{n}{i}$. Let

$$T_i(x_1, \ldots, x_{n_1}) = (c_1(x_1, \ldots, x_{n_1}), c_2(x_1, \ldots, x_{n_1}), \ldots, c_{n_2}(x_1, \ldots, x_{n_1}))$$

where the $c_i(x_1, \ldots, x_{n_1})$ range over all conjunctions that form valid terms of a k-DNF formula, i.e., all conjunctions of no more than k literals. We let T_p be the identity. For any k-DNF target function g with l terms, there exist $i_1, \ldots, i_l$ such that $g(x_1, \ldots, x_n)$ is the disjunction of $c_{i_1}(x_1, \ldots, x_n), \ldots, c_{i_l}(x_1, \ldots, x_n)$. Let $f : \{0,1\}^{n_2} \to \{0,1\}$ be defined by $f(y_1, \ldots, y_{n_2}) = y_{i_1} \vee \cdots \vee y_{i_l}$. Then $g = f \circ T_i$ as desired.

One can show that $n_2 \leq (2n)^k + 1$. To WINNOW1, it will appear that the function being learned is an l-literal monotone disjunction. Thus if the target concept has l terms, WINNOW1 will make $O(l \log n^k) = O(kl \log n)$ mistakes. By contrast, the algorithm for learning k-DNF and similar classes presented by Valiant (1984, 1985) can be forced to make $\binom{n}{k} - l$ mistakes, which is roughly n^k mistakes when l is small. Lemma 6 gives a lower bound on the Vapnik-Chervonenkis dimension of the class of l-term k-DNF formulas. This is also a lower bound on the mistake bound. In that lower bound, take $m = \lceil kl^{1/k} \rceil$. We have

$$\binom{m}{k} \geq \frac{m^k}{k^k} \geq l,$$

as required. Thus a lower bound on the mistake bound, in the case that $kl^{1/k} \leq n$, is

$$kl \lfloor \log_2 \frac{n}{\lceil kl^{1/k} \rceil} \rfloor.$$

If we know l and run WINNOW1 with $\alpha = 2$ and $\theta = \frac{n_2}{2l}$, then the number of mistakes made by the derived algorithm will be bounded by

$$2l(1 + \log_2 \frac{(2n)^k + 1}{l}) \leq 2l(2 + \log_2 \frac{(2n)^k}{l}) = 4l + 2kl \log_2 \frac{2n}{l^{1/k}}.$$

For fixed k, this is similar in form to the lower bound.

7. Conclusion

This paper divides into two parts. The first part contains general results about how many mistakes an effective learner might make if computational complexity were not an issue. The second portion describes an efficient algorithm for learning specific target classes. The general results of the first part lead to the inequalities:

$$VCdim(C) \leq opt(C) \leq M_{HALVING}(C) \leq \log_2(|C|)$$

for any non-empty target class C. Examples in Section 3 demonstrate that $VCdim(C)$ can be 1 for classes for which $opt(C)$ is large, and that $M_{HALVING}(C)$ can be 1 for classes for which $\log_2(|C|)$ is large. Example 2 also shows that *opt* and $M_{HALVING}$ can differ. There also exist classes C for which $VCdim(C) = \log_2(|C|)$, making all of the above inequalities into equalities. (This is true of any class that contains exactly those concepts required to shatter some particular finite subset of the domain.)

When we turn to efficient algorithms, we find that WINNOW1, WINNOW2, and their transformations do very well for certain concept classes. These include disjunctions, conjunctions, r-of-k threshold functions, other classes of linearly-separable functions with sufficiently large separation, and some classes of non-linearly-separable functions, such as k-DNF for fixed small k.

The results here contrast with those of Kearns et al. (1987a), who have demonstrated that if $P \neq NP$ and if the learner is required to choose hypotheses from the target class, then r-of-k threshold functions are not polynomially learnable in the Valiant learning model. Using methods that we mentioned in Section 3, WINNOW2 (which learns r-of-k threshold functions) can be converted into a polynomial learning algorithm in the Valiant model. This algorithm succeeds in efficiently learning r-of-k threshold functions by choosing hypotheses from a larger class of Boolean functions. WINNOW1 and WINNOW2 are natural algorithms for parallel implementation; Slade (1987) has implemented WINNOW on the Connection Machine.

A key advantage of WINNOW1 and WINNOW2 is their performance when few attributes are relevant. If we define the number of relevant variables needed to express a function in the class $F(\{0,1\}^n, \delta)$ to be the least number of strictly positive weights needed to describe a separating hyperplane, then the bounds for WINNOW1 and WINNOW2 tell us that the target class $F(\{0,1\}^n, \delta)$ for $n > 1$ can be learned with a number of mistakes bounded by a constant times $\frac{k \log n}{\delta^2}$ when the target function can be expressed with k relevant variables. (This follows from the bound given in Theorem 9 using the observation that, in the inequalities (1) and (2) in the definition of $F(X, \delta)$, any μ_i larger than 1 can be set to 1 without changing the function.)

Note that WINNOW1 (for the target class $F(X, 1)$) and WINNOW2 achieve this bound without necessarily producing a hypothesis expressed with few significant weights. For example, if several attributes match each other in every instance, then their weights will always match, and a hypothesis making significant use of any of them will make use of all.

One theme that recurs in this paper is transformation from one algorithm to another. We have discussed transformations of several kinds, including:

- transformations from algorithms for one target class to algorithms for another target class;
- transformations between mistake-bounded algorithms and query algorithms;
- transformations from mistake-bounded algorithms to algorithms that provably perform well in a probabilistic learning model;
- transformations from arbitrary mistake-bounded algorithms to "normalized" mistake-bounded algorithms (e.g., the transformation to a conservative algorithm).

Transformations can also be used with the hope of improving the behavior of an algorithm for a target class it is already capable of learning. In this regard, notice that monotone conjunctions can be learned by WINNOW2 with or without the use of the transformation described in Example 5. A k-literal monotone conjunction is just a k-of-k threshold function. Using WINNOW2 to learn a k-of-k threshold function, we have a mistake bound of $5k + (8 + 14 \ln n)k^2$, which applies when $\alpha = 1 + \frac{1}{2k}$ and $\theta = n$. If we use the transformation of Example 5, we end up using WINNOW2 to learn a derived 1-of-k threshold function. In this case, the mistake bound is $8 + (5 + 14 \ln n)k$ with $\alpha = \frac{3}{2}$ and $\theta = n$, which is better by a factor of k than the bound without the transformation. It is not clear to what extent this difference is an artifact of our analysis. However, note that to express a 1-of-k threshold function, any set of weights will work, as long as the weight of each relevant variable is above the threshold and the sum of all other weights is below the threshold. There are tighter constraints on weights used to represent a k-of-k threshold function. The sum of all weights, omitting only that of any single relevant variable, must be below the threshold, whereas the weights of the relevant variables must have a sum above the threshold. This suggests that a k-of-k threshold function might indeed be harder for WINNOW2 to learn than a 1-of-k threshold function.

Though we have shown that WINNOW2 is within a constant factor of optimal for some classes of functions, the ratio of the mistake bound to the optimum grows as δ shrinks. The number of linearly-separable Boolean functions of n attributes is at most 2^{n^2} for $n > 1$ (Blumer et al., 1987a; Muroga, 1971). Thus the halving algorithm would make no more than n^2 mistakes learning any such function. The bound for WINNOW2 grows in proportion to $1/\delta^2$, and there exist classes for which $1/\delta$ grows exponentially with n. Are there efficient algorithms that close this gap?

One advantage of WINNOW1 and WINNOW2 is that they perform well for functions with few relevant attributes without needing to know the number of relevant attributes in advance. This is not true with respect to the separation parameter δ, which affects the choice of the multiplier used by WINNOW2. For practical problems, it would be useful to have a version of WINNOW2 that could function without needing to know δ.

We have mentioned that any mistake-bounded algorithm can be transformed into an algorithm that provably performs well in a probabilistic learning model. One can also run a mistake-bounded algorithm without transformation but assume that the instances are chosen randomly, and then examine its behavior in probabilistic terms. It would be interesting to understand the behavior of WINNOW1 and WINNOW2 in such a setting.

Finally, when the input data to the learner contains errors, WINNOW1 is not robust: if a weight is mistakenly set to zero, the mistake will never be undone. WINNOW2 can learn all concept classes learnable by WINNOW1, and it is more robust. We are currently studying its performance when there are errors in the input data.

Acknowledgements

This research was supported by Contract N00014-86-K-0454 from the Office of Naval Research. I would like to acknowledge the inspiration provided by many valuable discussions with David Haussler and Manfred Warmuth, and by the many key questions that they asked. I also benefited from discussions with Sally Floyd, Ron Rivest, Dana Angluin, and Eli Upfal. Dick Karp raised the question of lower bounds for learning monotone disjunctions, and Larry Stockmeyer and Manfred Warmuth subsequently developed ideas relating to these lower bounds. Mike Paterson and Manfred Warmuth suggested improvements to the upper bounds. Ideas leading to complete k-mistake trees were worked out in conjunction with David Haussler.

References

Angluin, D. (1987). Queries and concept learning. *Machine Learning*, *2*, 319–342.

Angluin, D., & Smith, C. H. (1983). Inductive inference: Theory and methods. *Computing Surveys*, *15*, 237–269.

Banerji, R. B. (1985). The logic of learning: A basis for pattern recognition and for improvement of performance. *Advances in Computers*, *24*, 177–216.

Barzdin, J. M., & Freivald, R. V. (1972). On the prediction of general recursive functions. *Soviet Mathematics Doklady*, *13*, 1224–1228.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1987a). *Learnability and the Vapnik-Chervonenkis dimension* (Technical Report USCS-CRL-87-20). Santa Cruz: University of California, Computer Research Laboratory.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1987b). Occam's Razor. *Information Processing Letters*, *24*, 377–380.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis.* New York: John Wiley.

Hampson, S. E., & Volper, D. J. (1986). Linear function neurons: Structure and training. *Biological Cybernetics*, *53*, 203–217.

Haussler, D. (1985). Space efficient learning algorithms. Unpublished manuscript, University of California, Department of Computer and Information Sciences, Santa Cruz.

Haussler, D. (1986). Quantifying the inductive bias in concept learning. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 485–489). Philadelphia, PA: Morgan Kaufmann.

Haussler, D., Littlestone, N., & Warmuth, M. (1987). Predicting 0,1-functions on randomly drawn points. Unpublished manuscript, University of California, Department of Computer and Information Sciences, Santa Cruz.

Kearns, M., Li, M., Pitt, L., & Valiant, L. (1987a). On the learnability of Boolean formulae. *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (pp. 285–295). New York: The Association for Computing Machinery.

Kearns, M., Li, M., Pitt, L., & Valiant, L. G. (1987b). Recent results on Boolean concept learning. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 337–352). Irvine, CA: Morgan Kaufmann.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, *18*, 203–226.

Muroga, S. (1971). *Threshold logic and its applications.* New York: John Wiley.

Nilsson, N. J. (1965). *Learning machines.* New York: McGraw-Hill.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition.* Cambridge, MA: MIT Press.

Slade, S. (1987). *The programmer's guide to the Connection Machine* (Technical Report). New Haven, CT: Yale University, Department of Computer Science.

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

Valiant, L. G. (1985). Learning disjunctions of conjunctions. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 560–566). Los Angeles, CA: Morgan Kaufmann.

Vapnik, V. N. (1982). *Estimation of dependencies based on empirical data.* New York: Springer-Verlag.

Vapnik, V. N., & Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, *16*, 264–280.