Machine Learning, [Volumn Number], 1--?? ([Volumn Year])

Randomly Fallible Teachers: Learning Monotone DNF with an Incomplete Membership Oracle

DANA ANGLUIN ANGLUIN@CS.YALE.EDU
Department of Computer Science, Yale University, P. O. Box 2158, New Haven, CT 06520

DONNA K. SLONIM SLONIM@THEORY.LCS.MIT.EDU
MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139

Editor: Ming Li

Abstract. We introduce a new fault-tolerant model of algorithmic learning using an equivalence oracle and an *incomplete membership oracle*, in which the answers to a random subset of the learner's membership queries may be missing. We demonstrate that, with high probability, it is still possible to learn monotone DNF formulas in polynomial time, provided that the fraction of missing answers is bounded by some constant less than one. Even when half the membership queries are expected to yield no information, our algorithm will exactly identify m-term, n-variable monotone DNF formulas with an expected $O(mn^2)$ queries. The same task has been shown to require exponential time using equivalence queries alone. We extend the algorithm to handle some one-sided errors, and discuss several other possible error models. It is hoped that this work may lead to a better understanding of the power of membership queries and the effects of faulty teachers on query models of concept learning.

Keywords: concept learning, imperfect teachers, monotone DNF formulas, equivalence queries, membership queries with missing information, persistent errors in queries

1. Introduction

Consider the problem of teaching a computer to recognize verbs in English sentences. One approach for the teacher is to present sample sentences, pointing out some verbs as *positive examples* of the target concept and some other words as *negative examples*. From this, the learner might develop a general idea of the target concept. But in some sentences, other words may deceptively appear in verb form. A natural extension allows the learner to ask specific questions of the type, "is *this* word a verb?" In most cases, the teacher will know the answer from context. For example, in the sample sentence "The department stores open at nine," the learner might consider the possibility that "department" is the subject and "stores" is used as a verb, but would be corrected by the teacher. However, there may be instances in which even the teacher is unsure. The sentence "Tom is running back for his school football team" has at least two legitimate interpretations; the word "running" is a verb in one case but not in another. Without more information, the teacher cannot answer the learner's question, "is 'running' a verb?" Can the learner still learn, even when the teacher is sometimes unsure?

This paper answers that question in the affirmative for the class of monotone DNF formulas. We introduce a new fault-tolerant model of algorithmic learning using an equivalence oracle and an *incomplete membership oracle*, in which the answers to some of the learner's membership queries may be unavailable. The advantage of this model is that it imitates the natural fallibility of teachers in most learning systems. Previous work on query models has generally assumed an omniscient teacher that answers all queries with perfect accuracy. Such assumptions are impractical; even well-intentioned teachers are seldom all-knowing. It is important to consider the degree of teacher fallibility that these models can tolerate. This paper demonstrates that efficient learning is possible even when the teacher is unable to answer a constant fraction (less than one) of the questions asked. By defining a measurable tradeoff between membership and equivalence queries, our model yields some insight into the degree of additional information that membership queries provide a learning algorithm.

1.1. Related Work

There has been a good deal of work done on errors in the distribution-free model of learning introduced by Valiant (1984). Results are encouraging for the case of random misclassification errors. In this benign error model, the teacher produces labeled positive or negative examples, where the label for any example is incorrect independently with probability η. Angluin and Laird (1988) show that information-theoretically, as long as η is less than $\frac{1}{2}$, a sequence of labeled examples of length polynomial in $(\frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-2\eta})$ is sufficient for *pac*-learning. In particular, they show that k-CNF formulas are *pac*-learnable in polynomial time with a random noise rate of less than $\frac{1}{2}$.

Other work has focused on the case of malicious misclassification errors in examples. Valiant (1985) poses the question of learning k-CNF formulas despite an adversarial teacher that draws random positive or negative examples, but with error probability β returns an arbitrary response instead of the correctly labeled example. Valiant shows that a small rate of error can be tolerated in this model. Kearns and Li (1988) show that Valiant's error bound is tight; they use an information-theoretic argument to prove that a malicious error rate of at most $O(\epsilon)$ is tolerable when *pac*-learning any distinct concept class $\mathcal{C}$. A number of other papers further explore various models in which the examples themselves or their classifications are corrupted (see Laird, 1987; Shackelford and Volper, 1988; Sloan, 1988, 1989; among others).

Less is known about errors in query models. Sakakibara (1991) proposes a model of noise in queries, which assumes that every time a query is asked there is some independent probability of getting the wrong answer. Sakakibara gives a general technique to repeat a query sufficiently often to increase the confidence in the answer to a very high level, which allows existing algorithms to be used with appropriate modifications.

However, in some practical situations the problems of missing or incorrect information may not be so easy to remedy. For example, when we ask a teacher to classify a given element of the universe as a positive or negative example of the target concept, it may happen that the teacher simply does not know, and will not know no matter how many times we ask the same question. To address this problem, Goldman, Kearns, and Schapire (1990) consider a model of persistent noise in membership queries, which is related to the model we adopt here (see the discussion in Section 6).

1.2. Overview of the Model

Our model relies on the definition of a *minimally adequate teacher* (Angluin, 1987), in which a learner tries to learn a target concept h_* from a known concept class H. In this (error-free) model, the learner is assisted by a teacher that answers two types of queries. A membership query on a given string tells whether or not that string is a positive example of h_*; such a query is answered "yes" or "no". An equivalence query tests a hypothesis h, returning $\emptyset$ if h is equivalent to h_*, and a counterexample x such that $h(x) \neq h_*(x)$ otherwise. In this model, the choice of the counterexample is arbitrary.

We consider a *randomly fallible minimally adequate teacher*. In particular, we define an *incomplete membership oracle* that, for each string in the sample space, performs one flip of a biased coin that lands "heads" with probability p. On any string in the sample space whose coin landed "heads", membership queries are always answered with "I don't know". On all other strings in the sample space the membership oracle always answers correctly. In other words, with probability p, the teacher may be unsure about a given string and will never gain any more information about it. Note that this situation is "benign" in the sense that the algorithm has only to deal with missing information – the information it gets is guaranteed to be correct.

For equivalence queries we assume that the answers remain correct; that is, the answer "$\emptyset$" is returned if and only if the queried element h is equivalent to the target concept h_* and otherwise the answer is a counterexample x such that $h(x) \neq h_*(x)$. This assumption means that exact identification of the target concept is still possible (since an algorithm could simply perform identification by enumeration using equivalence queries.)

However, in this new model we must specify the type of adversary selecting the counterexamples. (This is also an issue in the standard model when randomized learning algorithms are considered, as Maass (1991) has shown.) We assume that the adversary is "on-line." That is, the choice of a counterexample may depend on the target hypothesis and the history of the computation to the point at which the query is asked, including the hypothesis queried, all previous queries and their answers, and any previous coin-flips of the learning algorithm. However, the choice of counterexample may not depend on the answers to membership queries not yet made. This adversary is strong enough to generate the "worst-case" counterexamples used to provide lower bounds for equivalence queries (Angluin, 1990), but it cannot predict the blind spots of the incomplete membership oracle.

1.3. Discussion of the Model

It may at first seem odd to assume that membership queries are flawed while equivalence queries remain correct. However, consider the situation in which a learning algorithm is attempting to predict the classification of a sequence of examples (produced and classified by Nature) with the assistance of a teacher who can correctly classify some but not all of the possible examples (modeled by an incomplete membership oracle.) If we have an efficient learning algorithm using equivalence queries and an incomplete membership oracle, then by a general transformation (Littlestone, 1988) we can obtain an efficient algorithm for the prediction task in the mistake bounded model that uses an incomplete membership oracle.

It may also seem unrealistic to consider a teacher whose failures occur uniformly at random. Despite the weight of precedent, it is natural to look for a more reasonable way of modeling a teacher's limitations. One possibility is for the teacher to be less certain about data points which are "close to the border" of the target concept. While this reasoning works well for certain graphic concepts, such as handwriting recognition or intersections of half-planes, defining a general version may be more challenging. Even if it were clear how to design such a model, working with it is likely to be difficult. Reasoning about randomly flawed teachers has proven considerably more tractable.

1.4. The Importance of Membership Queries

Certain classes of concepts, such as deterministic finite state acceptors and monotone DNF formulas, have been shown to be learnable in polynomial time using equivalence and membership queries, but not using equivalence queries alone (Valiant, 1984; Angluin, 1987, 1988, 1990). A natural question is to investigate which of these concept classes remain learnable in polynomial time in the appropriate sense in this new model, with an incomplete membership oracle. We may then derive some measure of the "importance" of membership queries to the learning algorithm as we vary the failure probability p from 0 (complete information from membership queries) to 1 (no information from membership queries.)

In a recent paper, Angluin and Kharitonov (1991) explore a number of cryptographic limitations on the power of membership queries. They show that, assuming the intractability of either testing quadratic residues modulo a composite, inverting RSA encryption, or factoring Blum integers, there is no polynomial time prediction algorithm using membership queries for several important concept classes, including the classes of Boolean formulas, 3μ-formulas, NFA's, and CFG's. They also show that if one-way functions exist, then membership queries provide no additional power in learning general CNF or DNF. That is, either these classes are learnable without membership queries, or they are not learnable even with membership queries. Their work is especially relevant in light of the results of this paper, showing that membership queries do provide additional power to some learning algorithms, even when only a fraction of the queries are answered. It would be interesting to find a general characterization of the concept classes for which this is the case.

Monotone concept classes are particularly promising for this new model, since there is the hope of reconstructing missing information from responses to additional queries. In this paper we examine the learnability of monotone DNF formulas over n variables. There is a known algorithm for exactly learning these formulas from a minimally adequate teacher in time polynomial in n and m, where m is the number of terms in the target formula (Valiant, 1984; Angluin, 1988). We present an algorithm for the new model that, for any failure probability $p < 1$, produces with probability at least $1 - e^{-s}$ a hypothesis equivalent to the target concept in time polynomial in n, m and s. The running time of this algorithm is not dominated by the failure probability for moderate p; when $p \leq \frac{1}{2}$, the expected total number of queries is $O(mn^2)$.

We observe that when p is nonzero, there is a nonzero probability that the algorithm will obtain *no* information from any of its membership queries. However, there is no algorithm that runs in time polynomial in n and m and exactly identifies any monotone DNF formula using equivalence queries only (Angluin, 1990). Thus, the quantification of "with high probability" is necessary in the statement of our main result.

2. Preliminaries

The target concepts are monotone DNF formulas over the variables $x_1, \ldots, x_n$ for some positive integer n. For example, for $n = 20$,

$$x_1x_4 + x_2x_{17}x_3 + x_9x_5x_{12}x_3 + x_8$$

is a possible target concept. The number of terms in the target formula will be denoted by m; in this example $m = 4$. Note that there is an efficient algorithm to minimize the number of terms of a monotone DNF formula. We shall assume that the target formula h_* has been minimized.

The *sample space* of examples is the set of all possible vectors of n $\mathbf{0}$'s and $\mathbf{1}$'s; that is, the set $\{\mathbf{0}, \mathbf{1}\}^n$. A monotone DNF formula h is interpreted as denoting the set of vectors from the sample space that satisfy h. If vector v satisfies formula h we write $h(v) = 1$; otherwise $h(v) = 0$. We view the sample space as a lattice, with componentwise "or" and "and" as the lattice operations. The top element is the vector of all $\mathbf{1}$'s, and the bottom element is the vector of all $\mathbf{0}$'s. The elements are partially ordered by $\leq$, where $v \leq w$ if and only if $v[i] \leq w[i]$ for all $1 \leq i \leq n$.

For convenience, we introduce an alternative representation of monotone DNF formulas in which each term is represented by the minimum vector in the ordering $\leq$ that satisfies the term. Thus, the vector $\mathbf{10011}$ denotes the term $x_1x_4x_5$. In this representation, if h is a monotone DNF formula and v is a vector in the sample space, v satisfies h if and only if for some term w of h, $w \leq v$. Figure 1 shows the four-variable lattice, with the enclosed area containing all vectors satisfying the formula $x_1x_4 + x_1x_2 + x_3$.

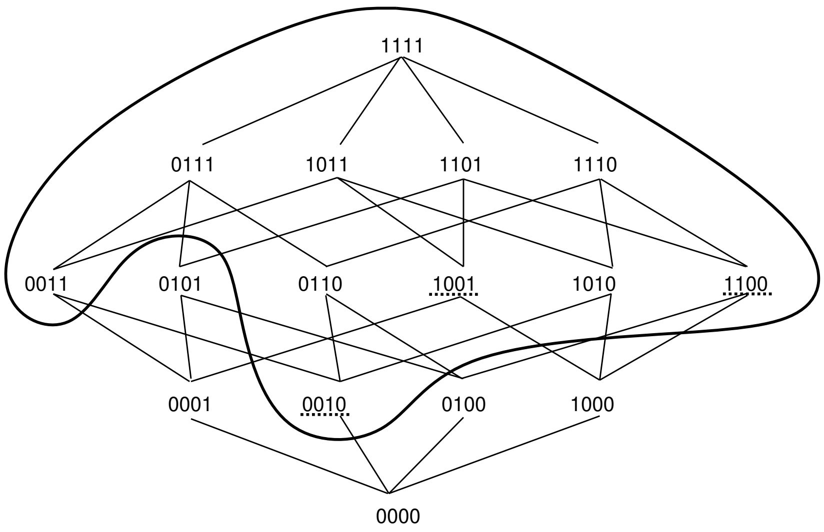

Figure 1. The target concept $x_1x_4 + x_1x_2 + x_3$.

In this representation, since we have assumed that the target concept h_* is minimized, the terms of h_* are precisely the minimal positive examples of h_*, also called *minterms.* Similarly, we define a *maxterm* t_m of the formula to be a maximal negative example of h_*. That is, $h_*(t_m) = 0$, but if any variable not in t_m is added, the resulting vector will force the target formula to 1.

The *descendants* of a vector v are all the vectors w in the sample space such that $w \leq v$. For any nonnegative constant d, the *d-descendants* of v are all the descendants w of v that can be obtained from v by replacing at most d **1**'s by **0**'s. For example, consider the term $x_2x_3x_4$, represented as **0111**. Its set of 1-descendants, $\{\mathbf{0111}, \mathbf{0011}, \mathbf{0101}, \mathbf{0110}\}$, contains the term itself and its three children. The set of 2-descendants is the union of the set of 1-descendants with **0111**'s grandchildren: **0001**, **0010**, and **0100**. Figure 2 shows the set of 2-descendants of the vector **0111**.

3. Using Incomplete Membership Queries

A key subprocedure in the monotone DNF algorithm of Angluin (1988) takes a positive example v of the target concept h_* and uses membership queries to reduce v to a minimum positive example of h_*. The algorithm starts with the empty

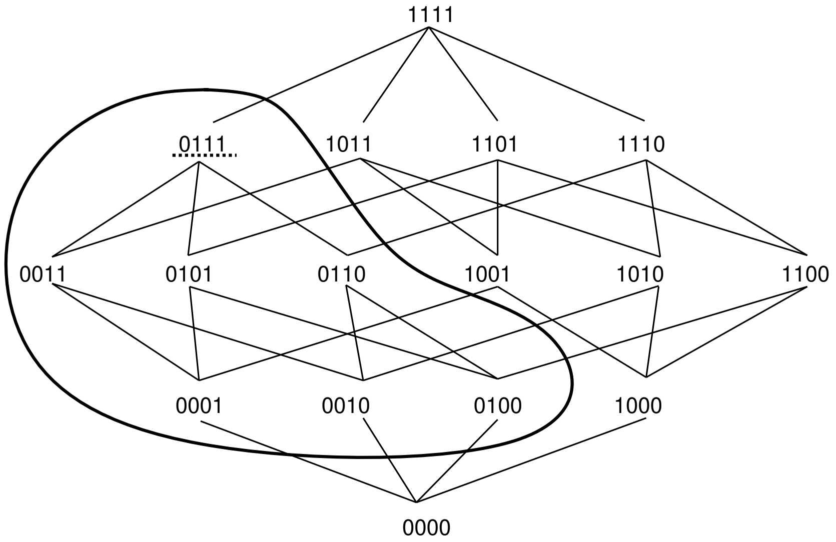

Figure 2. The 2-descendants of $x_2x_3x_4$.

formula and uses equivalence queries to generate new positive counterexamples to reduce. The result of each reduction is a new term of h_*, which is added to the current hypothesis. After m iterations of this process the current hypothesis is equivalent to h_*.

Our new algorithm is based on the same idea, but must use an incomplete membership oracle. The difficulty that arises is as follows. The reduction process has a current positive example v of h_* and makes membership queries for each of the children of v. As long as at least one of these membership queries is answered "yes," say for the vector y, then v can be replaced by y and the process repeated. However, eventually the process arrives at a positive example v of h_* such that membership queries for all of the children of v are answered either "no" or "I don't know." If there is at least one child of v answered "I don't know," then v may or may not be a minimum positive example of h_*.

We therefore propose and analyze a new reduction process to be used with an incomplete membership oracle. The goal is to take an initial positive example v of h_* and reduce it to a positive example that is "likely" to be "not too far above" a minimum positive example of h_*. By adding all sufficiently close descendants of this vector as terms to the current hypothesis, we will be "likely" to add a new term of h_*, and therefore, to make progress towards exact identification of h_*. In so doing, we may add terms to the current hypothesis that do not imply h_*, but these will eventually be removed in response to negative counterexamples.

The idea of the new reduction process is to use membership queries to search not only the children of v but also all the "close enough" descendants of v, looking for a vector $y \leq v$ that is answered "yes." If such a y is found, then the search continues with y in place of v. If no such y is found after querying all the "close enough" descendants of v, then v is returned. Note that the descendants are searched in breadth-first order – first the children of v, then the grandchildren, etc. The parameter $d \geq 1$ specifies the depth of search from v.

```
Reduce(v, d):
        Let D be the proper d-descendants of v
        For each y ∈ D in breadth-first order
                If membership-query(y) = "yes" then
                        Return Reduce(y, d)
                Endif
        Endfor
        Return v
End.
```

Suppose **Reduce** is called with an incomplete membership oracle for h_*, and inputs v (a positive example of h_*) and $d \geq 1$. It is clear that **Reduce** must eventually return a vector v' such that $v' \leq v$, v' is a positive example of h_*, and membership queries for all the proper d-descendants of v' were answered either "no" or "I don't know." We analyze the probability that there is NO minimum positive example of h_* among the d-descendants of v'.

We begin with a couple of simple observations. If w is a positive example of h_* and d is a positive integer, let $D_+(w, d)$ denote the number of proper d-descendants of w that are positive examples of h_*. Also, let $S(w, d)$ be the "success" indicator; that is, $S(w, d) = 1$ if w has a minimum positive example of h_* among its d-descendants, and $S(w, d) = 0$ otherwise.

Lemma 1 *Suppose w is a positive example of h_* and d is a positive integer such that $S(w, d) = 0$. Then $D_+(w, d) \geq 2^{d+1} - 2$.*

Proof: Since w is a positive example of h_*, there is some minimum positive example w' of h_* such that $w' \leq w$. Since by assumption w' is not among the d-descendants of w, there must be some set of $d + 1$ bit positions that contain a **1** in w and a **0** in w'. Every vector w'' obtained by taking w and setting at least one and not more than d of these bit positions to **0** is a proper d-descendant of w that is a positive example of h_* (because $w' < w''$). Since there are exactly $2^{d+1} - 2$ distinct such vectors w'', the result follows. ■

Lemma 2 *Suppose w and w' are positive examples of h_* and $w > w'$. Then for each positive integer d, $D_+(w, d) > D_+(w', d)$.*

Proof: Let z be the vector that is the bitwise "exclusive or" of w and w'; that is, z has **1**'s in precisely those positions where w has a **1** and w' has a **0**. Then for each vector y that is a proper d-descendant of w' and a positive example of h_*, the bitwise "or" of z and y is a proper d-descendant of w that is a positive example of h_*, and these are all distinct, so $D_+(w,d) \geq D_+(w',d)$.

To see that the inequality is strict, note that z must contain at least one **1**, say at position i, and the vector w'' obtained from w by setting the bit at position i to **0** is a proper d-descendant of w (since $d \geq 1$) that is a positive example of h_* (since $w'' \geq w'$) and is distinct from all the vectors obtained by bitwise "or" with z (since z has a **1** in position i.) It follows that $D_+(w,d) > D_+(w',d)$. ■

Now we proceed to the main lemma for **Reduce**. In order to be able to apply the lemma to successive calls to **Reduce** during a single run, we assume that some membership queries have already been made to the incomplete membership oracle (thus committing the oracle to the answers already given.) However, because of the assumptions of our model, when a previously unqueried element of the domain is queried for the first time, the determination of whether the answer will be "I don't know" may be assumed to be made by an independent biased coin toss at that point.

Lemma 3 *Let h_* be a monotone DNF formula. Assume we have an incomplete membership oracle for h_* with failure probability p, and some queries have already been made to the oracle. Assume that v is a positive example of h_* such that no descendant of v that is a positive example of h_* has yet been the subject of a membership query to the oracle. Let d be a positive integer, and let $g(d) = 2^{d+1} - 2$. Suppose $D_+(v,d) = r$ and $r \geq g(d)$. Let* **Reduce** *be called with arguments v and d and let y denote the vector returned. Then the probability that there is NO minimum positive example of h_* among the d-descendants of y (that is, the probability that $S(y,d) = 0$) is bounded above by*

$$p^{g(d)} + p^{g(d)+1} + \ldots + p^r.$$

Proof: We proceed by induction on increasing values of $D_+(v,d)$, starting with $g(d)$. Note that Lemma 2 guarantees that each recursive call of **Reduce** will have a strictly smaller value of $D_+(v,d)$ than the parent call. Moreover, Lemma 1 implies that if the value of $D_+(v,d)$ ever drops below $g(d)$, then **Reduce** must return a vector y such that $S(y,d) = 1$.

Suppose then that in the top level call of **Reduce**, $D_+(v,d) = g(d)$. In order to return a vector y such that $S(y,d) = 0$, **Reduce** must not be called recursively. In other words, the initial call to **Reduce** makes a membership query for each proper d-descendant of v, and each one is answered either "no" or "I don't know." In particular, all the queries to positive examples of h_* must be answered "I don't know."

Since there are $g(d)$ positive examples of h_* among the proper d-descendants of v, and since by hypothesis each of them has not been previously queried, the event that they are all answered "I don't know" has probability $p^{g(d)}$. Thus, in this case the probability of returning the vector y equal to v, so that $S(y,d)=0$, is bounded by $p^{g(d)}$, establishing the base case of the lemma.

Now suppose that for any vector v such that $g(d) \le D_+(v,d) \le r-1$, the lemma holds. Suppose that **Reduce** is called with arguments v and d, with $D_+(v,d)=r$. There are two cases: either a recursive call is made to **Reduce**, or not.

If not, all the proper d-descendants of v are queried, and all of them are answered either "no" or "I don't know." There are r positive examples of h_* among the elements queried and they must all be answered "I don't know." Since none of these has been queried previously, the probability of this outcome is p^r.

Otherwise, when the first proper d-descendant of v in breadth-first order is found whose query is answered "yes" a recursive call is made to **Reduce** with that vector, say v', and the same d. Note that because the search is done in breadth-first order, no proper descendant of v' that is a positive example of h_* has yet been queried. Moreover, since v' is a proper descendant of v, $D_+(v',d) < D_+(v,d)$ (by Lemma 2), so $D_+(v',d) \le r-1$. If $D_+(v',d) < g(d)$ then the recursive call, and also the top-level call, to **Reduce** cannot return a vector y such that $S(y,d)=0$. Otherwise, $D_+(v',d) \ge g(d)$ and the inductive hypothesis is satisfied, so the probability that the recursive call of **Reduce** with v' and d returns a vector y such that $S(y,d)=0$ is bounded by

$$p^{g(d)} + p^{g(d)+1} + \ldots + p^{r-1}.$$

Thus, summing the bounds on the probabilities of failure at the top level and in the recursive call, the probability that the call of **Reduce** on v and d returns a vector y such that $S(y,d)=0$ is bounded above by

$$p^{g(d)} + p^{g(d)+1} + \ldots + p^{r-1} + p^r.$$

This completes the induction. ■

Note that for $0 < p < 1$, the probability of failure is strictly bounded by $p^{2^{d+1}-2}/(1-p)$. Lemma 4 characterizes the values of d that guarantee that this quantity is at most $\frac{1}{2}$. In the rest of this paper, ln will be used to denote $\log_e$ and log will denote $\log_2$.

Lemma 4 *Let $0 < p < 1$. Then $p^{2^{d+1}-2}/(1-p) \le \frac{1}{2}$ if and only if*

$$d \ge \log(2 + \frac{1+\log(1/(1-p))}{\log(1/p)}) - 1.$$

In particular, if $\epsilon = (1-p)$, it suffices to choose

$$d \ge \lceil \log(\frac{1}{\epsilon}) + \log\log(\frac{1}{\epsilon}) \rceil.$$

Proof: The first assertion is easily verified. For the second, we argue by cases. Note that $p^{2^{d+1}-2}/(1-p)$ is increasing in p and decreasing in d. By calculation, $d = 1$ suffices for all $p \leq .5$ and $d = 2$ suffices for all $p \leq .72$.

Assume $.72 < p < 1$, let $\epsilon = (1-p)$ and assume

$$d \geq \lceil \log(\frac{1}{\epsilon}) + \log\log(\frac{1}{\epsilon}) \rceil. \tag{1}$$

Note that

$$p^{2^{d+1}-2} = (1-\epsilon)^{2^{d+1}-2} \leq e^{-\epsilon(2^{d+1}-2)}, \tag{2}$$

so it suffices if

$$e^{\epsilon(2^{d+1}-2)} \geq \frac{2}{\epsilon}. \tag{3}$$

This is equivalent to

$$d \geq \log(\frac{1}{\epsilon}\ln\frac{2}{\epsilon} + 2) - 1. \tag{4}$$

Thus, if we can show

$$\log(\frac{1}{\epsilon}\log\frac{1}{\epsilon}) \geq \log(\frac{1}{\epsilon}\ln\frac{2}{\epsilon} + 2) - 1, \tag{5}$$

the proof will be completed. By exponentiating both sides and rearranging, we see that it is sufficient to show

$$\frac{2}{\epsilon}\log\frac{1}{\epsilon} \geq \frac{1}{\epsilon}\ln\frac{2}{\epsilon} + 2, \tag{6}$$

which is equivalent to

$$(\frac{2}{\ln 2} - 1)\ln\frac{1}{\epsilon} \geq \ln 2 + 2\epsilon. \tag{7}$$

As ϵ decreases, the lefthand side increases and the righthand side decreases, so it suffices to check the inequality for $\epsilon = .28$, where the lefthand side is 2.40^{+} and the righthand side is 1.26^{-}. ■

4. The Monotone DNF Learning Algorithm

Now we are in a position to describe and analyze the learning algorithm. The algorithm begins with the empty hypothesis (false on all vectors) and makes equivalence queries and processes counterexamples one at a time until the hypothesis is equivalent to the target function. Because it tries to "guess" some of the terms in the target concept, the algorithm may introduce terms that do not actually imply the target concept. Thus a counterexample v may be a negative example of h_*, in which case $h_*(v) = 0$ and $h(v) = 1$. The algorithm processes such a v by removing all the terms from the current hypothesis that are satisfied by the vector v.

Otherwise, a counterexample v is a positive example of h_*. Intuitively, what we'd like to do is to call **Reduce** on v to obtain a vector y that with probability at least $1/2$ has a minimum positive example of h_* among its d-descendants for some moderate value of d. Then we would add all the d-descendants of y as terms to the current hypothesis h and continue. If this worked for each positive example, we'd expect to add a new term of h_* to h for every two positive counterexamples, which would be sufficient progress.

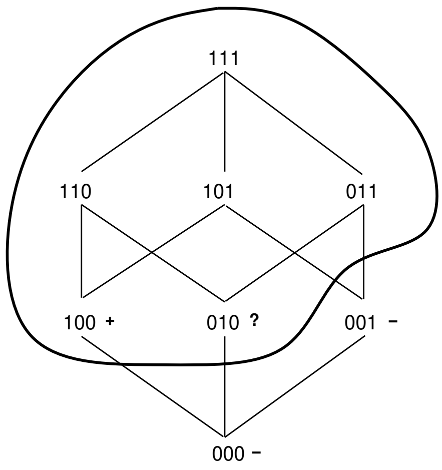

Figure 3. The $+, -$, and ? symbols respectively indicate positive, negative, and "I don't know" answers to membership queries. The first positive counterexample is **110**; vectors **010**, **100**, and **000** are queried. If the second counterexample is **011**, vector **010** would be queried again.

The difficulty is that in order to use Lemma 3, we must guarantee that at each call to **Reduce** with argument v, all the descendants of v that are positive examples of h_* have not been previously queried. This is certainly true the first time **Reduce** is called, but may not be true on subsequent calls. For example, consider the target concept $h_* = x_1 + x_2$ over three variables, as shown in Figure 3, and assume that the initial (positive) counterexample is **110**. When called with $(\mathbf{110}, 1)$ as its arguments, **Reduce** performs membership queries for the children **010** and **100**. Suppose the first query is answered with "I don't know" and the second with "yes." Then the process is iterated with **100**, and the child **000** is queried. Suppose the answer in this case is "no." The hypothesis is then set to x_1 and an equivalence query is made. Assume now the counterexample is **011**. Then **Reduce** is called with arguments (**011**,1) and makes membership queries for the children **001** and **010**. However, **010** is a descendant of the argument that is a positive example h_*, and it has already been queried and answered with "I don't know." Thus we can no longer claim that the probability a membership query on **010** returns "I don't know" is just p.

Our solution for this difficulty is to add to the current hypothesis as terms ALL the vectors queried by **Reduce** that result in an answer of either "yes" or "I don't know." This will guarantee that when a positive counterexample v to the current hypothesis is generated, none of the descendants of v that are positive examples of h_* have been previously queried, since otherwise they would still be present as terms in the current hypothesis, contradicting the fact that v is a counterexample to the current hypothesis. In fact, a vector u ($u \neq y$) queried by **Reduce** that receives the answer "yes" must be a direct ancestor of the returned vector y, which is the *last* vector whose query by **Reduce** answers "yes". A positive counterexample to a hypothesis containing y would be a counterexample to any such vector u as well. So it is sufficient to add to the hypothesis just the vector y and all those vectors queried by **Reduce** that result in "I don't know" answers.

Thus we assume that **Reduce** has been modified to return two results: the vector y previously returned, and the set Q of all the vectors queried by the top-level and all the recursive calls to **Reduce** that resulted in answers of "I don't know." Note that Q must include all the proper d-descendants of y that are positive examples of h_*, since they must all have been queried when **Reduce** returns, and they must all have been answered "I don't know." The modified version of **Reduce** will be called **Reduce1**, and will have an additional parameter Q, which should initially be the empty set.

Reduce1(v, d, Q):
    Let D be the proper d-descendants of v
    For each $y \in D$ in breadth-first order
        If *membership-query*(y) = "I don't know" then
            $Q = Q \cup \{y\}$
        Endif
        If *membership-query*(y) = "yes" then
            Return **Reduce1**(y, d, Q)
        Endif
    Endfor
    Return v, Q
End.

Lemma 3 remains true with **Reduce1** in place of **Reduce**. We can now describe our learning algorithm for monotone DNF formulas using an equivalence oracle and an incomplete membership oracle. The choice of constant d depends on p and will be discussed in the analysis of the algorithm. Recall that we denote a term by the minimum vector satisfying it, so that the current hypothesis h may be thought of as a set of vectors.

```
Learn-mDNF:
    Let h be the empty formula
    While (v = equivalence-query(h)) is not ∅ do
        If (h(v) = 1) then
            Remove from h all w with w ≤ v
        Else
            Call Reduce1(v, d, ∅)
            Let y and Q be the values returned
            Add y and all elements of Q to h
        Endif
    Endwhile
    Output h and halt
End.
```

For the analysis of this algorithm, we let

$$f(p) = \log(2 + \frac{1 + \log(1/(1-p))}{\log(1/p)}) - 1.$$

Recall that Lemma 4 guarantees that for $0 < p < 1$ and $d \geq f(p)$,

$$p^{2^{d+1}-2}/(1-p) \leq \frac{1}{2}.$$

Theorem 5 *Let $0 < p < 1$ be fixed. There is a polynomial $q(n, m, s)$ with the following properties. Let h_* be any monotone DNF formula over n variables with m terms. Suppose* **Learn-mDNF** *is run with an equivalence oracle for h_* and an incomplete membership oracle for h_* with failure probability p, with d set to $\lceil f(p) \rceil$. Then with probability at least $1 - e^{-s}$ the algorithm* **Learn-mDNF** *halts within $q(n, m, s)$ steps and outputs a hypothesis h equivalent to h_*.*

Proof: Assuming **Learn-mDNF** halts, its output h is equivalent to h_*, by the correctness of the equivalence oracle for h_*. Hence we need only prove that it halts within $q(n, m, s)$ steps with probability at least $1 - e^{-s}$.

Suppose at some point the current hypothesis h contains all the terms of h_*. Since no term of h_* will ever be deleted from h, there will be no further positive counterexamples. Negative counterexamples will eventually cause the removal of all the terms of h that do not imply h_*, at which point h will be equivalent to h_* and the algorithm will halt.

All the terms deleted from h in response to negative counterexamples must first be added to h as a consequence of positive counterexamples. The time to process either type of counterexample is bounded by a polynomial in n and the number of terms in the current hypothesis. Therefore it suffices to analyze the total number of terms added to the current hypothesis until it contains all the terms of h_*.

For each positive counterexample processed, **Reduce1** will query $O(n^{d+1})$ elements, so the cardinality of the set Q returned is $O(n^{d+1})$. Hence, each positive counterexample will cause $O(n^{d+1})$ terms to be added to the current hypothesis.

Our goal is now to bound the number of positive counterexamples processed until h contains all the terms of h_*. Let h be the current hypothesis when an equivalence query is made that returns the positive counterexample v. Then, in particular, $h(v) = 0$. Let the result of calling **Reduce1** on input v be the vector y and the set Q. When some d-descendant of y is a minimum positive example of h_*, (that is, $S(y, d) = 1$) we define this call of **Reduce1** to be "successful."

When the call is successful, one of the d-descendants of y, say y', is a minimum positive example of h_*, and therefore a term of h_*. Either $y' = y$ or $y' \in Q$, so y' will be added to h. Note that y' is not currently a term of h, since $y' \leq v$ and $h(v) = 0$. Thus, when a call to **Reduce1** is successful, it adds a new term of h_* to h. There can be at most m successful calls of **Reduce1** before h contains all the terms of h_*.

Now we claim that every call of **Reduce1** is successful with probability at least $\frac{1}{2}$ for the choice of $d = \lceil f(p) \rceil$. To see this, we first argue that each time an equivalence query is made with the current hypothesis h, every positive example of h_* that has previously been the subject of a membership query is a positive example of h. The only place membership queries are made is in the procedure **Reduce1**. When a positive example w of h_* is the subject of a membership query, the incomplete membership oracle must answer either "yes" or "I don't know." In the latter case, w is added to the set Q and returned by **Reduce1**, and then added as a term to h. Since w is a positive example of h_*, the term w implies h_*, so w is not subsequently removed from h. In the former case, either w, or some $u < w$ for which the membership oracle also answered "yes," is added to h. Again, that term implies h_*, so it will never be removed from h. Therefore, whenever an equivalence query is made with the current hypothesis h, every w that is a positive example of h_* that has previously been the subject of a membership query also satisfies h.

Now consider a positive counterexample v returned by the equivalence query. Since v is a counterexample, $h(v) = 0$, which means that for every w that satisfies h, $w \not\leq v$. If w' is any positive example of h_* that is a descendant of v, then w' is not a positive example of h and therefore must not have been previously the subject of a membership query. Thus, the conditions of Lemma 3 are satisfied, and Lemma 4 implies that for our choice of d, the probability that the call of **Reduce1** with argument v is not successful is at most $\frac{1}{2}$. Therefore, each call of **Reduce1** succeeds with probability at least $\frac{1}{2}$, as claimed.

Thus, the expected number of calls to **Reduce1** until h contains all the terms of h_* is at most $2m$. Therefore, the total expected number of terms added to h is $O(mn^{d+1})$.

Moreover, the probability that more than $2m(s+1)$ calls to **Reduce1** will be required before h contains all the terms of h_* is bounded by e^{-s}. To see this, we may apply Karp's method of probabilistic recurrence relations. In (Karp, 1991), recurrence relations of the form $T(x) = a(x) + T(r(x))$ are used to analyze the running time of recursive randomized algorithms. In this relation, $a(x)$ is a nonnegative real function of x corresponding to the amount of effort expended on the original problem, and $r(x)$ is a random variable in the range $[0, x]$ corresponding to the size of the next subproblem to solve recursively. Let $\mu(x)$ be an upper bound on the expectation of random variable $r(x)$, and let $u(x)$ be the least nonnegative solution to the deterministic equivalence relation $\tau(x) = a(x) + \tau(\mu(x))$. Then the following theorem (Theorem 1 in (Karp, 1991)) is useful in showing the bound stated above:

Theorem 6 (Karp) *Suppose there is a constant d such that $a(x) = 0,\ x < d$ and $a(x) = 1,\ x \geq d$. Let $c_t = \min\{x \mid u(x) \geq t\}$. Then, for every positive real x and every positive integer w,*

$$\Pr[T(x) \geq u(x) + w] \leq \left(\frac{\mu(x)}{x}\right)^{w-1} \frac{\mu(x)}{c_{u(x)}}.$$

To apply this to our problem, let x represent the number of terms of h_* that are not yet in h, and let $T(x)$ represent the number of calls made to **Reduce1**. Then $a(x) = 1$ for $x \geq 1$ and $a(x) = 0$ for $x < 1$, since a call will be made to **Reduce1** just in case any terms remain to be found. Since each call succeeds (and reduces x by at least 1) with probability at least $\frac{1}{2}$, we take $\mu(x) = x - \frac{1}{2}$ and $u(x) = 2x$. Applying Theorem 6,

$$Pr\{T(m) \geq 2m + w\} \leq \left(1 - \frac{1}{2m}\right)^{w} \leq e^{-\frac{w}{2m}},$$

which implies that the probability that more than $2m(s+1)$ calls to **Reduce1** will be needed is at most e^{-s}.

Thus, with probability at least $(1 - e^{-s})$ a total of at most $O(smn^{d+1})$ terms are added to h before it contains all the terms of h_*. Translating this bound on the total number of terms added to h into a running time, we may obtain a polynomial $q(n, m, s)$ such that with probability at least $(1 - e^{-s})$ the learning algorithm **Learn-mDNF** halts within $q(n, m, s)$ steps. ■

Recall that for $p = \frac{1}{2}$, $f(p) = 1$, so $d = 1$ suffices. In this case, the expected total number of terms added to h is $O(mn^2)$. When $p = 1 - \epsilon$ for small ϵ, the expected total number of terms added to h is $O(mn^{O(\log(\frac{1}{\epsilon}))})$.

Note that as we have described it, there is a different algorithm for each failure probability p, since the search depth d depends on p. It is clear that an upper bound on p gives an upper bound on a sufficiently large d, so the precise value of p need not be known. It remains to be seen whether the usual methods of sampling to approximate p or iteratively approximating p from below can be adapted to this new setting.

5. Handling Some Errors

One important question is what happens when membership queries may be answered incorrectly. In the case of missing information, at least the learner can rely on the correctness of the information that is obtained. With possibly incorrect answers, the learning problem seems to become harder. One natural extension of the current model is to permit the membership oracle to give one of a variety of "corrupted" answers for those strings whose coin flip results in "heads," while requiring correct answers on the remaining strings.

For one variant, namely $1 \rightarrow 0$ one-sided errors, a minor modification to **Learn-mDNF** permits it to cope with such errors. In this model, for each string in the sample space whose coin flip results in "heads," the answer of the membership oracle is always "no." The modification to **Learn-mDNF** is to add a term to h for EVERY vector queried with membership queries, since now an answer of "no" may be given for a positive example. The analysis of positive examples answered "no" is then the same as the previous analysis of positive examples answered "I don't know." The key observation is that queries answered "yes" are answered correctly; in fact, this modification copes with a model in which corrupted examples may be answered either "no" or "I don't know" arbitrarily.

The dual problem, that of $0 \rightarrow 1$ one-sided errors, is more difficult. A trivial modification of **Learn-mDNF** is no longer sufficient, because the reduction procedure will not terminate until queries on all descendants of some vector return "no". One might assume that if $1 \rightarrow 0$ errors are easily handled by an algorithm that finds minterms of the target formula, then a similar bottom-up approach could be applied to handle $0 \rightarrow 1$ errors and find all maxterms of the target formula. However, for a monotone DNF formula with m minterms over n variables, there may be up to n^m maxterms! Thus, this approach is impractical as well.

6. Future Directions

There are a number of additional questions in this area that could be investigated. Theorem 5 relies on the assumption that all the "I don't know"s are distributed independently at random. It would be informative to find useful results for a more natural model, as discussed in section 1.3.

Another important area is to explore methods of coping with persistent errors in membership queries beyond the $1 \rightarrow 0$ one-sided errors considered above. Since *pac*-learning can tolerate a large degree of random misclassification errors, general (two-sided) random errors in membership queries might seem approachable. Indeed, the results of Goldman, Kearns, and Schapire (1990) show that the classes of logarithmic-depth read-once majority formulas and logarithmic-depth positive NAND formulas can be learned with high probability using only membership queries, even if the membership queries are subject to persistent two-sided errors.

Kharitonov suggests the problem of combining our model with errors in equivalence queries. Persistent errors in deterministic equivalence queries are too strong an adversary; clearly, if an equivalence oracle ever claims the hypothesis is correct when it is not, the algorithm will fail. It is more interesting to ask what happens when the equivalence oracle is approximated by a sufficiently large number of labeled random examples, drawn according to some natural probability distribution. What happens if there is random misclassification noise in the sampling used by the equivalence oracle, in addition to "I don't know" or erroneous answers to membership queries?

It would be interesting to determine if anything can be done in the case of a *malicious* incomplete membership oracle, where an adversary is given some control over which membership queries the teacher cannot answer. In one such model, given failure bound p, the adversary is allowed to specify up to $p \times 2^n$ vectors for which the teacher cannot answer membership queries. Certainly, if p is a constant fraction, any algorithm can be forced to make an exponential number of queries to learn monotone DNF. For example, let $p = 1/4$; the adversary refuses to answer all queries on vectors whose first two bits are set to 1. Then the problem of learning any target concept where every term contains both x_1 and x_2 is effectively reduced to learning with just equivalence queries, which is known to require exponential time. Exactly how much power does an adversary need to prevent learning in a malicious model?

Another question is whether we can find polynomial time algorithms in this model for other learning problems known to have polynomial time algorithms using equivalence and membership oracles. For example, deterministic finite state acceptors (Angluin, 1987), simple deterministic languages (Ishizaka, 1990), read-once formulas (Angluin, Hellerstein, and Karpinski, 1993), μ-formula decision trees (Hancock, 1990), switch configurations (Raghavan and Schach, 1990), and propositional Horn sentences (Angluin, Frazier, and Pitt, 1992) all have such algorithms. Of these, the problem of propositional Horn sentences is the closest to the case of monotone DNF, but the basic algorithms are quite different. However, in each case there might be enough redundancy in the concepts that a constant fraction of missing answers to membership queries might not pose an insuperable problem.

Acknowledgements

We thank Avrim Blum, Prasad Chalasani, Joe Chang, Rob Schapire and Umesh Vazirani for enjoyable and helpful discussions of this material. We also gratefully acknowledge funding for this research from the National Science Foundation, in the form of a Graduate Fellowship to the second author and grant CCR-9014943 to the first author. The majority of the work was done while the second author was a student at Yale University and at the University of California at Berkeley. A preliminary version of this paper appeared in the proceedings of the 1991 Workshop on Computational Learning Theory (Angluin and Slonim, 1991).

References

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation, 75*, 87-106.

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

Angluin, D. (1990). Negative results for equivalence queries. *Machine Learning, 5*, 121-150.

Angluin, D., Frazier, M., & Pitt, L. (1992). Learning conjunctions of Horn clauses. *Machine Learning, 9*, 147-164.

Angluin, D., Hellerstein, L., & Karpinski, M. (1993). Learning read-once formulas with queries. *JACM, 40*, 185-210.

Angluin, D., & Kharitonov, M. (1991). When won't membership queries help? In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, (pp. 444-454). New Orleans, LA: ACM Press.

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

Angluin, D., & Slonim, D. (1991). Learning monotone DNF with an incomplete membership oracle. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, (pp. 139-146). Santa Cruz, CA: Morgan Kaufmann.

Goldman, S., Kearns, M., & Schapire, R. (1990). Exact identification of circuits using fixed points of amplificationfunctions. In *Proceedings of the Thirty-First Annual Symposium on Foundations of Computer Science*, (pp. 193-202). St. Louis, MO: IEEE Computer Society Press.

Hancock, T. (1990). Identifying μ-formula decision trees with queries. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, (pp. 23-37). Rochester, NY: Morgan Kaufmann.

Ishizaka, H. (1990). Polynomial time learnability of simple deterministic languages. *Machine Learning, 5*, 151-164.

Karp, R.M. (1991). Probabilistic recurrence relations. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, (pp. 190-197). New Orleans, LA: ACM Press.

Kearns, M. (1990). *The Computational Complexity of Machine Learning.* Cambridge, MA: MIT Press. (Also, Doctoral dissertation, Harvard University, 1989.)

Kearns, M., & Li, M. (1988). Learning in the presence of malicious errors. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, (pp. 267-280). Chicago, IL: ACM Press.

Kearns, M., Li, M., Pitt, L., & Valiant, L.G. (1987). On the learnability of Boolean formulae. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, (pp. 285-295). New York, NY: ACM Press.

Kearns, M., & Valiant, L. (1989). Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, (pp. 433-444). Seattle, WA: ACM Press.

Laird, P. (1987). *Learning from Good Data and Bad.* Doctoral dissertation, Department of Computer Science, Yale University, New Haven, CT. (Published as *Learning from Good and Bad Data*, Kluwer Academic Publishers, 1988.)

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning, 2*, 285-318.

Maass, W. (1991). On-line learning with an oblivious environment and the power of randomization. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, (pp. 167-175). Santa Cruz, CA: Morgan Kaufmann.

Raghavan, V., & Schach, S. (1990). Learning switch configurations. In *Proceedings of Third Annual Workshop on Computational Learning Theory*, (pp. 38-51). Rochester, NY: Morgan Kaufmann.

Sakakibara, Y. (1991). On learning from queries and counterexamples in the presence of noise. *Information Processing Letters, 37*, 279-284.

Shackelford, G. & Volper, D. (1988). Learning k-DNF with noise in the attributes. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, (pp. 97-103). Cambridge, MA: Morgan Kaufmann.

Sloan, R. (1988). Types of noise in data for concept learning. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, (pp. 91-96). Cambridge, MA: Morgan Kaufmann.

Sloan, R. (1989). *Computational Learning Theory: New Models and Algorithms*. Doctoral dissertation, MIT Laboratory for Computer Science.

Valiant, L. (1984). A theory of the learnable. *CACM, 27*, 1134-1142.

Valiant, L. (1985). Learning disjunctions of conjunctions. In *Proceedings of the 9th IJCAI*, (pp. 560-566). Los Angeles, CA: Morgan Kaufmann.