# Learning With Unreliable Boundary Queries


**Avrim Blum**[*]
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
avrim@theory.cs.cmu.edu

**Prasad Chalasani**
Los Alamos National Lab.
Los Alamos, NM 87544
chal@lanl.gov

**Sally A. Goldman**[†]
Dept. of Computer Science
Washington University
St. Louis, MO 63130
sg@cs.wustl.edu

**Donna K. Slonim**[‡]
MIT Lab. for CS
545 Technology Square
Cambridge, MA 02139
slonim@theory.lcs.mit.edu



## Abstract

We introduce a new model for learning with membership queries in which queries near the boundary of a target concept may receive incorrect or "don't care" responses. In partial compensation, we assume the distribution of examples has zero probability mass on the boundary region. The motivation behind this model is that the reason for the incorrect (or "don't care") response is that these examples are extremely rare in practice. Thus, it does not matter how the learner classifies them.

We present several positive results in this new model. We show how to learn the intersection of two halfspaces when membership queries near the boundary may be answered incorrectly. Our algorithm is an extension of an algorithm of Baum [6, 5] which learns intersections of two homogeneous halfspaces in the PAC-with-membership-queries model. We also describe algorithms for learning several subclasses of monotone DNF formulas.


## 1 Introduction

In most of the theoretical work in concept learning it is assumed that there is a well-defined boundary separating positive from negative examples. In practice, though, classification is often unclear. For example, consider a membership query algorithm for learning to recognize the number 3 from pixel images. A typical strategy would involve taking a 3 and a non-3 (maybe a picture of a 2) and asking for classifications of examples halfway between them until two nearby examples with different classification are found. A problem with this type of approach[1], as noticed by Lang and Baum [16] is that questions of this sort that are near the concept boundary may result in unreliable answers. Merging an image of a 2 and a 3 tends to produce something that (A) looks a bit like both and (B) we don't really care about anyway, since we don't expect to see one in practice.

More generally, one unrealistic aspect of the PAC-with-membership-query model is that it relies much more heavily on its assumptions than the passive PAC model. Consider, for instance, the above situation of learning images of 3's in which the learner is using some simple hypothesis class (say a simple neural network). For a passive algorithm, one would want the data seen to be consistent with some hypothesis in the class (or nearly so). For a membership query algorithm, however, one needs the stronger condition that the target concept *can actually be represented* in such a simple form. The difference is that typical images of 3's may be distinct enough from images of other characters that many simple consistent hypotheses exist. However, if one were to probe the exact boundary of the "3" concept, one would likely find it has a complicated structure that even depends on which "expert" you ask.

In this paper we propose and study a model for learning with membership queries that addresses the above issues. The basic idea of our model is that queries near the boundary of a target class may receive either incorrect or "don't care" responses. But, in partial compensation, we assume the distribution of examples has zero probability mass on the boundary region. (The motivation is that the oracle gives incorrect or "don't care" answers because these examples do not actually appear in the world and thus it does not matter how the learner classifies them.) We do not *require* the oracle to answer incorrectly or "don't care" in the boundary region, since that would just make the learning problem that of learning a different (perhaps ternary) target concept in the standard model; for instance, one could then simply perform binary search between the boundary and non-boundary examples, defeating the purpose of the model. One way of viewing our model (actually our model is a bit more general) is that the true target concept is in fact some horribly complicated func-


[*]Supported in part by NSF National Young Investigator grant CCR-9357793 and a Sloan Foundation Research Fellowship.

[†]Supported in part by NSF Grant CCR-9110108 and NSF National Young Investigator grant CCR-9357707 with matching funds provided by Xerox Corporation.

[‡]Supported by NSF Grant CCR-93-10888.


[1]Particularly when a human "expert" serves as the membership query oracle.

tion, but differs from a simple function only in a boundary region that has zero probability measure.

The contributions of this work are: (1) the introduction of the model of learning with unreliable boundary queries, (2) an efficient algorithm that PAC-learns the intersection of two half-spaces with membership queries when the boundary queries are noisy, and (3) efficient algorithms to exactly learn (with membership queries) several subclasses of monotone DNF formulas when there are one-sided false positive errors in the boundary queries for a small boundary size.

## 2 Definitions

We assume the reader is familiar with Valiant's probably approximately correct (PAC) learning model [22] and Angluin's model of learning with membership and equivalence queries [2]. We use *PAC-memb* to refer to the variation of the PAC model in which the learner can make membership queries. Likewise we say that a concept class is *exactly learnable* if it is learnable with membership and equivalence queries.

Given a concept $f$ over an instance space that has a distance metric, we say that the *distance to the boundary* of an example $x$ is the distance to the nearest example $y$ such that $f(x) \neq f(y)$. For continuous input spaces we use the infimum over distances to $y$'s such that $f(x) \neq f(y)$. In the boolean domain we use the Hamming distance as our metric. Thus an example is at distance 2 from the boundary if it is possible to flip two bit positions and change its classification. We define the *boundary region of radius* $r$ to be the set of examples whose distance to the boundary is at most $r$. We define the *negative boundary region of radius* $r$ to be the set of all examples $x$ in the boundary region such that $f(x) = 0$.

We now define the *unreliable boundary query* (UBQ) model. This model is the same as the standard PAC-memb model except for the following difference: there is a value $r$ (the boundary radius) such that (A) any query to an example in a boundary region of radius $r$ *may* receive an incorrect response, and (B) the example distribution $D$ has zero probability measure in that boundary region. In the *incomplete boundary query* (IBQ) model, the learner never receives an incorrect response to a query, but in the boundary region might receive the answer "don't care". We also consider a one-sided false-positive-only UBQ model in which the learner may receive false positive answers to any queries in the negative boundary region, and the distribution $D$ is only required to have zero probability on the negative boundary region. Finally, we extend these definitions to the exact learning model by requiring that counterexamples to equivalence queries not be chosen from the boundary region.

## 3 Related Work

There has been much theoretical work on PAC or mistake-bound learning in cases where the training examples may be mislabelled [2, 15, 20, 13] and additional work in models which allow attribute noise [19, 10, 17]. The p-concepts model of Kearns and Schapire [14] falls somewhat into this category and is related to our work since their model allows a "graded" boundary between the positive and negative portions of the instance space.

There have also been a number of results on learning with randomly generated noisy responses to membership queries. Sakakibara [18] considered the case where each membership query is incorrectly answered with a fixed probability, so that one could increase reliability by asking the *same* membership query a sufficient number of times and taking a majority vote. A more realistic model is that of *persistent* membership query noise in which repeated queries to the same example receive the same answer as in the first call. Goldman, Kearns and Schapire [11] gave a positive result for learning certain classes of read-once formulas under this noise model. Their work uses membership queries to simulate a particular distribution. Frazier and Pitt [9] showed that CLASSIC sentences are learnable in this noise model, using the fact that many distinct membership queries can be formulated that redundantly give the same information.

Angluin and Slonim [4] introduced a model of incomplete membership queries, in which a membership query on a given instance may persistently generate a "don't know" response. The "don't know" instances are chosen uniformly at random from the entire domain and may account for up to a constant fraction of the instances. Additional positive results in this model were obtained by Goldman and Mathias [12]. This model allows for a large number of "don't know" instances, but positive results in this model are typically highly dependent on the precisely uniform nature of the noise.

Sloan and Turan [21] introduced the *limited membership query model*. In this model, an adversary may arbitrarily select some number $\ell$ of examples on which it refuses to answer membership queries (or answers "don't know"), but the number of queries the learner asks may be polynomial in $\ell$. Sloan and Turan presented algorithms in this model for learning the class of monotone $k$-term DNF formulas with membership queries alone and the class of monotone DNF formulas with membership and equivalence queries. Angluin and Kriķis [3] introduced a similar model of *malicious membership queries* in which the adversary may respond with *incorrect* answers instead of "don't know". Their paper proved that the class of monotone DNF formulas is learnable in this model. Angluin [1] has shown that read-once DNF formulas are also learnable with malicious membership queries.

The main difference in motivation between our model and those above is that instead of supposing that there is a clear boundary between the positive and negative examples but with some noise included, we are attempting to model the very different situation in which the classification of examples in the boundary region is just not well defined (for example, a "2" merged with a "3"). Our model is more difficult than those above in that the membership query errors or omissions are chosen by an *adversary* (unlike [4]) and algorithms must run in time that is polynomial in the usual parameters regardless of the number of queries that might receive incorrect answers (unlike [21, 3]). For example, in the case of a 1-term monotone DNF formula with the boundary radius $r = 1$, there may be exponentially many (in $n$) instances in the boundary region. (Example: let $x_4x_7x_9$ be the target term. Then all positive instances, and all negative instances with exactly one of $\{x_4, x_7, x_9\}$ turned off, are in the boundary region of radius 1.) On the other hand, to partially compensate for this difficulty, we restrict membership query errors or omissions to the boundary region and we require that counterexamples to equivalence queries be chosen from outside the boundary region.

In other related work, Frazier, Goldman, Mishra and Pitt [8] introduced a learning model in which there is incomplete information about the target function due to an ill-defined boundary. While the omissions in their model may be adversarially placed, all examples labeled with "?" (indicating unknown classification) must be consistent with knowledge about the concept class. In other words, the classification of any instance labeled with "?" should not be possible to determine from knowledge of the concept class and the positive and negative instances. They require the learner to construct a *ternary* function with values {0,1,?} that, with high probability, correctly classifies most randomly drawn instances. One of the key differences between their model and ours is that they allow time polynomial in the complexity of that ternary function: thus if the "?" region has a complicated shape, then their learner is allowed a correspondingly longer time. Once again, a goal of our work is to produce algorithms whose running time is polynomial in the usual parameters *regardless* of the number of queries that might receive incorrect answers. In the model of Frazier, *et al.* the time complexity depends heavily on the placement of the "?" examples. In their paper they give positive results for the classes of monotone DNF formulas and $d$-dimensional boxes. Negatively, they show that learning the conjunctions of Horn clauses in their model is as hard as learning DNF. They also give a general technique for converting a standard PAC (or PAC-memb) algorithm for any concept class closed under union and intersection to an algorithm that learns in their model.

## 4 Learning an Intersection of Two Halfspaces

We now describe one of our main positive results: an algorithm for learning an intersection of two halfspaces in the UBQ model, for any boundary radius $r$ (see Figure 1). Our algorithm is an extension of an algorithm of Baum [6, 5] for learning the simpler class of intersections of two *homogeneous* halfspaces in the standard PAC-with-queries model[2].

The idea of Baum's algorithm is to reduce the problem of learning an intersection of two homogeneous halfspaces to the problem of learning an XOR of halfspaces, for which a PAC algorithm exists [7]. (That algorithm produces a hypothesis that is the threshold of a degree-2 polynomial.) The idea of the reduction is to notice that negative examples in the quadrant opposite from the positive quadrant—the troublesome examples keeping the data set from being consistent with an XOR of halfspaces—are exactly those examples $\vec{x}$ such that $-\vec{x}$ is positive. His algorithm is as follows:

> Draw a sufficiently large set $S$ of examples.[3] Mark all of the negative examples $\vec{x} \in S$ which have the property that a membership query to $-\vec{x}$ returns "positive". Then find a linear function $P$ such that $P(\vec{x}) < 0$ for all the marked (negative) examples and $P(\vec{x}) \geq 0$ for all the positives. Finally, run the XOR-of-halfspaces learning algorithm of [7] to find a hypothesis $H'$ that correctly classifies $\{\vec{x} \in S : P(\vec{x}) \geq 0\}$. The final hypothesis is:
>
> "If $P(\vec{x}) < 0$ then predict negative, else predict $H'(\vec{x})$."

Baum's algorithm seems appropriate for our model because it does not explicitly try to query examples near the boundary. In fact, it is "almost true" that if a negative example has distance at least $r$ from the boundary, then the example $-\vec{x}$ has distance at least $r$ from the boundary as well. This fails only on the negative examples in the "A-shaped" region shown in Figure 1.

Our algorithm for learning an intersection of (not necessarily homogeneous) halfspaces in the UBQ model is a small extension of Baum's algorithm, though the analysis requires a bit more care. In our algorithm, instead of reflecting through the origin, we reflect through a positive example. We use a potential function to prove that some "good" positive example for reflection must exist. (The algorithm tries all of them.) Specifically, our algorithm is the following:

> Draw a sufficiently large set $S$ of examples.
>
> For each positive example $\vec{x}_{pos} \in S$ do the following. For each negative example $\vec{x}_{neg} \in S$, query the example $2\vec{x}_{pos} - \vec{x}_{neg}$, and if the response to that query is "positive", then mark $\vec{x}_{neg}$. Now, attempt to find a linear function $P$ such that $P(\vec{x}) < 0$ for all the marked (negative) examples and $P(\vec{x}) \geq 0$ for all the positives. If no such function exists, then repeat this step using a different positive example $\vec{x}_{pos} \in S$ (we prove below that this step must succeed for *some* positive example $\vec{x}_{pos}$).
>
> Finally (assume we have found a legal linear function $P$), let $S'$ be the set of $\vec{x} \in S$ such that $P(\vec{x}) \geq 0$, and use the XOR-of-halfspaces learning algorithm to find a hypothesis $H'$ that correctly classifies the examples in $S'$. The final hypothesis is:
>
> "If $P(\vec{x}) < 0$ then predict negative, else predict $H'(\vec{x})$."

**Theorem 1** *For any radius $r$ of the boundary region, our algorithm succeeds in the UBQ model.*

Before giving a proof of correctness, we point out the simplifying observation that our algorithm is invariant under translation. If we add some vector $\vec{v}$ to each $\vec{x} \in S$, this

[2] A halfspace is homogeneous if its bounding hyperplane passes through the origin.

[3] The VC-dimension of the hypothesis class is $O(n^2)$ so a corresponding number of examples are needed.

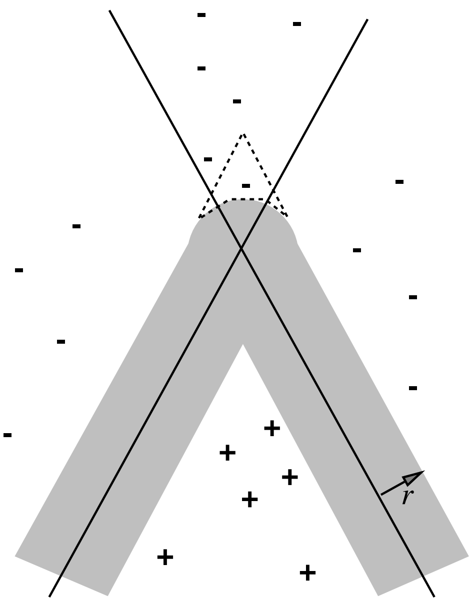


Figure 1: An intersection of 2 halfspaces. Boundary region is shaded. Notice that its apex is curved, which complicates the proof somewhat.

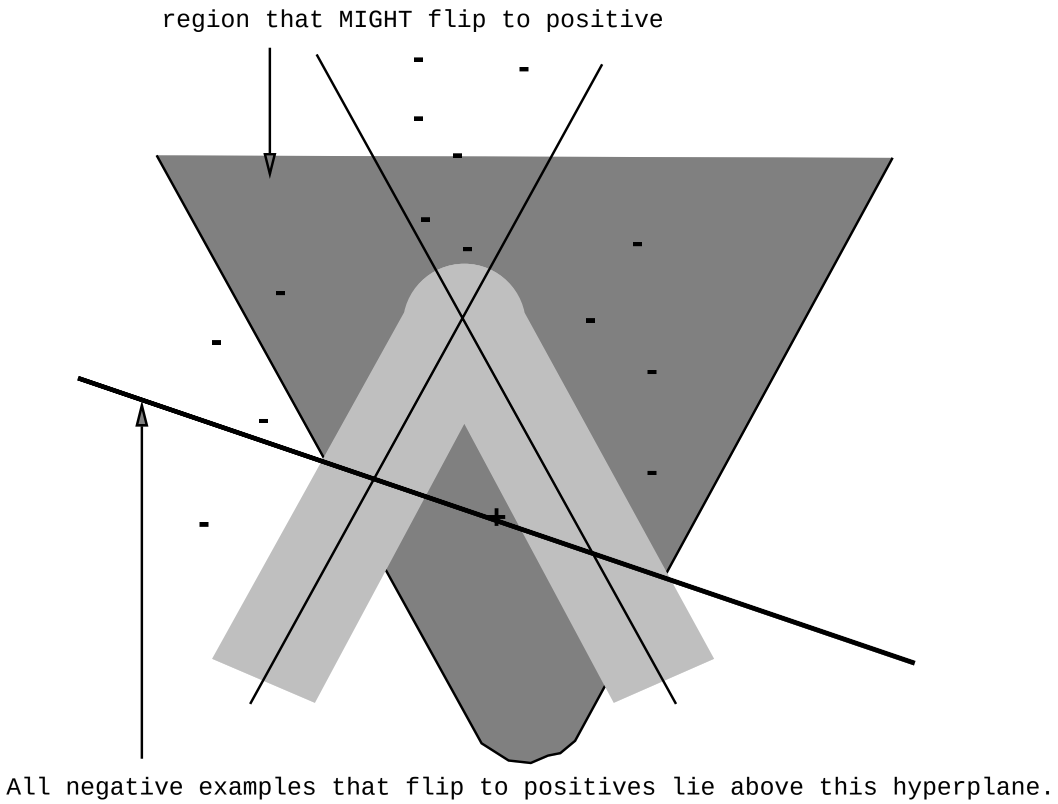


Figure 2: For clarity, $\vec{x}_{pos}$ is the only positive example shown. All marked negative examples lie within the convex hull of the dark-shaded region. Lemma 2 states that the intersection of this region with the non-boundary negative region is linearly separable from the positive region. The hyperplane pictured is the linear equality $P(x) = 2$ from that lemma.

results in adding $\vec{v}$ to each point of the form $2\vec{x}_{pos} - \vec{x}_{neg}$ as well. In particular, this means that if we can prove that our algorithm succeeds when the hyperplanes are homogeneous, then this implies that our algorithm also succeeds in the general (non-homogeneous) case. Therefore, we will assume in our proof for simplicity that the hyperplanes are, in fact, homogeneous.

We now fix some notation. Let $r$ be the radius of the boundary region (which, notice, is not used by the algorithm). The target concept is defined by two *unit* vectors $\vec{p}_1$ and $\vec{p}_2$, and the positive region $POS = \{\vec{x} : \vec{p}_1 \cdot \vec{x} \geq 0 \text{ and } \vec{p}_2 \cdot \vec{x} \geq 0\}$. We define the "opposite quadrant" to be $\{\vec{x} : \vec{p}_1 \cdot \vec{x} < 0 \text{ and } \vec{p}_2 \cdot \vec{x} < 0\}$. We say a point (or example) is "non-boundary" if it is not within the boundary region.

The negative non-boundary region $NEG_{nb}$ is the set of negative points not in the boundary region. I.e.,

$$NEG_{nb} = \{\vec{x} : (\vec{p}_1 \cdot \vec{x} < 0 \text{ or } \vec{p}_2 \cdot \vec{x} < 0) \text{ and } d(\vec{x}, POS) > r\}.$$

Notice that if either $\vec{p}_1 \cdot \vec{x} < -r$ or $\vec{p}_2 \cdot \vec{x} < -r$ then $\vec{x}$ is in $NEG_{nb}$, though these are not necessary conditions (see Figure 1). In fact, let us define

$$NEG_{far} = \{\vec{x} : \vec{p}_1 \cdot \vec{x} < -r \text{ or } \vec{p}_2 \cdot \vec{x} < -r\},$$

so $NEG_{far} \subseteq NEG_{nb}$. To get necessary conditions for lying in the region $NEG_{nb}$, notice that if

$$-r \leq \vec{p}_1 \cdot \vec{x} < 0 \text{ and } \vec{x} \in NEG_{nb}$$

then it must be the case that $(\vec{x} + r\vec{p}_1) \cdot \vec{p}_2 < 0$ (otherwise the point $\vec{y} = \vec{x} + r\vec{p}_1$ would be in the positive region). Similarly, if

$$-r \leq \vec{p}_2 \cdot \vec{x} < 0 \text{ and } \vec{x} \in NEG_{nb}$$

then $(\vec{x} + r\vec{p}_2) \cdot \vec{p}_1 < 0$. Thus,

$$\begin{aligned} NEG_{nb} \subseteq\ & NEG_{far} \cup \\ & \{\vec{x} : (\vec{p}_1 \cdot \vec{x} < 0 \text{ and } \vec{p}_2 \cdot \vec{x} < -r\vec{p}_1 \cdot \vec{p}_2) \text{ or} \\ & (\vec{p}_2 \cdot \vec{x} < 0 \text{ and } \vec{p}_1 \cdot \vec{x} < -r\vec{p}_1 \cdot \vec{p}_2)\} \qquad (1) \end{aligned}$$

We begin by showing that the negative examples in the opposite quadrant do in fact get marked by our algorithm.

**Lemma 2** *For any non-boundary positive example $\vec{x}_{pos}$ and any negative example $\vec{x}_{neg}$ in the opposite quadrant, the point $2\vec{x}_{pos} - \vec{x}_{neg}$ is a non-boundary positive example.*

**Proof:** Since $\vec{x}_{neg}$ lies in the opposite quadrant, we have

$$\vec{p}_1 \cdot (2\vec{x}_{pos} - \vec{x}_{neg}) \geq \vec{p}_1 \cdot 2\vec{x}_{pos} > r$$

and

$$\vec{p}_2 \cdot (2\vec{x}_{pos} - \vec{x}_{neg}) \geq \vec{p}_2 \cdot 2\vec{x}_{pos} > r.$$

□

What remains to be shown is that there exists a positive example $\vec{x}_{pos}$ such that the set of negative examples marked when using $\vec{x}_{pos}$ for reflection is linearly separable from the positives. In particular, we show the positive example $\vec{x} \in S$ that minimizes $(\vec{p}_1 \cdot \vec{x} + r)(\vec{p}_2 \cdot \vec{x} + r)$ will succeed. Letting $\vec{x}_{pos}$ be that example and $a_1 = \vec{p}_1 \cdot \vec{x}_{pos}$ and $a_2 = \vec{p}_2 \cdot \vec{x}_{pos}$, we show that a legal separator is the linear equation $\frac{\vec{p}_1 \cdot \vec{x} + r}{a_1 + r} + \frac{\vec{p}_2 \cdot \vec{x} + r}{a_2 + r} \geq 2$.

**Lemma 3** *Let $\vec{x}_{pos}$ be the example $\vec{x} \in S$ minimizing $(\vec{p}_1 \cdot \vec{x} + r)(\vec{p}_2 \cdot \vec{x} + r)$ and let $a_1 = \vec{p}_1 \cdot \vec{x}_{pos}$ and $a_2 = \vec{p}_2 \cdot \vec{x}_{pos}$. Then the linear function*

$$P(\vec{x}) = \frac{\vec{p}_1 \cdot \vec{x} + r}{a_1 + r} + \frac{\vec{p}_2 \cdot \vec{x} + r}{a_2 + r}$$

*is at least 2 for each positive example $\vec{x}$ and at most 2 for each negative example $\vec{x}$ marked when using $\vec{x}_{pos}$ for reflection.*

**Proof:** First we consider the positive examples. Let $\vec{x}'$ be some positive example in $S$. Define $\alpha$ and $\beta$ so that $(\vec{x}' \cdot \vec{p}_1 + r) = \alpha(a_1 + r)$ and $(\vec{x}' \cdot \vec{p}_2 + r) = \beta(a_r + r)$. By definition of $\vec{x}_{pos}$ we have $\alpha\beta \geq 1$, and by definition of the positive region we know both $\alpha$ and $\beta$ are at least 0. These inequalities imply that $\alpha + \beta \geq 2$, which implies $P(\vec{x}') \geq 2$.

Now consider the negative examples. The set of examples $\vec{x}$ with the property that $2\vec{x}_{pos} - \vec{x}$ might be classified as positive by a membership query is pictured in Figure 2. This set is contained within the set

$$\text{MAYFLIP} = \{\vec{x} : \vec{p}_1 \cdot \vec{x} \leq 2a_1 + r \text{ and } \vec{p}_2 \cdot \vec{x} \leq 2a_2 + r\}.$$

We now consider the possible cases for marked negative examples $\vec{x} \in S$ from Equation (1) (cases 1 and 2 below handle the possibility that $\vec{x} \in NEG_{far}$).

**Case 1.** Suppose $\vec{x} \in \text{MAYFLIP} \cap \{\vec{x} : \vec{p}_1 \cdot \vec{x} < -r\}$. Then $P(\vec{x}) < 0 + \frac{2a_2 + 2r}{a_2 + r} = 2$.

**Case 2.** Suppose $\vec{x} \in \text{MAYFLIP} \cap \{\vec{x} : \vec{p}_2 \cdot \vec{x} < -r\}$. Then $P(\vec{x}) < \frac{2a_1 + 2r}{a_1 + r} + 0 = 2$.

**Case 3.** Suppose $\vec{x} \in \{\vec{x} : \vec{p}_1 \cdot \vec{x} < 0 \text{ and } \vec{p}_2 \cdot \vec{x} < -r\vec{p}_1 \cdot \vec{p}_2\}$. Then $P(\vec{x}) < \frac{r}{a_1 + r} + \frac{r(-\vec{p}_1 \cdot \vec{p}_2 + 1)}{a_2 + r} < 1 + \frac{2r}{a_2 + r}$, which is at most 2 since $a_2 \geq r$.

**Case 4.** Suppose $\vec{x} \in \{\vec{x} : \vec{p}_2 \cdot \vec{x} < 0 \text{ and } \vec{p}_1 \cdot \vec{x} < -r\vec{p}_1 \cdot \vec{p}_2\}$. Same reasoning as Case 3. □

**Proof of Theorem 1:** Follows from Lemmas 2 and 3. □

## 5 Learning Subclasses of Monotone DNF Formulas

In this section we describe algorithms to learn two subclasses of monotone DNF formulas in the UBQ model with one-sided false-positive error, for small values of the boundary radius $r$. (Learnability in the one-sided UBQ model implies learnability in the IBQ model by simply treating each "don't care" response as positive.)

Specifically, we give an algorithm to learn the class of "read-once monotone DNF formulas in which each term has size at least 4" in the UBQ model with boundary radius $r = 1$. While this is clearly a highly-restrictive class, it is as hard to learn as general DNF formulas in the passive PAC model. Thus it does demonstrate that unreliable queries provide some power over the passive model in a boolean setting. We also give an algorithm to properly learn a subclass of constant-term DNF formulas for $r$ any constant. While the class of $k$-term

DNF formulas is learnable in the passive model, membership queries are required for proper learnability.

Let $y_1, \ldots, y_n$ denote the $n$ boolean variables, and $x = (x_1, \ldots, x_n)$ denote an example. As is commonly done, we view the sample space, $\{0,1\}^n$, as a lattice with top element $\{1\}^n$ and bottom element $\{0\}^n$. The elements are partially ordered by $\leq$, where $v \leq w$ if and only if each bit in $v$ is less than or equal to the corresponding bit in $w$. The *descendants* (respectively, *ancestors*) of a vector $v$ are all vectors $w$ in the sample space such that $w \leq v$ (respectively, $w \geq v$). For a monotone term, by moving down in the lattice (i.e. changing a 1 to 0), the term can only be "turned off". Thus every monotone term can be uniquely represented by the minimum vector in the ordering $\leq$ for which it is true.

Let $A(i,v)$ be the set of examples obtained by flipping exactly $i$ zeros to ones in vector $v$. The *parents of* $v$ are the elements of $A(1,v)$, and the *grandparents of* $v$ are the elements of $A(2,v)$. Likewise, $D(i,v)$ are the set of examples obtained by flipping exactly $i$ ones to zeros in $v$, and for a set $V$ of examples we let $D(i,V) = \cup_{v \in V} D(i,v)$, We define the *children of* $v$ as all elements in $D(1,v)$, and the *siblings of* $v$ are all elements in $D(1, A(1,v))$.

We often think of examples as terms and vice-versa, associating with a monotone term the minimal positive example that satisfies it. For example $v$ let $term(v)$ denote the most specific monotone term that $v$ satisfies. Thus, we say an example is a sibling of a term, meaning that it is a sibling of that term's associated example. Given an example $x$ we define $vars(x)$ to be the set of variables set to 1 by $x$. We also treat a term $t$ as the set of variables it contains.

We now describe the high-level algorithm that is used to obtain both of our results. Following our definitions in Section 2, we say an example $x$ is *in the boundary region of term* $t$ if $t(x) = 0$, but there exists an $x' > x$ such that $t(x') = 1$ and *dist*$(x, x') \leq r$. Our hypothesis $h$ contains candidates for terms of the target function $f$, and possibly some additional terms used to ensure that provided counterexamples are not in the boundary region of any known terms. We begin with $h = \emptyset$, and perform a loop in which we make an equivalence query with our current hypothesis, and then perform a collection of membership queries to update our hypothesis in light of the counterexample received, until a successful equivalence query is made. We maintain the invariant that after $i$ positive counterexamples have been received, $h_t$ contains $i$ distinct terms $t_1, \ldots, t_i$ of the target concept (and possibly other terms that may or may not be in the target concept). We define the set of *boundary examples* $B = \{v \mid v$ is in the positive or boundary region of some term in $h \cap f\}$.

We now describe how a counterexample $x$ is processed so that we can maintain this invariant. When it receives a negative counterexample, our algorithm simply removes from $h$ all terms that classify $x$ as positive. Clearly no term from $f$ (or even terms in the boundary of $f$) will be removed. If $x$ is a positive counterexample we first run the procedure Exit-Boundary($x$), which returns an example $v \notin B$ such that MQ($v$) is positive (which could be a false positive).

We then run the following process to "reduce" $v$ so it is "near" a new target term. To ensure that we do not rediscover a known term, the procedure Move-Further *must* return an example that is not in $B$. Below is our procedure, which is guaranteed to add a new term from $f$ to $h_T$.

1. Let $v$ be the example returned from Exit-Boundary. (So $v$ is positive or in the boundary region of a new term of $f$.)
2. Now, so long as $v$ has some child to which a membership query reports "positive", replace $v$ by that child and repeat. (Note that since $v \notin B$ and the target formula is monotone, it follows that no child of $v$ could be in $B$.)
3. We now call a procedure Move-Further($v$) for which one of the two cases will occur.

   **Case 1:** Move-Further($v$) returns an example $v' \notin B$ such that $v'$ has strictly fewer ones than $v$ and MQ($v'$) is positive. In this case we return to step 2 using $v'$ as the current example.

   **Case 2:** Move-Further($v$) reports failure. Here we are guaranteed that $v$ is "near" a new term $t_{i+1}$ of $f$. Formally, we have that

   $$|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq r.$$

   That is, the number of irrelevant variables in $vars(v)$ is at most the number of relevant variables from $t_{i+1}$ missing from $vars(v)$, which in turn is at most $r$. In this case we call the procedure Generate-Candidates($v$) that returns a polynomial-sized set $T$ of terms with the guarantee that $t_{i+1} \in T$. We then add all terms in $T$ to $h$.

At this point our algorithm is ready to make its next equivalence query.

**Lemma 4** *Given that* Exit-Boundary, Move-Further, *and* Generate-Candidates *satisfy the stated conditions and run in polynomial time for subclass $\mathcal{C}$ of monotone DNF formulas, the above procedure learns $\mathcal{C}$ in the false-positive-only UBQ model (or the IBQ model) in polynomial time.*

**Proof Sketch:** If there are no counterexamples to $h$, we are done. The number of positive counterexamples received is at most the number of terms in the target DNF. Once Exit-Boundary is completed all examples that the "reduce" process recurses on have the property that they are positive examples (possibly false positive) not in the positive or boundary region of any term of $h \cap f$. Thus from the correctness of Move-Further and Generate-Candidates, we are guaranteed that a new term is added to $h$ after at most $n$ calls to Move-Further. Furthermore, each negative counterexample removes at least one "extra" term placed in $h$ by Generate-Candidates and we are guaranteed that there are at most a polynomial number of such terms. Thus, there are only a polynomial number of negative counterexamples, so our algorithm runs in polynomial time as long as all of the provided procedures do. □

### 5.1 Learning A Subclass of Read-Once Monotone DNF Formulas

We now describe how to complete the generic procedure above to obtain an algorithm that learns the class $\mathcal{C}$ of "read-once monotone DNF formulas in which each term has size at least 4" in the UBQ model where $r = 1$.

We begin by describing a utility routine, Study-Example, used in the algorithm. This routine takes an example returned by Move-Further (which is guaranteed to be "near to" some term of the target) and produces a more useful approximation to that term. The desired behavior of the routine Study-Example is specified in Property 1.

**Property 1** *Let $f$ be a function in $\mathcal{C}$, and let $v$ be an example such that there exists a term $t_{i+1}$ of $f$ such that $v$ is either equal to, a sibling of, or a child of $t_{i+1}$. Then* Study-Example *produces an approximation $\hat{t}_{i+1}$ of $t_{i+1}$ along with one of these two guarantees:*

*(1) $\hat{t}_{i+1}$ is equal to $t_{i+1}$ or a parent of $t_{i+1}$ (so it is a superset of $t_{i+1}$), or*

*(2) $\hat{t}_{i+1}$ is equal to $t_{i+1}$ or a child of $t_{i+1}$ (so it is a subset of $t_{i+1}$).*

The Study-Example routine asks a membership query on all siblings of $v$. Let $P$ be the set of siblings for which the membership oracle replied "yes." Then Study-Example outputs based on the following cases:

1. If $P = \emptyset$, let $\hat{t}_{i+1} = term(v)$ and report "subset".
2. Otherwise, let $u$ be the term containing exactly the variables in $\cup_{p \in P} vars(p)$.
   (a) If $|u| = |term(v)| + 1$, let $\hat{t}_{i+1} = u$ and report "superset".
   (b) Otherwise ($|u| > |term(v)| + 1$), if some variable $y_i \in vars(v)$ is "responsible for" at least two of the variables in $u - vars(v)$ in the sense that two variables in $u - vars(v)$ are set to 1 in examples setting $y_i$ to 0, then let $\hat{t}_{i+1} = u - \{y_i\}$ and report "subset". (If there are several such variables $y_i$, just pick one.)
   (c) Else let $\hat{t}_{i+1} = u$ and report "superset". (Note: we separate this case from case (a) just for convenience in the proof.)

**Lemma 5** *The routine* Study-Example *as described above correctly satisfies Property 1.*

**Proof:** Note that no siblings of $v$ are in the boundary region of any of the *other* terms in the target function. That is because a sibling of $v$ may have at most two variables set to 1 that are *not* in $t_{i+1}$, and every other term must have at least four variables not in $t_{i+1}$ (since they all have size at least 4 and the target function is read-once). Thus, we may analyze the routine as if $t_{i+1}$ were the only term in the target function.

We can see that Study-Example behaves correctly in case (1) by noting that if $v$ is positive but none of $v$'s siblings are positive (or false positive), then $term(v)$ is either $t_{i+1}$ or a child of $t_{i+1}$. The correctness of case (2a) is similarly easy to see because if $v$ is a child of $t_{i+1}$ then $u$ equals $t_{i+1}$ and otherwise $u$ is a parent of $t_{i+1}$. Notice that if $v$ is a child of $t_{i+1}$ then either case (1) or (2a) occurs.

The reasoning for case (2b) is as follows. If $v = t$ then it is clear. On the other hand, if $v$ is a sibling of $t_{i+1}$, then the only variable $y_i$ in $v$ that can possibly be "responsible for" more than one other variable in $u - vars(v)$ is the variable *not* in term $t_{i+1}$. In fact, if we are not in cases (1) or (2a) and $v$ is a sibling of $t_{i+1}$, then case (2b) *must* occur because $y_i$ will be responsible for the variable in $t_{i+1}$ missing from $v$ (several variables may be responsible for this one) as well as any other variables added to $u$. Thus, case (2c) holds because it can only be reached if $v = t$. □

We now prove our main result of this subsection.

**Theorem 6** *The class of read-once monotone DNF formulas where each term in the target formula has at least four variables is exactly learnable in the false-positive-only UBQ model (or the IBQ model) for boundary radius $r = 1$ using polynomial queries and time.*

**Proof:** From Lemmas 4 and 5 we know that our algorithm maintains the invariant that after $i$ positive counterexamples have been received, $h$ contains at least $i$ distinct terms of the target function $f$ and a collection of approximations $\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_i$ each corresponding to different terms $t_1, t_2, \ldots, t_i$ of $f$, and labeled as "subset" or "superset" appropriately.

We also maintain the invariant that all children of terms in $h \cap f$ are in $h$. This invariant is initially enforced by Generate-Candidates whenever a new term is placed in $h$. Finally, once a term of $f$ or any of its children are placed in $h$, they cannot be removed by any negative counterexample (since a child of a term in $f$ is in the boundary region of $f$). Thus any positive counterexample received is not in $B$. So Exit-Boundary($x$) simply returns $x$.

We now describe the procedure Move-Further($v$). Note that the input $v$ has the property that it is not in $B$ and MQ($v$)=1.

1. For each $\hat{t}_j$ (for $j = 1, 2, \ldots, i$) labeled as "subset", set every variable in $vars(\hat{t}_j)$ to 0 in $v$. (This new example is still in the positive or boundary region of a new term since $f$ is read-once. And since $f$ is monotone this example is not in $B$.) We fix these variables at 0 for the remainder of this procedure.
2. Let $V$ be the set of variables set to 0 by $v$ and not fixed to 0 in Step 1. For each variable $y_\ell \in V$, consider the example $v'$ obtained from $v$ by flipping to 0 all variables in the terms $\hat{t}_j$ that contain $y_\ell$, and then flipping $y_\ell$ to 1. Let $P$ be the set of all such examples for which a membership query reports "positive".
3. (a) If there is an example in $P$ that has fewer 1's than $v$, then return this example.
   (b) If not, then query all children and grandchildren of examples in $P$ and if one of them has fewer 1's

than $v$ and is reported as positive, then return this example.

(c) Otherwise report failure.

Move-Further maintains the invariant that $v \notin B$ and $v$ is in the positive or boundary region of a new term of $f$. We have already argued that this holds after Step 1. Thus, at this point, there exists some term $t_{i+1} \in f$, distinct from $t_1, \ldots, t_i$, such that $v$ sets to 0 at most one variable in $t_{i+1}$. The reason is simply that since $v \notin B$, it immediately follows that $v$ is not in the boundary of any of the terms $t_1, \ldots, t_i$.

We now argue that each example in $P$ has at most one variable in common with term $t_j$ for $1 \leq j \leq i$. If $v' \in P$ was obtained by flipping to 1 some variable appearing in, say, term $t_j$ ($j \leq i$) then either (A) $\hat{t}_j$ is a "subset" of $t_j$ and therefore this is the *only* variable that $vars(v')$ has in common with $t_j$ (since all others in $t_j$ have been fixed to 0) or else (B) $\hat{t}_j$ is a "superset" of $\hat{t}_j$ in which case this variable is also in $\hat{t}_j$ and so to obtain $v'$ we flipped all the rest of the variables in $\hat{t}_j$ to 0. Thus no example in $P$ is in $B$.

So if an example is returned in step (3a) or (3b) then it has the desired property. We now argue that if step (3c) reports failure then $v$ satisfies $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq 1$. We have already argued that $v$ is the the positive or boundary region of a new target term, $t_{i+1}$, and thus at most one relevant variable from $t_{i+1}$ is missing from $vars(v)$. Furthermore, if $vars(v)$ contains exactly the variables in $t_{i+1}$ then all irrelevant variables would be removed by the standard reduce procedure. If $vars(v)$ is missing one relevant variable, $y_\ell$ from $t_{i+1}$ then when $y_\ell$ is added in step (2), the membership query would be positive and thus this example is added to $V$. Now if there were two or more variables in $vars(v)$ that were not in $t_{i+1}$ then the example in which two of those variables were set to 0 would be returned in either step (3a) or (3b). Thus if we reach step (3c) the required property must hold.

The procedure Generate-Candidates$(v)$ first calls Study-Example$(v)$. Let $t$ be the term returned. If Study-Example reports "subset" then place $t$ and its children into $T$. Otherwise, if Study-Example reports "superset" then place $t$ and its parents into $T$. From Lemma 5 it follows that $t_{i+1} \in T$. Finally return $T \cup D(1, T)$. □

### 5.2 Learning $(r+1)$-Separable $k$-Term Monotone DNF Formulas

We now show that a subclass of monotone $k$-term DNF formulas are properly learnable in the false-positive-only UBQ model for any constant boundary radius. We say that two terms $t_i$ and $t_j$ are *$\ell$-separable* if there are $\ell$ variables in $t_j$ that are not in $t_i$, and there are $\ell$ variables in $t_i$ that are not in $t_j$. *A monotone DNF formula $f$ is $\ell$-separable* if all pairs $(t_i, t_j)$ of terms of $f$ are $\ell$-separable.

**Theorem 7** *The class of $(r+1)$-separable $k$-term monotone DNF formulas is exactly learnable in the false-positive-only UBQ model (or the IBQ model) using polynomial queries and time (for $r$ and $k$ constant). Furthermore, all equivalence queries made by our algorithm are $(r+1)$-separable $k$-term monotone DNF formulas.*

**Proof:** We first prove this result under the assumption that Generate-Candidate not only finds a set of candidates that contains some new term of the target formula, but has the power to "guess" which one is right. Thus $h$ will always contain a subset of the terms of the target. Then we argue that our algorithm can be modified to remove this assumption.

Exit-Boundary$(v)$ performs a membership query on examples not in $B$ obtained by setting up to $(k-1)r$ variables in $vars(v)$ to 0. It returns the first such example for which the membership oracle replies "yes." We now prove that Exit-Boundary is correct.

**Lemma 8** *The procedure* Exit-Boundary *successfully returns an example $v \notin B$ for which $MQ(v) = 1$.*

**Proof:** We must show that given a positive counterexample to hypothesis $h$, Exit-Boundary$(v)$ returns an example $v$ that is not in $B$ and for which MQ$(v) = 1$.

Since $v$ was a positive counterexample, it must be in the non-boundary positive region of some term $t_{new}$ in $f - h$. Suppose it is also in the boundary region of some terms in $h$. Consider one such term $t_{known}$. Since $f$ is $(r+1)$-separable, if we set to 0 the variables in $vars(v) \in t_{known} - t_{new}$ then $v$ will no longer be in the boundary of $t_{known}$. However, we still know that all variables in $t_{new}$ are in $vars(v)$ since we do not change any variables in $t_{new}$. (In fact, if all $r+1$ variables in $t_{known} - t_{new}$ are 1 in $v$, then it suffices to pick any $r$ of them to set to 0, since we know that $v$ is already in the boundary region of $t_{known}$.) We can repeat this for the at most $(k-1)$ other terms in $h$. Thus after setting at most $r(k-1)$ variables in $var(v)$ to 0 we obtain an example that is not in $B$ and is in the truly positive region of $t_{new}$. Since this is one of the examples queried by Exit-Boundary we know that at least one membership query will respond "yes". Of course, it could be that another membership query responds "yes". However, in this case we are still guaranteed that the example $v$ returned is not in $B$ (since we do not query those in $B$), and MQ$(v)$ is positive. □

The procedure Move-Further works as follows. For each $i$ such that $r+1 \leq i \leq rk$, it performs a membership query on examples $v'$ in $D(i, A(r, v))$ and returns $v'$ if MQ$(v') = 1$. If no such examples are found after all values of $i$ have been tried, then it returns "failure."

We now argue that when Move-Further$(v)$ reports failure the following two properties hold:

1. Example $v$ sets to 0 at most $r$ variables from term $t_{i+1}$ of the target formula (i.e. $|t_{i+1} - vars(v)| \leq r$).
2. The number of variables *not* in $t_{i+1}$ that are one in $v$ is at most the number of variables *in* $t_{i+1}$ that are zero in $v$ (i.e. $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)|$).

Since $v \notin B$ and MQ$(v) = 1$ it must be in the positive or boundary region of some new term from $f$. Since the adversary can reply "positive" only on an example that sets to 0 at most $r$ variables from a term in $f$, the first property follows. We now prove that the second property holds. Let

$t_{i+1}$ be any term of $f$ for which $v$ has $\ell \leq r$ variables set to 0. Suppose that the second property fails. Thus there are at least $\ell + 1$ variables not in $t_{i+1}$ that are one in $v$. Since the target is $(r+1)$-separable we know that $t_{i+1}$ is not in $B$. Since $t_{i+1}$ has at most $r$ variables set to 0 in $v$, at least one example in $A(r, v)$ has all variables in $t_{i+1}$ set to 1. When adding these $r$ 1's, we have at worst just set to 1 $r$ variables in each of the known terms. Thus there exist at most $rk$ variables that when set to 0 takes us to an example not in $B$. Since we try all $i$ such that $r + 1 \leq i \leq kr$, we know that we query some point in the positive region of $t_{i+1}$ that is not in $B$ (since we must query $t_{i+1}$ or one of its ancestors). But this contradicts the fact that $v$ was returned. Thus the second property holds. Also note that only a polynomial number of examples were queried (since $r$ and $k$ are constant). Thus this procedure runs in polynomial time.

Generate-Candidates($v$) lets $T = \cup_{i=0}^{r} D(i, A(r, v))$ and non-deterministically guesses which one is in $f$. It follows from the correctness of Generate-Candidates that $t_{i+1}$ is placed in $T$.

To remove the need to non-deterministically select the right term from $T$ we just try all guesses halting when failure is detected because a negative counterexample is received or a $(k+1)$st positive counterexample is received. Since $r$ and $k$ are constant, only a polynomial number of runs occur and thus the overall queries and time complexity are still polynomial. □

The proof of Theorem 7 can be extended to obtain the following result.

**Corollary 9** *The class of 2-term monotone DNF formulas is exactly learnable by the class of 2-term monotone DNF formulas in polynomial time in the false-positive-only UBQ model (or the IBQ model) with a boundary region of radius $r$ (for constant $r$).*

**Proof Sketch:** Let $f = t_1 + t_2$. If $t_1$ and $t_2$ are $(r+1)$-separable then the result immediately follows. Thus, without loss of generality, assume that $t_2$ has all variables from $t_1$ except at most $r$ of them, as well as any number of additional variables. If $t_1$ is placed in $h$ first then no counterexample is created by $t_2$ since it is entirely contained within the boundary region of $t_1$. If $t_2$ is placed in $h$ first, then we receive a positive counterexample for $t_1$ (unless it is contained within $t_2$'s boundary in which case we are done). This counterexample is processed to add $t_1$ to $h$. □

## 6 Concluding Remarks

We have introduced two related models of learning with noise near the boundary of the target concept, and we have presented positive results in these models in both continuous and discrete domains. However, there is much more work to be done. The algorithms described here learn fairly simple concept classes. We do not yet know how to extend these results to learn general monotone DNF formulas or the intersection of more than two halfspaces. One eventual goal might be a general result describing ways to transform classes of PAC-memb or exact-learning algorithms to work in the IBQ or UBQ model.


## Acknowledgements

The third author thanks Lenny Pitt for valuable discussions on this material.


## References


[1] D. Angluin. Exact learning of $\mu$-DNF formulas with malicious membership queries. Technical Report YALEU/DCS/TR-1020, Yale University Department of Computer Science, March 1994.

[2] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[3] D. Angluin and M. Kriķis. Learning with malicious membership queries and exceptions. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pages 57–66. ACM Press, New York, NY, 1994.

[4] D. Angluin and D. K. Slonim. Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14(1):7–26, January 1994. A preliminary version of this paper appeared in COLT '91.

[5] E. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks*, 2:5–19, 1991.

[6] E. B. Baum. Polynomial time algorithms for learning neural nets. In *Proc. 3rd Annu. Workshop on Comput. Learning Theory*, pages 258–272, San Mateo, CA, 1990. Morgan Kaufmann.

[7] A. Blum and R. L. Rivest. Training a 3-node neural net is NP-Complete. In *Advances in Neural Information Processing Systems I*, pages 494–501. Morgan Kaufmann, 1989.

[8] M. Frazier, S. Goldman, N. Mishra, and L. Pitt. Learning from a consistently ignorant teacher. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pages 328–339. ACM Press, New York, NY, 1994. To appear, *J. of Comput. Syst. Sci.*

[9] M. Frazier and L. Pitt. CLASSIC learning. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pages 23–34. ACM Press, New York, NY, 1994.

[10] S. Goldman and R. Sloan. Can PAC learning algorithms tolerate random noise? Technical Report WUCS-92-25, Washington University Department of Computer Science, July 1992. To appear, *Algorithmica*.

[11] S. A. Goldman, M. J. Kearns, and R. E. Schapire. Exact identification of circuits using fixed points of amplification functions. *SIAM J. Comput.*, 22(4):705–726, August 1993.

[12] S. A. Goldman and H. D. Mathias. Learning k-term DNF formulas with an incomplete membership oracle. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 77–84. ACM Press, New York, NY, 1992.

[13] M. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM J. Comput.*, 22:807–837, 1993.

[14] M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *Proc. of the 31st Symposium on the Foundations of Comp. Sci.*, pages 382–391. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[15] Philip D. Laird. *Learning from Good and Bad Data*. Kluwer international series in engineering and computer science. Kluwer Academic Publishers, Boston, 1988.

[16] K.J. Lang and E.B. Baum. Query learning can work poorly when a human oracle is used. In Proceedings of *International Joint Conference on Neural Networks*, IEEE, 1992.

[17] N. Littlestone. Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 147–156, San Mateo, CA, 1991. Morgan Kaufmann.

[18] Y. Sakakibara. On learning from queries and counterexamples in the presence of noise. *Inform. Proc. Lett.*, 37(5):279–284, March 1991.

[19] G. Shackelford and D. Volper. Learning $k$-DNF with noise in the attributes. In *Proc. 1st Annu. Workshop on Comput. Learning Theory*, pages 97–103, San Mateo, CA, 1988. Morgan Kaufmann.

[20] R. Sloan. Types of noise in data for concept learning. In *Proc. 1st Annu. Workshop on Comput. Learning Theory*, pages 91–96, San Mateo, CA, 1988. Morgan Kaufmann.

[21] R. H. Sloan and G. Turán. Learning with queries but incomplete information. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pages 237–245. ACM Press, New York, NY, 1994.

[22] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.