Report Number: WUCS-97-17

1997-01-01

Learning with Unreliable Boundary Queries

Avrim Blum, Prasad Chalasani, Sally A. Goldman, and Donna K. Slonim

We introduce a model for learning from examples and membership queries in situations where the boundary between positive and negative examples is somewhat ill-defined. In our model, queries near the boundary of a target concept may receive incorrect or "don't care" responses, and the distribution of examples has zero probability mass on the boundary region. The motivation behind our model is that in many cases the boundary between positive and negative examples is complicated or "fuzzy." However, one may still hope to learn successfully, because the typical examples that one sees to not come from that region. We present... **Read complete abstract on page 2.**

Learning with Unreliable Boundary Queries

Avrim Blum, Prasad Chalasani, Sally A. Goldman, and Donna K. Slonim

Complete Abstract:

We introduce a model for learning from examples and membership queries in situations where the boundary between positive and negative examples is somewhat ill-defined. In our model, queries near the boundary of a target concept may receive incorrect or "don't care" responses, and the distribution of examples has zero probability mass on the boundary region. The motivation behind our model is that in many cases the boundary between positive and negative examples is complicated or "fuzzy." However, one may still hope to learn successfully, because the typical examples that one sees to not come from that region. We present several positive results in this new model. We show how to learn the intersection of two arbitrary halfspaces when membership queries near the boundary may be answered incorrectly. Our algorithm is an extension of an algorithm of Baum [7, 6] that learns the intersection of two halfspaces whose bounding planes pass through the origin in the PAC-with-membership-queries model. We also describe algorithms for elarning several subclasses of monotone DNF formulas.

Learning with Unreliable Boundary Queries

Avrim Blum, Prasad Chalasani, Sally A. Goldman and Donna K. Slonim

WUCS-97-17

February 1997

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

Learning With Unreliable Boundary Queries

Avrim Blum[*] Prasad Chalasani[†] Sally A. Goldman[‡] Donna K. Slonim[§]

February 19, 1997

Abstract

We introduce a model for learning from examples and membership queries in situations where the boundary between positive and negative examples is somewhat ill-defined. In our model, queries near the boundary of a target concept may receive incorrect or "don't care" responses, and the distribution of examples has zero probability mass on the boundary region. The motivation behind our model is that in many cases the boundary between positive and negative examples is complicated or "fuzzy." However, one may still hope to learn successfully, because the typical examples that one sees do not come from that region.

We present several positive results in this new model. We show how to learn the intersection of two arbitrary halfspaces when membership queries near the boundary may be answered incorrectly. Our algorithm is an extension of an algorithm of Baum [7, 6] that learns the intersection of two halfspaces whose bounding planes pass through the origin 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 on concept learning, the environment is modeled as an omniscient oracle that classifies all objects as positive or negative instances of the concept to be learned. Thus, it is assumed that there is a well-defined boundary separating positive from negative examples. In many cases, however, classification may be much less clear. 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 [22], 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

[*]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891. avrim@theory.cs.cmu.edu. Supported in part by NSF National Young Investigator grant CCR-9357793 and a Sloan Foundation Research Fellowship.

[†]Los Alamos National Laboratory, Los Alamos, NM 87544. chal@lanl.gov.

[‡]Department of Computer Science, Washington University, St. Louis, MO 63130. sg@cs.wustl.edu. Supported in part by NSF Grant CCR-9110108 and NSF National Young Investigator grant CCR-9357707 with matching funds provided by Xerox Corporation.

[§]MIT Center for Genome Research, One Kendall Square, Building 300, Cambridge, MA 02139. slonim@genome.wi.mit.edu. This work was supported by NSF Grant CCR-93-10888 and NIH Grant 1 T32 HG00039-01.

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

something that looks a bit like both, and that 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. In both models, one typically assumes there is a target function belonging to some class C that is labeling the data, and one then tries to prove that one's algorithm will succeed under that assumption. In the passive model, however, all that is really needed is that the target function and the distribution on examples conspire in such a way that the data *actually seen* is consistent with a function in C. In contrast, with membership queries one needs the function on the entire space to be consistent with some function in C. For instance, suppose a learning algorithm is using a simple hypothesis class (say a simple neural network) to learn images of 3's. For a passive algorithm, one would want the data observed to be consistent with some hypothesis in the class. For a membership query algorithm, however, one needs the stronger condition that the target concept *over the entire input space* 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 responds incorrectly or doesn't know the 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 state "I 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. In that case, 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 (although our model is actually a bit more general) is that the true target concept is in fact some horribly complicated function, but differs from a simple function only in a boundary region that has zero probability measure. See Figure 1 for an example.

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 halfspaces 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 can be "don't care" responses to boundary queries for a small boundary size.

2 Definitions

2.1 Standard Learning Models

A *concept* f is a boolean function over an instance space $\mathcal{X}$. In this paper we consider two instance spaces: the boolean domain $\{0,1\}^n$ and the continuous domain R^n. A point $x \in \mathcal{X}$ is an *example*, and is a *positive example* of f if $f(x) = 1$ and a *negative example* of f if $f(x) = 0$. A concept class C is a set of functions, along with an associated representation language for describing them. For instance, C

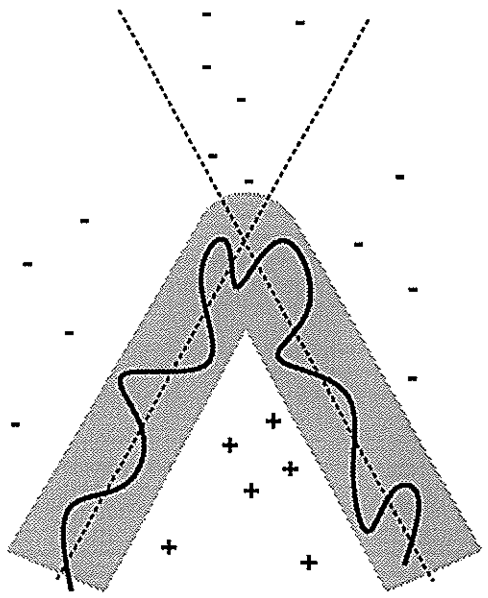

Figure 1: The thick curve is the actual concept boundary. However, because the distribution has zero probability mass in the shaded region, we can view the concept as an intersection of two halfspaces in our model.

might be the class of monotone DNF formulas.

In the distribution-free or PAC learning model [29], to obtain information about an unknown target function $f \in \mathcal{C}$, the learner is provided access to labeled (positive and negative) examples of f, drawn randomly according to some unknown target distribution D over $\mathcal{X}$. The learner is also given as input $\epsilon, \delta > 0$. Its goal is to output, with probability at least $1 - \delta$, the description of a function h that has probability at most ϵ of disagreeing with f on a randomly drawn example from D (thus, h has *error* at most ϵ). An algorithm $\mathcal{A}$ *PAC-learns* $\mathcal{C}$ if for any $f \in \mathcal{C}$, any distribution D, and any $\epsilon, \delta > 0$, $\mathcal{A}$ meets this goal and runs in time polynomial in n (the size of an example), $1/\epsilon$, $1/\delta$, and $|f|$ (the description length of the target function).

In the above definition, we have not specified what a "description of a function h" is. An algorithm is said to be a *proper* learning algorithm if the hypothesis h is always chosen from the description language associated with the concept class. On the other hand, more generally we may allow a learning algorithm to output *any polynomial-time algorithm* as a hypothesis. This less constrained model is sometimes called "PAC-predictability" [17, 16].

Another well-investigated model of learning is that of *exact learning from equivalence queries* [3]. In this model, the learner proposes as a hypothesis some $h \in \mathcal{C}$, and in response is told "yes" if $h = f$, or is given a counterexample x such that $h(x) \neq f(x)$. There is no distribution on examples; the learner is required to exactly identify f (obtain a "yes" answer) in time (and queries) polynomial in n (the length of the counterexamples) and $|f|$ (the size of the representation of the target f), regardless of the choice of target function and sequence of (adversarially chosen) counterexamples. It is not hard to see that any class learnable exactly from equivalence queries can be learned in the PAC setting [3] (though the converse does not hold [8]).

The PAC and exact learning models are *passive* in that the learner cannot directly affect the type of examples it receives as input — in the PAC setting they are randomly generated, and in the exact setting they are chosen by an adversary. Evidence suggests that only relatively simple types of concepts can be learned passively in this way [2, 20, 24]. Consequently, researchers have considered augmenting this learning protocol by allowing the learner to perform experiments. In addition to drawing a randomly labeled example (or posing a hypothesis, in the exact model), the learner can perform a membership query MQ(x) in which it supplies an example x in the instance space and is told the value $f(x)$.

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.

2.2 Our New Learning Model

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. In continuous domains, the L_2 metric[2] is often most natural. 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 any query to an example in a boundary region of radius r *may* receive an incorrect response, and 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". 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 [3, 21, 27, 18] and additional work in models that allow attribute noise [26, 13, 23]. The p-concepts model of Kearns and Schapire [19] also falls somewhat into this category.

There have also been a number of results on learning with randomly generated noisy responses to membership queries. Sakakibara [25] considers the case where each membership query is incorrectly answered with a fixed probability, but where one can increase reliability by asking the *same* membership query several times. In models of *persistent* membership query noise, repeated queries to the same example receive the same answer as in the first call. Goldman, Kearns and Schapire [14] give a

[2]Under the L_2 metric the distance $d(\vec{x}, \vec{y})$ between $\vec{x} = \{x_1, \ldots, x_n\}$ and $\vec{y} = \{y_1, \ldots, y_n\}$ is $\sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$.

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 [12] show that CLASSIC sentences are learnable in this noise model, using the fact that many distinct membership queries can be formulated that redundantly yield the same information.

Angluin and Slonim [5] introduce 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 are obtained by Goldman and Mathias [15]. 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 [28] introduce the *limited membership query model*. In this model, an adversary may arbitrarily select some number ℓ of examples on which it refuses to answer membership queries (or answers "don't know"), but the learner is now allowed to ask a number of queries polynomial in ℓ. Sloan and Turan present 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 [4] introduce a similar model of *malicious membership queries* in which the adversary may respond with *incorrect* answers instead of "don't know". Their paper proves 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 most previous work supposes that there is a clear boundary between the positive and negative examples with some noise included. Our goal is 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 the sense that the membership query errors or omissions are chosen by an *adversary* (unlike the random noise models [5]), 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 [28, 4]). 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.) The algorithms of Sloan and Turan, and of Angluin and Kriķis make use of the allowance to use time polynomial in the number of lies. In particular, if there are an exponential number of lies, then their algorithms will use exponential time and queries. 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 [11] introduce 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. They require the learner to construct a *ternary* function with values {0,1,?} that, with high probability, correctly classifies most randomly drawn instances, and give positive results for the classes of monotone DNF formulas and d-dimensional boxes. 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. In our model, a learner is required to run quickly regardless of the complexity of the "?" region.

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 n dimensions in the UBQ model, for any boundary radius r (see Figure 2). Our algorithm is an extension of an algorithm of Baum [7, 6] for learning the simpler class of intersections of two *homogeneous* halfspaces in the standard PAC-with-queries model[3].

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 [9]. (That algorithm works by noticing that the XOR of $\vec{v}\cdot\vec{x} \geq 0$ and $\vec{w}\cdot\vec{x} \geq 0$ is equivalent to the degree-2 threshold function $(\vec{v}\cdot\vec{x})(\vec{w}\cdot\vec{x}) < 0$ (except on degenerate inputs) which can be learned as a linear threshold function over an $O(n^2)$-dimensional space.) The idea of Baum's 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 $O\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n^2}{\epsilon}\lg\frac{1}{\epsilon}\right)$ examples.[4] 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 [9] 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 the case 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 2.

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 $O\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n^2}{\epsilon}\lg\frac{1}{\epsilon}\right)$ 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

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

[4] The VC-dimension of the hypothesis class is $O(n^2)$ so a corresponding number of examples as given by the results of Blumer et al. [10] are needed.

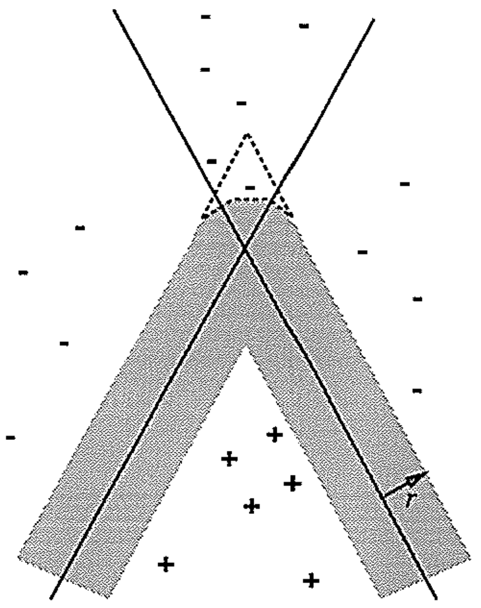

Figure 2: An intersection of 2 halfspaces. Boundary region is shaded. Notice that its apex is curved, which complicates the proof somewhat. Note that NEG_{nb} is the negative region that is *not* darkly shaded. The region NEG_{far} contains all the points from NEG_{nb} except for those in the dashed "A"-shaped region.

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. The sample size s for our algorithm is $O\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n^2}{\epsilon}\lg\frac{1}{\epsilon}\right)$ and the time complexity $O(s \cdot T_{LP}(n,s) + T_{LP}(n^2,s))$ where $T_{LP}(v,c)$ is the time complexity of solving a linear program over v variables with c constraints*[5].

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 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

[5]Using an algorithm due to Vaidya[30], $T_{LP}(v,c)$, the time needed to solve a linear programming problem with v variables and c constraints (examples) each specified to b bits of precision, is $O(bv(c+v)^{1.5})$.

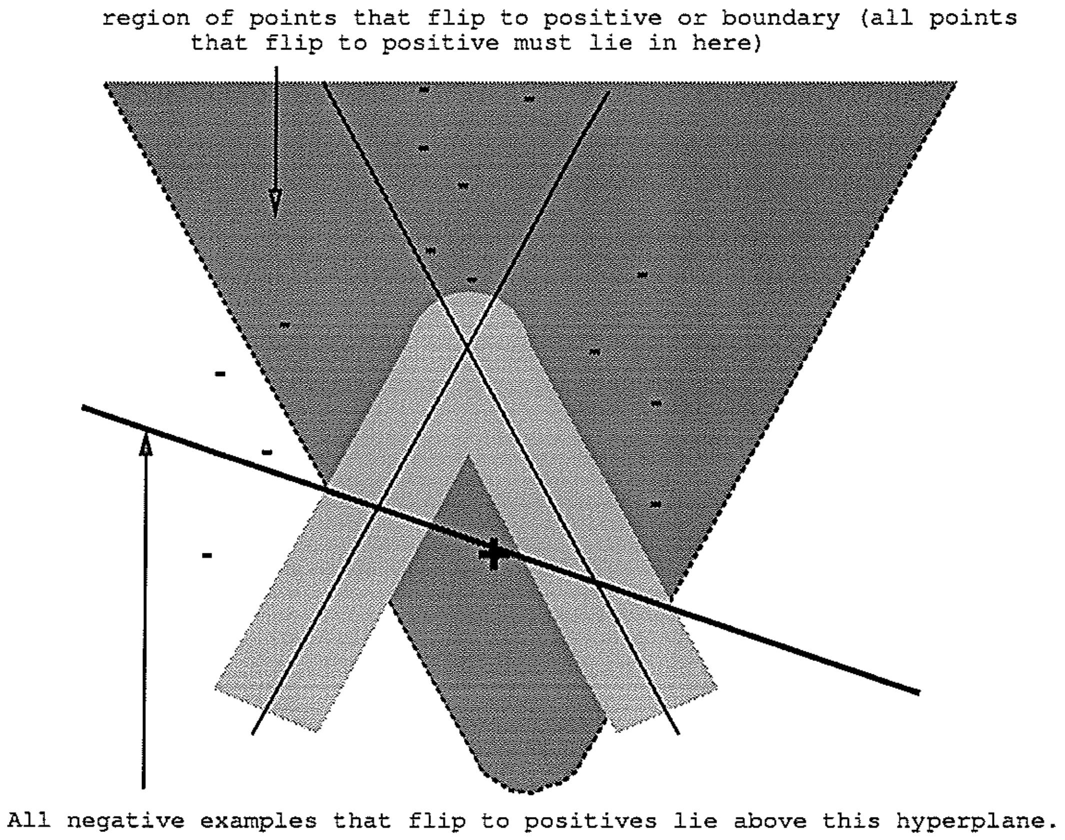
region of points that flip to positive or boundary (all points that flip to positive must lie in here)
All negative examples that flip to positives lie above this hyperplane.

Figure 3: For clarity, $\vec{x}_{pos}$ is the only positive example shown. All marked negative examples lie within the dark-shaded region, which is the reflection of the positive and boundary regions through $\vec{x}_{pos}$. Lemma 2 states that the intersection of this region with the non-boundary negative region is linearly separable from the set of positive examples in S. The hyperplane pictured is the linear equality $P(x) = 2$ from that lemma.

succeeds when the hyperplanes are homogeneous, then this implies that our algorithm also succeeds in the general (non-homogeneous) case. Therefore, we can 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). We define the distance between a point x and a set S as the infimum over all $y \in S$ of $d(x, y)$. 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. Formally,

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

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 2). 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}, suppose that $\vec{x} \in NEG_{nb} - NEG_{far}$. Since $\vec{x}$ is negative but not in NEG_{far}, either $-r \leq \vec{p}_1 \cdot \vec{x} < 0$ or $-r \leq \vec{p}_2 \cdot \vec{x} < 0$. However, since $\vec{x} \in NEG_{nb}$, any point within distance r of $\vec{x}$ must also be negative. In particular, if $-r \leq \vec{p}_1 \cdot \vec{x} < 0$ then it must be that $(\vec{x} + r\vec{p}_1) \cdot \vec{p}_2 < 0$, and if $-r \leq \vec{p}_2 \cdot \vec{x} < 0$ it must be that $(\vec{x} + r\vec{p}_2) \cdot \vec{p}_1 < 0$. Thus,

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

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

Lemma 1 *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 $\vec{p}_1 \cdot \vec{x}_{neg} < 0$ and $\vec{p}_2 \cdot \vec{x}_{neg} < 0$. Since $\vec{x}_{pos}$ is a non-boundary positive example, we know that $\vec{p}_1 \cdot \vec{x}_{pos} > r$ and $\vec{p}_2 \cdot \vec{x}_{pos} > r$. Therefore,

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

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 inequality $\frac{\vec{p}_1 \cdot \vec{x} + r}{a_1 + r} + \frac{\vec{p}_2 \cdot \vec{x} + r}{a_2 + r} \geq 2$. (Intuitively, what we want is for $\vec{x}_{pos}$ to be the "closest" positive example to the origin according to some measure, and the correct notion of "closest" is that of being on the hyperbola $(\vec{p}_1 \cdot \vec{x} + r)(\vec{p}_2 \cdot \vec{x} + r) = c$ for minimum c.)

Lemma 2 *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 $\vec{x}$ that is marked when using $\vec{x}_{pos}$ for reflection.

Proof: First we consider the positive examples. Let $\vec{y}$ be some positive example in S. Define $\alpha = (\vec{y} \cdot \vec{p}_1 + r)/(a_1 + r)$ and $\beta = (\vec{y} \cdot \vec{p}_2 + r)/(a_2 + r)$. By definition of $\vec{x}_{pos}$ we have $\alpha\beta \geq 1$, and by definition of the positive region we know both α and β are at least 0. These inequalities imply that $\alpha + \beta \geq 2$, which implies $P(\vec{y}) \geq 2$.

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 3. Any such example $\vec{x}$ must satisfy $\vec{p}_1 \cdot (2\vec{x}_{pos} - \vec{x}) \geq -r$ and $\vec{p}_2 \cdot (2\vec{x}_{pos} - \vec{x}) \geq -r$, and therefore must belong to 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 $\vec{x} \in S$, using the characterization of the negative non-boundary region given by 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: The correctness of our algorithm follows from Lemmas 1 and 2. For the time complexity, for each of the at most s positive examples, we do the following. First we must do a query for each of the at most s negative examples to mark the appropriate ones. Next, we attempt to find a linear separator (which can be done in $T_{LP}(n, s)$ time). Finally, the XOR algorithm requires solving a linear program over n^2 variables and s constraints. Thus the overall time complexity is $O(s(s + T_{LP}(n, s)) + T_{LP}(n^2, s)) = O(s \cdot T_{LP}(n, s) + T_{LP}(n^2, s))$. □

Note that the proof of Lemma 2 (and thus the proof of Theorem 1) relies on the fact that there are no examples from the boundary region. Otherwise, we could not prove that we would get the needed separation.

5 Learning Subclasses of Monotone DNF Formulas

In this section we describe algorithms to learn two subclasses of monotone DNF formulas in the IBQ model for small values of the boundary radius r. Our algorithms will treat each "don't care" response as positive. Thus one could view this result has learning in 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, but receives correct answers in the positive boundary region.

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 IBQ model with boundary radius $r = 1$. While this is clearly a highly-restrictive class, it is not difficult to show, using standard *prediction preserving reductions* [24], that it is as hard to learn as general DNF formulas in the passive PAC model. In particular, we can reduce an arbitrary DNF formula f to a read-once monotone DNF formula f' in which each term has size at least 4 by replacing each literal of f with the conjunction of four new variables, and mapping an example x for f into an example x' for f' such that $f(x) = f'(x')$ in the straightforward way. Thus our algorithm demonstrates 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 monotone DNF formulas for any constant r. While the class of k-term DNF formulas is learnable in the passive model, membership queries are required for proper learnability.

One reason for studying the one-sided, false-positive error model (which is what we obtain from the IBQ model by treating all "don't care" responses as positive) is that the monotonicity of the target class provides some inherent ability to handle false-negative errors. In a related model, Angluin and Slonim [5] show how to learn monotone DNF with random false-negative answers to membership queries allowed anywhere in the domain (not just in the boundary region). However, it is not known how to extend their results to handle false positive errors. We hope that the results presented here may be combined with these previous results to produce general techniques for learning monotone concepts with two-sided noise.

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 the relation $\leq$, 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 represented uniquely 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, $D(i, v)$ is 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. Our hypothesis h contains candidates for terms of the target function f, and possibly some additional terms used to ensure that counterexamples provided are not in the boundary region of any known terms. We begin with $h = \emptyset$. We then enter a loop in which we make an equivalence query with our current hypothesis, and then ask 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 contains at least 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).

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 $dist(x, x') \leq r$. We define the set of *boundary/positive examples* $B = \{v \mid v$ is in the positive or boundary region of some term in $h \cap f\}$, where "$h \cap f$" denotes the set of terms that appear in both formulas. Observe that since each term is monotone, B is monotone. Note that B depends not only on f, but on the current hypothesis h as well. Thus, there may be some example $u \in B$ such that u is in the boundary region of some term t in h, but is a truly positive example of the target function f. Such a u might be returned as a positive counterexample to an equivalence query made on h. Thus to maintain the invariant, we need to show that we can always use counterexample u to find a *new* term of f not already in h.

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 will be removed. No terms in the boundary of f will be removed either, since no counterexamples are chosen from the boundary region.

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 (this 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.

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. So long as v has some child to which a membership query reports "positive", replace v by that child and repeat. This is the standard Reduce procedure common to many monotone-DNF learning algorithms. (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 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. Example v is "near" t_{i+1} if 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. Formally, we require that

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

   In this case we call the procedure Generate-Candidates(v), which 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 3 *Given that* Exit-Boundary, Move-Further, *and* Generate-Candidates *satisfy the stated conditions and run in polynomial time for some subclass C of monotone DNF formulas, the above procedure learns C in the IBQ model (or the false-positive-only UBQ model) in polynomial time.*

Proof: 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) outside 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. □

We now briefly describe some of the difficulties we encountered in designing an algorithm to learn the class of general monotone DNF formulas in the IBQ model. The challenge is to develop an algorithm that finds new terms of the target concept sufficiently often and that avoids rediscovering known terms exponentially-many times. Each positive counterexample must be positive for at least one new term, so one might hope to add that new term to the hypothesis each time. However, if the "reduce" procedure is allowed to move through positive or "don't care" examples that are in the boundary region of some known term, then there is a chance of moving away from the new term entirely, so that all the positive or "don't care" examples nearby are only positive for (or in the boundary of) previously-discovered terms.

We avoid this hazard by using the *Exit-Boundary* procedure. The motivation for defining *Exit-Boundary* as we do is the following. Once the learner has an example $v \notin B$ such that $\mathrm{MQ}(v)$ is positive (this could be a false positive), then, at least for constant r, a procedure like the standard "reduce" procedure can be used to find a polynomial-sized set of terms, one of which will be a new term of the target concept. The drawback of requiring *Exit-Boundary* to return an example classified as positive or don't care that is *not* in B is that for some general monotone DNF target functions and some subsets h of the terms in the target, no such example exists. That is, all positive counterexamples might be in the boundary of some already known term. Even when such an example exists, it may not be possible to move from the initial counterexample x to an example $y \notin B$ such that the Hamming distance between x and y is constant and $\mathrm{MQ}(y)$ is positive. Thus designing an efficient *Exit-Boundary* procedure is hard in general.

For the subclasses described in this section, we solve these problems in two different ways. In the first of our algorithms, the concept class is restricted so that by building a hypothesis that contains a subset of the terms of h and the "children" for each such term in h, the counterexample we obtain already satisfies the requirements for *Exit-Boundary*. For the second of our algorithms, we show that by examining (via a membership query) a polynomial number of examples we are able to find one that satisfies the requirements for *Exit-Boundary*.

One alternative approach we considered was to begin the "reduce" procedure from the given counterexample, but to add the extra constraint of never moving to an example already classified as positive by the hypothesis. One difficulty here is that there may be incorrect terms in the hypothesis preventing the algorithm from reducing towards the new target term. Another is that once again, the learner can move from the new term satisfied by the original counterexample to an example that is in the boundary region for a known term (and no longer near any term in $f-h$). Because of these difficulties, we adopted our current approach.

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 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 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 (where v is the example returned by Move-Further). Let P be the set of siblings for which the membership oracle replied "yes." Let $P' = P \cup \{v\}$, and let t be the term that contains exactly the variables in $\bigcup_{p \in P'} vars(p)$. The procedure Study-Example outputs based on the following cases:

1. If $|t| = |term(v)| + 1$, let $\hat{t}_{i+1} = t$ and report "superset".
2. Else
   (a) If some variable $y_i \in vars(v)$ is "responsible for" at least two of the variables in $t - vars(v)$ in the sense that at least two variables in $t - vars(v)$ are set to 1 in examples of P setting y_i to 0, then let $\hat{t}_{i+1} = term(v) - \{y_i\}$ and report "subset". (If there are several such variables y_i, just pick one.) For example, let $v = 0011$ and $P = \{1001, 1010, 0101\}$. Then $t = y_1y_2y_3y_4$ and $t - vars(v) = \{y_1, y_2\}$. Here y_3 is responsible for y_1 and y_2, since y_3 is 0 in 1001 and 0101. However, y_4 is only responsible for y_1. Thus in this case we would let y_i be y_3 and $\hat{t}_{i+1} = y_4$.

   (b) Else let $\hat{t}_{i+1} = term(v)$ and report "subset".

Lemma 4 *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 now consider three cases based on whether v is equal to, a sibling of, or a child of t_{i+1}. First suppose $v = t_{i+1}$. Notice that when Case 1 occurs, t must be a parent of v (since it has exactly one more variable). Thus the output of returning t and "superset" is correct since Study-Example returns a parent of $v = t_{i+1}$. In Case 2a Study-Example returns a child of t_{i+1} and "subset" which is correct, and in Case 2b Study-Example returns t_{i+1} itself (and "subset") which is correct.

Next we consider the case in which v is a child of t_{i+1}. That is v contains exactly the variables in t_{i+1} except some relevant variable y_{rel}. Note that any element of P that is positive (all would be false positives), must have set y_{rel} to 1 and set some irrelevant variable to 0. Thus if Case 1 occurs, $t = t_{i+1}$ and thus Case 1 is correct. Notice that in Case 2b, Study-Example returns $term(v)$ and subset which is correct since $term(v)$ is a child of t_{i+1}. Finally, notice that removing any relevant variable from $term(v)$ would give an example that must be reported as negative *unless* y_{rel} is set to 1. Thus no variable in $vars(v)$ could be responsible for more than the one variable, namely y_{rel}, in $t - vars(v)$ and thus Case 2a cannot apply here.

Finally, we consider the case in which v is a sibling of t_{i+1}. That is v contains exactly the variables in t_{i+1} except it is missing some relevant variable y_{rel} and includes one irrelevant variable y_{irrel}. Notice that t_{i+1} is a sibling of v and must be reported as positive. Thus t_{i+1} is in P and thus y_{rel} will be in t. Thus t will contain all the relevant variables of t_{i+1} along with some number of irrelevant variables. Note that if Case 1 occurs, then t contains exactly the variables in t_{i+1} plus y_{irrel}. Thus returning t and "superset" is correct since t is a parent of t_{i+1}. We now argue that if Case 1 does not occur (in which case $|t| \geq |term(v)| + 2$), then Case 2a must occur. Let y' be a variable (besides y_{rel}) that is in $t - vars(v)$. (Note that $vars(t) \supseteq vars(term(v))$ and thus since $|t| \geq |term(v)| + 2$, there are at least two variables in $t - vars(v)$). In Case 2a consider when $y_i = y_{irrel}$. Notice that when $y_i = 0$, there was an element of P corresponding to when either y_{rel} or y' were set to 1. Thus y_{irrel} is responsible for at least two vars in $t - vars(v)$ and thus either Case 1 or Case 2a applies. Finally, if any variable in $vars(v)$, besides y_{irrel}, is set to 0, an example is only placed in P if y_{rel} is 1 (otherwise, two relevant examples would be 0 and thus the membership oracle must respond with negative). Thus all variables in $vars(v)$ besides y_{irrel} could only be responsible for a single variable in $t - vars(v)$. Thus $\hat{t}_{i+1} = term(v) - \{y_{irrel}\}$ is a child of t_{i+1} and thus returning $\hat{t}_{i+1}$ and "subset" yields the correct answer. □

We now prove our main result of this subsection.

Theorem 2 *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 IBQ model (or false-positive-only UBQ model) for boundary radius $r = 1$ using $O(n^3)$ equivalence queries and $O(n^5)$ time and membership queries where n is the number of variables.*

Proof: By Lemma 3, we need only define subroutines Exit-Boundary, Move-Further, and Generate-Candidates, and show that they satisfy the conditions outlined in the previous section.

We first describe Generate-Candidates. The procedure Generate-Candidates(v) calls Study-Example(v). Study-Example(v) returns a term $\hat{t}_{i+1}$ and a label σ (either "subset" or "superset"). Next, add the pair $(\hat{t}_{i+1}, \sigma)$ to the set $\mathcal{L}$, which stores all the $\hat{t}_i$s and their labels. If label $\sigma =$ "subset" then place $\hat{t}_{i+1}$ and its children into T. Otherwise, if label $\sigma =$ "superset" then place $\hat{t}_{i+1}$ and its parents into T.

From Lemma 4 it follows that if Study-Example is called on example v such that v is equal to, a sibling of, or a child of a new term t_{i+1}, then t_{i+1} is placed in T. Thus the set $D(1, T)$ must include all children of t_{i+1}. We therefore have Generate-Candidates return $T \cup D(1, T)$, so all terms in $T \cup D(1, T)$ are added to h.

Next, we claim that the identity function satisfies all the requirements for the procedure Exit-Boundary. For positive counterexample x, we want Exit-Boundary(x) to return some example $v \notin B$

such that MQ(v) is positive. If x is a positive counterexample, then certainly MQ(x) is positive.

We now argue that x cannot be in B. Initially, whenever we add a new term t_{i+1} to h, we also add its children, as described above. (Note that it is not possible for Generate-Candidates to accidently add an additional term t_{i+2} to h without its children. If it did so, t_{i+2} would have to be in $D(1,T)$ but not in T. But since f is read-once and each term has size at least 4, it is not possible to have two terms of f, t_{i+1} and t_{i+2}, such that $t_{i+1} \in T$ and $t_{i+2} \in D(1,T)$.) Once a term of f or any of its children is placed in h, it cannot be removed by any negative counterexample (since any child of a term in f is in the boundary region of f). So no positive counterexample can be in B. Thus, Exit-Boundary(x) simply returns x.

We now describe the procedure Move-Further(v). Note that the input v has the properties that v is not in B and that MQ(v)=1. Furthermore, since v must have failed the standard Reduce procedure, $MQ(v')$ is negative for all $v' \in D(1,v)$.

1. For each $\hat{t}_j$ in $\mathcal{L}$ (for $j = 1, 2, \ldots, i$) that is labeled "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_ℓ, and then flipping y_ℓ 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.

Lemma 5 *The procedure* Move-Further(v) *as described above either returns an example $v' \notin B$ such that v' has strictly fewer ones than v and MQ(v') is positive, or it returns failure, in which case $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq 1$.*

Proof: 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}.

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 y appearing in, say, term t_j ($j \leq i$) then one of two cases holds. If $\hat{t}_j$ is a "subset" of t_j, then y is the *only* variable that $vars(v')$ has in common with t_j, since all others in t_j have been fixed to 0. Otherwise, $\hat{t}_j$ is a "superset" of t_j. In this case, y is also in $\hat{t}_j$, so to obtain v' we flipped all the rest of the variables in $\hat{t}_j$ to 0. Thus, since each term of f has at least 4 literals, no example in P is in B. So if an example v' is returned in step (3a) or (3b) then it has the desired properties: $v' \notin B$, MQ(v') is positive, and $|vars(v')| < |vars(v)|$.

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 $vars(v)$ (i.e., $|t_{i+1} - vars(v)| \leq 1$). Furthermore, if $vars(v)$ contains all the variables in t_{i+1} then $t_{i+1} = vars(v)$, because all irrelevant variables would have been removed by the standard Reduce procedure. (Recall that one input condition on v is that no children of v are positive.) Therefore, if $|t_{i+1} - vars(v)| = 0$, then $|vars(v) - t_{i+1}| = 0$ as well.

Otherwise, suppose that $|t_{i+1} - vars(v)| = 1$, so v is missing one relevant variable, y_ℓ, from t_{i+1}. Then when y_ℓ is added in step (2), the membership query would be positive and thus this example would be added to P. Now suppose there were two or more variables in $vars(v)$ that were not in t_{i+1}. Then an example in which two of those variables were set to 0 would have been returned in either step (3a) or (3b). Thus if we reach step (3c) we know that $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq 1$ holds. □

Finally, we claim that since the example v returned by Move-Further satisfies the property $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq 1$, v must either be equal to t_{i+1}, or be a sibling or a child of t_{i+1}. Thus, Study-Example is called on a vector satisfying the conditions of Property 1, proving the correctness of Generate-Candidates and of Theorem 2.

We now analyze the time and query complexity. Let m be the number of terms in the target formula (and since the formula is read-once we have that $m = O(n)$). There are at most m positive counterexamples since we are guaranteed to find at least one new term for each positive counterexample. We now analyze the time and membership queries used to process each positive example. For each such counterexample we must use Reduce and Move-Further until we are "stuck." We can move to a "lower" example in the lattice at most n times and for each move down we could perform a membership query on the at most n children and then use Move-Further. In the first step of Move-Further we use $O(n^2)$ time. In the second step we use $O(n^2 m)$ time and make at most n membership queries. Finally the third step uses $O(n^3)$ membership queries and time since $|P| \leq n$ and at most n^2 membership queries are made per element of P. Thus our reduction procedure uses $O(n^4)$ membership queries and $O(n(n^3 + n^2 m)) = O(n^4)$ time per positive counterexample. Finally, Study-Example makes $O(n^2)$ membership queries and uses $O(n^3)$ time, and Generate-Candidates returns a set $T \cup D(1, T)$ that has size at most $O(n^2)$ since T has size at most $n + 1$. Putting it all together, the number of counterexamples is at most $O(mn^2) = O(n^3)$ since n^2 terms could be placed in h for each of the up to m positive counterexamples. The time complexity and number of membership queries is $O(mn^4) = O(n^5)$. □

It is possible to extend the algorithm of this section to learn in the IBQ model, for any constant r, the class of "read-once monotone DNF formulas in which each term has size at least c" where c is a "constant" that depends on r. However, in the above proof the details of Study-Example, the proof that the identity function mets the needs for Exit-Boundary, and the Move-Further procedure all currently rely on the requirement that $r = 1$ (and $c = 4$).

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

We now show that a subclass of monotone k-term DNF formulas is properly learnable in the IBQ model (where again we will treat all "don't care" examples as positive) for any constant boundary radius. We say that two terms t_i and t_j are *ℓ-separable* if there are ℓ variables in t_j that are not in t_i, and there are ℓ variables in t_i that are not in t_j. A monotone DNF formula f is *ℓ-separable* if all pairs (t_i, t_j) of terms of f are ℓ-separable.

Theorem 3 *The class of $(r+1)$-separable k-term monotone DNF formulas is exactly learnable in the IBQ model (or false-positive-only UBQ 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. Our algorithm uses $O(n^{3rk+3r+k})$ time and queries.*

Proof: We first prove this result under the assumption that Generate-Candidates 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 always contains a subset of the terms of the target. Then we argue that our algorithm can be modified to remove this assumption.

We first define the procedure Exit-Boundary. For each term t_i of f already in h, choose a set s_i of $(r+1)$ variables in t_i. Let $S = \cup_i s_i$. Then let $v' = v$ with all variables in S set to 0. The procedure Exit-Boundary(v) performs a membership query on each possible v' obtained in this fashion, and returns the first such example for which the membership oracle replies "yes." Thus, each query sets up to $(r+1)(k-1)$ variables in $vars(v)$ to 0. We now prove that Exit-Boundary is correct.

Lemma 6 *The procedure* Exit-Boundary(v) *successfully returns an example $v' \notin B$ for which $MQ(v') =$ "positive."*

Proof: 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 zero $r+1$ 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 $(r+1)(k-1)$ variables in $vars(v)$ 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 is possible that some other membership query responds "yes". However, note that Exit-Boundary never queries any example in B, since all examples queried are obtained by setting $r+1$ variables from each known term to 0. Thus, in this case we are still guaranteed that the example v' returned is not in B and that MQ(v') is positive. □

The procedure Move-Further works as follows. For each i such that $r+1 \le i \le rk$, it performs a membership query on all examples v' in $D(i, A(r,v))$ that are not in B and returns the first v' for which MQ$(v') = 1$. If no such examples are found after all values of i have been tried, then it returns "failure." If Move-Further returns v', then v' must have strictly fewer ones than v, since $v' \in D(i, A(r,v))$ for some $i > r$.

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

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

Since $v \notin B$ and $\mathrm{MQ}(v) = 1$, v 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 new 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 x 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. For each term $t_i \in h$, let s_i be the set of all variables in t_i but not in t_{i+1}, and let $S = \cup_i s_i$. Let x' be example x with all variables in S set to zero. Since the target concept is monotone and $(r+1)$-separable, we know that x' is still in the positive region of term t_{i+1} and that $x' \notin B$. Move-Further queries examples in $D(i, A(r, v))$ for all i such that $r + 1 \leq i \leq kr$, so x' must be one of the examples queried by the procedure. Finally, since there are at least $\ell + 1$ variables not in t_{i+1} that are 1 in v, and since $x' \in D(i, A(r, v))$ for some $i > r \geq \ell$, x' must have strictly fewer ones than v. But this contradicts the assumption that Move-Further(v) reported failure. 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 Move-Further that t_{i+1} is placed in T.

We now analyze the time and query complexity for the non-deterministic case. In Exit-Boundary we consider turning off each subset of up to $(r+1)(k-1)$ of the n variables. Thus this step uses $O(n^{(r+1)(k-1)}) = O(n^{rk+k})$ time and queries. Reduce uses $O(n^2)$ time and queries. In Move-Further, we may have to query all elements of $D(i, A(r, v))$ for $r+1 \leq i \leq rk$. Since $|A(r, v)| = O(\binom{n}{r}) = O(n^r)$, then $|D(i, A(r, v))| \leq \sum_{i=0}^{rk} (n^r \cdot n^i) = O(n^{rk+r})$. Thus the time per call of Move-Further is $O(n^{rk+r})$. Finally, Move-Further may be called up to n times per each of the k positive counterexamples. The routine Generate-Candidates creates $O(n^{2r})$ candidates (of which one is the non-deterministically selected). Thus the overall time and sample complexity is $O(kn^{rk+r+k}) = O(n^{rk+r+k})$

To remove the need to non-deterministically select the right term from T we just try all guesses. We halt when failure is detected because a negative counterexample is received or a $(k+1)$st positive counterexample is received.

Any negative counterexample stops and restarts the algorithm. Since in Generate-Candidates at most $O(n^{2r})$ candidates are generated and we need to choose one of these correctly at each of $k+1$ possible steps, there may be up to $n^{2r(k+1)}$ runs of the deterministic algorithm. Thus the overall time and query complexity is $O(n^{rk+r+k} \cdot n^{2r(k+1)}) = O(n^{3rk+3r+k})$ which is polynomial since r and k are constant. □

The proof of Theorem 3 can be extended to obtain the following result.

Corollary 4 *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 IBQ (or false-positive-only UBQ model) with a boundary region of radius r (for constant r). The time and query complexity is $O(n^{9r+2})$.*

Proof: 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 contains all of the variables in 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. The time and query complexities directly follow from Theorem 3 by just setting $k = 2$. □

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.

Acknowledgments

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

References

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

[2] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] A. Blum. Separating distribution-free and mistake-bound learning models over the Boolean domain. *SIAM J. Comput.*, 23(5):990–1000, October 1994.

[9] 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.

[10] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis Dimension. *J. ACM*, 36(4):929–965, 1989.

[11] M. Frazier, S. Goldman, N. Mishra, and L. Pitt. Learning from a consistently ignorant teacher. *J. of Comput. Syst. Sci.*, 52(3):472–492, June 1996.

[12] 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.

[13] S. Goldman and R. Sloan. Can PAC learning algorithms tolerate random noise. *Algorithmica*, 14(1):70–84, July 1995.

[14] 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.

[15] 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.

[16] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. *Inform. Comput.*, 95(2):129–161, December 1991.

[17] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting {0,1} functions on randomly drawn points. *Inform. Comput.*, 115(2):284–293, 1994.

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

[19] 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.

[20] M. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *J. ACM*, 41(1):67-95, 1994.

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

[22] 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.

[23] 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.

[24] L. Pitt and M. K. Warmuth. Prediction preserving reducibility. *J. of Comput. Syst. Sci.*, 41(3):430–467, December 1990. Special issue for the *Third Annual Conference of Structure in Complexity Theory* (Washington, DC., June 1988).

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

[26] 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.

[27] R. Sloan. Types of noise in data for concept learning. *Inform. Proc. Lett.*, 54:157–162, 1995.

[28] 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.

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

[30] P. M. Vaidya. Speeding-Up Linear Programming Using Fast Matrix Multiplication. In *Proc. of the 30th Symposium on the Foundations of Comp. Sci.*, pages 332–337. IEEE Computer Society Press, Research Triangle Park, NC, 1989.