On Learning Discretized Geometric Concepts

– Extended Abstract –

Nader H. Bshouty* Zhixiang Chen† Steve Homer[1]

Abstract

We present a polynomial time online learning algorithm that learns any discretized geometric concept generated from any number of halfspaces with any number of known (to the learner) slopes in a constant dimensional space. In particular, our algorithm learns (from equivalence queries only) unions of discretized axis-parallel rectangles in a constant dimensional space in polynomial time. The algorithm also runs in polynomial time in l if the teacher lies on l counterexamples.

We then show a PAC-learning algorithm for the above discretized geometric concept when the example oracle lies on the labels of the examples with a fixed probability $p \leq \frac{1}{2} - \frac{1}{r}$ that runs in polynomial time also with r.

We use these methods, as well as a bounded version of the finite injury priority method, to construct algorithms for learning several classes of rectangles. In particular we design efficient algorithms for learning several classes of unions of discretized axis-parallel rectangles in either arbitrary dimensional spaces or constant dimensional spaces.

*Department of Computer Science, The University of Calgary, Calgary, Alberta, Canada T2N 1N4. Email: bshouty@cpsc.ucalgary.ca

†Department of Computer Science, Boston University, Boston, MA 02215. Email: zchen@cs.bu.edu. The author was supported by NSF grant CCR91-03055 and by a Boston University Presidential Graduate Fellowship.

[1]Department of Computer Science, Boston University, Boston, MA 02215. Email: homer@cs.bu.edu. The author was supported by NSF grant CCR91-03055.

1 Introduction

A discretized geometric concept G is a set of integer points $G \subseteq \mathcal{N}_n^d$ where $\mathcal{N}_n = \{0, \ldots, n-1\}$. In this paper we consider discretized geometric concepts whose boundaries lie on hyperplanes.

The first learning model we consider in this paper is online learning [A88,L88], i.e., exact learning from equivalence queries only, and it is different from the PAC-learning in that the final output of the learning algorithm must be exactly the concept being learned rather than some approximation of the concept. The equivalence oracle receives from the learner a hypothesis $H \subseteq \mathcal{N}_n^d$ and returns either "YES" indicating that H is (exactly) equal to the target concept G or "NO" with a counterexample $a \in H \Delta G$. It is known from [A88] that online learning implies PAC-learning under any distribution. Online learning with t counterexamples also guarantees PAC-learning that learns exactly the target concept after t counterexamples. Blum in [B90] gave some evidence that online learning is harder than PAC-learning.

We also consider PAC-learning under any distribution when the example oracle lies with a fixed probability p. Obviously, if $p = 1/2$ then the example oracle gives random values for each point and no learning is possible. We show that PAC-learning is possible if p is polynomially close to $1/2$.

The discretized geometric concepts considered in this paper are those that have their boundaries on hyperplanes $\sum_{j=1}^{d} a_{i,j} x_j = b_{i,j}$ $i = 1, \ldots, m$. The possible slopes of those hyperplanes, i.e., $a_i = (a_{i,1}, \ldots, a_{i,d})$, are known to the learner, but the same slope with different shifts b can be used for many hyperplanes in G. To formalize this, let $S \subset \mathcal{Z}^d$ where $\mathcal{Z}$ is the set of integer numbers. Let $z = \|S\|$ denote the size of S (the sum of the logarithm of the absolute values of the integers in S). Let G be a geometric concept with its boundary on m hyperplanes in $\mathcal{N}_n^d$ with slopes from S. The main results of this paper are: For a constant dimension d,

1. The geometric concept G is online learnable in

$poly(l, m, z, \log n)$ time and queries if the equivalence oracle lies on l examples.

2. The geometric concept G is PAC-learnable in $poly(r, m, z, \log n)$ time when the example oracle lies on the labels of the examples with a fixed probability $p \leq 1/2 - 1/r$.

3. Polynomial time algorithms exist for learning (a) unions of rectangles[1], (b) unions of $k \geq 3$ rectangles whose projections at some dimension are pairwise-disjoint by unions of $(k-1)(\log n + 1)$ rectangles, and (c) unions of two disjoint rectangles by unions of two rectangles. Results of (b) and (c) hold for arbitrary dimensional spaces.

There are two substantially different algorithms for result $3(a)$. One is as a consequence of result 1 by choosing the slopes $S = \{e_i\}$ as the standard basis in the algorithm for result 1. Another algorithm that learns unions of k rectangles by unions of $O((k \log n)^d)$ rectangles, as well as algorithms for results $3(b)$ and $3(c)$, are designed with a new and very different technique, the bounded injury priority method, which was developed specifically for learning rectangles. By making the analogy with priority arguments from recursion theory explicit in our algorithm designs for on-line learning unions of rectangles, we can obtain a more precise analysis of methods present in earlier work in this area. And as well, the priority method provides a more general and canonical construction of learning algorithms which can be used for new cases of learning of rectangles which were not possible before. The methods developed here are far from the full strength of finite injury arguments used in modern recursion theory. They make essential and strong use of the notion of requirements and of their (bounded) injury, but only little use of priority assigned to these requirements. Nonetheless they provide a new and useful method with which to explain several complicated constructions and proofs.

Result $3(a)$ is a generalization of one of the algorithms in [GGM94] that online learns unions of rectangles with *membership queries* and the algorithm in [C93] that online learns unions of *two* rectangles in the *2*-dimensional space. See also [MT90,MT91] and [MT92]. Another generalization of it is online learning polynomial size decision trees over the basis "Is $x_i \succ d$" where $\succ \in \{>, <, \geq, \leq\}$ (in the nodes) in the constant dimensional space. If the space is the plane $\mathcal{N}_n^2$ and $S = \{(0,1), (1,0), (1,1), (1,-1), (1,2), (2,1), (1,-2), (2,-1)\}$ then our algorithm for result 1 in particular can learn the geometric concepts that generated from lines that makes the angles 0; 30; 45; 60; 90; 110; 135 and 150 with the x-axis in polynomial time. In the three dimensional space the algorithm can learn in polynomial time a union of any number of regular polyhedra when their boundary's slopes are known.

In the literature PAC-learning of restricted (nondiscretized) geometric concepts is also studied. PAC-learning a union of a constant number of (nondiscretized) rectangles in any dimensional space and a union of any number rectangles in the constant dimensional space is studied in [BEHW89] and [LW90]. Our algorithm for result 1 also PAC-learns nondiscretized geometric concepts under any distribution.

The paper is organized as follows. In section 2 we give some preliminary results in geometry. In section 3 we describe our learning models. In section 4 we give the online algorithm for result 1 and in section 5 we prove the correctness of the algorithm[2]. In section 6 we present the PAC-learning algorithm for result 2 and prove its correctness. In section 7 we describe the bounded injury priority method. Technical results about the method and three efficient learning algorithms using it will be given in section 8 (see [CHb94] for details).

2 Preliminaries

Let $\mathcal{N}, \mathcal{Z}$ and $\mathcal{R}$ be the set of nonnegative integers, integers and reals, respectively. Let $\mathcal{N}_n = \{0, \ldots, n-1\}$ and $S \subseteq \mathcal{Z}^d$ be a set of slopes. A d-dimensional *hyperplane* is $\sum_{i=1}^d a_i x_i = b$ for some $a_i, b \in \mathcal{Z}$ (when we write $\sum_{i=1}^d a_i x_i = b$ we mean $\{x | \sum_{i=1}^d a_i x_i = b\}$). A *halfspace* is $\sum_{i=1}^d a_i x_i \succ b$ where $\succ \in \{>, \geq, <, \leq\}$. A *discretized halfspace* is $H \cap \mathcal{N}_n^d$ for some halfspace H. A *geometric concept generated from hyperplanes* with slopes from $S \subseteq \mathcal{Z}^d$ is a set $\hat{G} \subseteq \mathcal{R}^d$ that its boundary $\partial \hat{G}$ is the union of connected components that are on hyperplanes with slopes from S. A *discretized geometric concept generated from hyperplanes* with slopes from $S \subseteq \mathcal{Z}^d$ is $\hat{G} \cap \mathcal{N}_n^d$ where $\hat{G}$ is a geometric concept generated from hyperplanes with slopes from S.

The *complexity* $C_S(\hat{G})$ of a geometric concept $\hat{G}$ is the minimal number of hyperplanes with slopes from

[1]In work independent from ours, Maass and Warmuth [MW94] also proved this result using hypotheses different from unions of rectangles and achieving, as well, an optimal number of equivalence queries.

[2]The results of [GGM94] as well as more details for the results in sections 4 and 5 are given in [BGGM94].

S that their union contains the boundary of $\hat{G}$. The complexity $C_S(G)$ of a discretized geometric concept G is the minimal $C_S(\hat{G})$ over all geometric concept $\hat{G}$ that satisfies $G = \hat{G} \cap \mathcal{N}_n^d$. It is obvious that any exact learning algorithm for a discretized geometric concept G cannot run in time less than the complexity $C_S(G)$ (information theory bound). Also the complexity of learning one point in $\mathcal{N}_n^d$ is at least $\log n$. In our algorithms the input will be S. The size $\|S\|$ of S is the number of bits we need to represent the elements of S (which is the sum of the logarithm of the absolute values of all the entries of the slopes in S). Therefore we say that a learning algorithm runs in polynomial time if the time is $poly(\|S\|, C_S(G), \log n)$.

Let $S = \{a_1, \ldots, a_s\} \subset \mathcal{Z}^d$ be a set of slopes and $B = \{B_1, \ldots, B_s\}$ where $B_i \subseteq \mathcal{R}$. A *grid* $\hat{G}(S, B)$ is the set of connected components in $\mathcal{R}^d$ generated from the hyperplanes $a_i x^T = b_{i,j}$ where $x = (x_1, \ldots, x_d)$, $i = 1, \ldots, s$ and $b_{i,j} \in B_i$. Each connected component is of the form

$$C_{j_1,\ldots,j_s} = \bigcap_{i=1}^{s} \{x \,|b_{i,j_i-1} \le a_i x^T < b_{i,j_i}\}$$

for

$$0 \le j_w \le |B_i| + 1, \quad w = 1, \ldots, s.$$

where $B_i = \{b_{i,1}, \ldots, b_{i,l_i}\}$, $l_i = |B_i|$ and $-\infty = b_{i,0} < b_{i,1} < \cdots < b_{i,l_i} < b_{i,l_i+1} = \infty$.

Let $E \subseteq \mathcal{N}_n^d \times \{0, 1\}$ be a set of labeled examples (x, y). We say that E is *consistent* with $G(S, B)$ if for every two examples (x_1, y_1) and (x_2, y_2) if x_1 and x_2 are in the same component in $G(S, B)$ then $y_1 = y_2$. For a grid $G(S, B)$ and a consistent set E we define the boolean function $H(G, E) : \mathcal{N}_n^d \to \{0, 1\}$ as follows:

$$H(G,E)(a) = \begin{cases} y & \text{if } \exists (x,y) \in E,\ x \text{ and } a \text{ are} \\ & \text{in the same component} \\ 0 & \text{otherwise} \end{cases}$$

This hypothesis is consistent with the examples in E.

We will denote by $\mathcal{Z}_n$ the set $\{0, \mp 1, \ldots, \mp(n-1)\}$.

In the paper we will use the following lemma

Lemma 1. Let $S = \{a_1, \ldots, a_s\} \subset Z^d$ be a set of slopes and $B = \{B_1, \ldots, B_s\}$ where $B_i \subset Z$ and $|B_i| \le m$ for all $i = 1, \ldots, s$. Then the number of connected components of $\hat{G}(S, B)$ is at most $(ms)^d + 1$.

Proof. We show a more general result: Any t d-dimensional hyperplanes in a $d+1$-dimensional hyperplane H divides it into at most $t^d + 1$ connected components. The proof is by induction on d. For $d = 0$, t points in a line divide the line into $t + 1$ connected components. For any d we have: Let $H_1, \ldots, H_t$ be d-dimensional hyperplanes. If $H_1, \ldots, H_t$ are parallel then they divide H to $t + 1 \le t^d + 1$ connected components.

Suppose $H_1, \ldots, H_t$ are not parallel. Then each connected component in H is bounded by at least two hyperplanes in $H_1, \ldots, H_t$. For each H_i let $R_i = \{H_i \cap H_j | H_i \cap H_j \ne \emptyset, j \ne i\}$ be $d - 1$ dimensional hyperplanes in H_i. Since $|R_i| \le t - 1$, by the induction hypothesis the hyperplanes in R_i intersect H_i in at most $(t - 1)^{d-1} + 1$ connected components. Each connected component corresponds to at most two connected components generated by $H_1, \ldots, H_t$ in H (one from each side). Therefore, the number of connected components generated by $H_1, \ldots, H_t$ is at most twice the number of connected components in $\cup_{i=1}^t R_i$. Since the number of connected components in $\cup_{i=1}^t R_i$ is at most $((t - 1)^{d-1} + 1)t$ and since each connected component generated by $H_1, \ldots, H_t$ is bounded by at least two hyperplanes, the number of connected components generated by $H_1, \ldots, H_t$ is at most

$$((t - 1)^{d-1} + 1)t \le t^d + 1.$$

3 The Learning Models

A *domain* is a set of elements. We will call the elements of a domain *points*. Given a *domain* X. A *concept* is $G \subseteq X$. A *class of concepts* $\mathcal{G}$ is $\mathcal{G} \subseteq 2^X$. The elements of $\mathcal{G}$ are represented in some representation. The *size* of $G \in \mathcal{G}$, *size*(G), is the number of bits needed to represent G in this representation.

The first learning model we consider here is the *online learning model*, [A88, L88]. In the online learning a teacher has a *target concept* G that is from a class of concepts $\mathcal{G}$. The learner wants to exactly learn G in polynomial time using polynomial number of *equivalence* queries. In the equivalence queries the learner gives the teacher a hypothesis H and the teacher answers "YES" if $H = G$ or "NO" with a counterexample (CE) $x \in H \Delta G$ if $H \ne G$. If $x \in (H - G)$ then x is called a negative counterexample (NCE) or a positive counterexample (PCE). Polynomial time means polynomial in $\log |X|$ and the size of the target concept.

A *l-liar* teacher is a teacher that is allowed to lie at most l times in the equivalence queries.

In the *online learning with membership queries* model the teacher also answers *membership* queries. In the membership queries the learner gives the teacher a point x and the teacher answers "YES" if $x \in G$ and "NO" if $x \notin G$.

In the *PAC-learning model* the teacher gives labeled examples $(x, G[x])$ where $G[x] =$"YES" if $x \in G$ and $G[x] =$"NO" if $x \notin G$. The labeled examples are given according to an unknown (to the learner) fixed distribution D over the points in the domain. The goal of the learner is to find a hypothesis H that, with probability at least $1/2$, ϵ-approximate G with respect to the distribution D. That is, the learner should output a hypothesis $H \subset X$ such that

$$Pr\left[\sum_{x\in H\Delta G} D[x] \le \epsilon\right] \ge \frac{1}{2}.$$

The algorithm must run in time polynomial in the size of the target concept $\log|X|$ and $1/\epsilon$. The probability $1/2$ can be replaced with any probability $1/poly(size(G), \log|X|, 1/\epsilon)$. Such an algorithm can be changed to a PAC-learning algorithm that runs in time polynomial in $size(G), \log|X|, 1/\epsilon$ and $\log(1/\delta)$ and with probability at least $1-\delta$ output an ϵ-approximation of G.

A *probability p-liar* teacher will lie on the labels of the examples $G[x]$ with probability p. A 1/2-liar teacher is a teacher that gives examples with random labels for each point.

4 The Online Learning Algorithm

In this section we present the online algorithm. The hypotheses we used here, as introduced in [GGM94], is obtained by dividing the domain into regions (or connected components) based on a set of hyperplanes selected that cut through the whole domain. However, we modify their basic technique by replacing the membership queries, used as part of a binary search to find a hyperplane to add to the hypothesis that defines a side of the target, with the introduction of additional hyperplanes selected using equivalence queries as decribed below.

The idea of the algorithm is simple. The algorithm starts with a positive and a negative counterexample a and c, respectively. (We ask equivalence queries with $\emptyset$ and $\mathcal{N}_n^d$.) Since a is positive and c is negative the straight line between a and c must intersect one of the hyperplanes. Since we cannot ask membership queries we cannot tell where is the intersection and even if we can tell we cannot know the slope of the hyperplane that pass through that intersection. Our algorithm will assume that the intersection is in the point $(a+c)/2$ and will assume that all the hyperplanes with all the slopes pass through this point. This generous assumption turns to be not "very dramatic". We split the space in the point $(a+c)/2$ using hyperplanes with all possible slopes. Then we recursively run the algorithm for each region.

```
A set of slopes $S = \{a_1, \ldots, a_s\}$
OnlineLearn($S$)
1) $B_i \leftarrow \emptyset, i = 1, \ldots, s; E \leftarrow \emptyset; H \leftarrow 0$;
2) EQ($H$) $\rightarrow a$; If the answer is "YES"
       then output $H$;
3) $E \leftarrow E \cup \{(a, \overline{H(a)})\}$;
4) If there is $(c, y) \in E$ such that $a$ and $c$ are
      in the same component then
        $u \leftarrow (a+c)/2$;
        for $i = 1$ to $s$ do $B_i \leftarrow B_i \bigcup\{a_i u^T\}$;
5) $H \leftarrow H(G(S, B), E)$;
6) Goto 2;
```

In the algorithm S is the set of the possible slopes of the hyperplanes. We first assume that the geometric concept is empty ($B_i \leftarrow \emptyset, H \leftarrow 0$). The target concept G is regarded as a boolean function f_G where $G = \{x | f_G(x) = 1\}$. Therefore, the hypothesis in the algorithm is a boolean function $H : \mathcal{N}_n^d \rightarrow \{0, 1\}$. We ask equivalence query with H and add the counterexample a to E. If this counterexample is the first counterexample in the component it belongs to (in $G(S, B)$) then we just update $G(S, B)$. If this counterexample is in some component that already contains a point in E then a and c have different values in G and the line that pass through the two points must intersect a hyperplane in G. We then define $u = (a+c)/2$ and add all possible hyperplanes that pass through u to the hypothesis H.

5 Correctness and the Complexity of the Online Algorithm

In this section we prove the correctness of the algorithm and prove the following:

Theorem. Let G be a discretized geometric concept in $\mathcal{N}_n^d$ where d is constant. Let S be a set of slopes. There is an online algorithm that learns G in time $poly(\|S\|, C_S(G), \log n)$.

5.1 Correctness of the Algorithm

Since the size of S is $\|S\|$ we have $S = \{a_1, \ldots, a_s\} \subseteq \mathcal{Z}^d_{2^{\|S\|}}$. Let $d_i x^T = b_i$, $i = 1, \ldots, m =$

$C_s(G)$, $d_i \in S$, be a minimal set of hyperplanes that generates the boundary of the geometric concept G. Let $f_G : \mathcal{N}_n^d \to \{0,1\}$ be the boolean function of G, i.e., $G = \{x | f_G(x) = 1\}$. Since all the hyperplanes $d_i x^T = b_i$ pass through $\mathcal{N}_n^d$ there are $x_i \in \mathcal{N}_n^d$ such that $d_i x_i^T \geq b_i$. Therefore

$$|b_i| \leq |d_i x_i^T| \leq \gamma = dn2^{\|S\|}.$$

Let $B_i^{(\delta)}, E^{(\delta)}$ and $H^{(\delta)}$ be the variables of the algorithm at the δ-th iteration. For each hyperplane $d_i x^T = b_i$ let $z_i^{(\delta)} \leq b_i < y_i^{(\delta)}$ be the closest two points in $B_j^{(\delta)} \cup \{-\gamma, \gamma\}$ to b_i where $d_i = a_j$ and $\gamma = dn2^{\|S\|}$. Consider $\Delta_{i,1}^{(\delta)} = y_i^{(\delta)} - b_i$ and $\Delta_{i,2}^{(\delta)} = b_i - z_i^{(\delta)}$. Since $B_1^{(1)} = \cdots = B_m^{(1)} = \emptyset$ and since $|b_i| \leq dn2^{\|S\|}$ we have

$$\Delta_{i,1}^{(1)} \leq \gamma, \qquad \Delta_{i,2}^{(1)} \leq \gamma \tag{1}$$

for all $i = 1, \ldots, m$.

Suppose a is a counterexample to $H^{(\delta)}$ and c and a are in the same component where $(c, y) \in E$. Then $f_G(a) \neq f_G(c)$. Therefore the line that pass through the two points a and c must intersect some hyperplane $d_{i_0} x = b_{i_0}$. Since a and c are in the same component we must have

$$z_{i_0}^{(\delta)} \leq d_{i_0} c^T \leq b_{i_0} \leq d_{i_0} a^T < y_{i_0}^{(\delta)},$$

or

$$z_{i_0}^{(\delta)} \leq d_{i_0} a^T \leq b_{i_0} \leq d_{i_0} c^T < y_{i_0}^{(\delta)}.$$

Suppose we have the former case. Now at the $(\delta+1)$-iteration we have two cases: if $d_{i_0}^{(\delta)} u^T > b_{i_0}$ where $u = (a+c)/2$ then $z_{i_0}^{(\delta+1)} = z_{i_0}^{(\delta)}$ and $y_{i_0}^{(\delta+1)} = d_{i_0} u^T$ and then

$$\Delta_{i_0,2}^{(\delta+1)} = b_{i_0} - z_{i_0}^{(\delta+1)} = b_{i_0} - z_{i_0}^{(\delta)} = \Delta_{i_0,2}^{(\delta)}$$

and

$$\begin{aligned}\Delta_{i_0,1}^{(\delta+1)} = y_{i_0}^{(\delta+1)} - b_{i_0} &= d_{i_0} u^T - b_{i_0} \\ &= \frac{d_{i_0} a^T}{2} + \frac{d_{i_0} c^T}{2} - b_{i_0} \\ &\leq \frac{y_{i_0}^{(\delta)} - b_{i_0}}{2} + \frac{d_{i_0} c^T - b_{i_0}}{2} \\ &\leq \frac{y_{i_0}^{(\delta)} - b_{i_0}}{2} = \frac{\Delta_{i,1}^{(\delta)}}{2}.\end{aligned}$$

The second case is when $d_{i_0} u^T \leq b_{i_0}$, and then

$$\Delta_{i_0,1}^{(\delta+1)} = \Delta_{i_0,1}^{(\delta)}, \qquad \Delta_{i_0,2}^{(\delta+1)} = \frac{\Delta_{i_0,2}^{(\delta)}}{2}.$$

This proves that at any iteration δ if a and c are in the same component then there is i_0 such that

$$\Delta_{i_0,1}^{(\delta+1)} = \frac{\Delta_{i_0,1}^{(\delta)}}{2} \quad \text{or} \quad \Delta_{i_0,2}^{(\delta+1)} = \frac{\Delta_{i_0,2}^{(\delta)}}{2}. \tag{2}$$

Now the hyperplane $d_{i_0} x^T = b_{i_0}$ can be changed to any hyperplane $d_{i_0} x^T = b_{i_0} + \tau$, $\tau < 1$ or $d_{i_0} x^T = b_{i_0} - \tau$, $\tau < 1$ (depending on whether G also contains the point on $d_{i_0} x^T = b_{i_0}$ or not). Therefore, when $\Delta_{i_0,1}^{(\delta)} < 1$ and $\Delta_{i_0,2}^{(\delta)} < 1$ then the algorithm has determined the hyperplane $d_{i_0} x^T = b_{i_0}$ and no other hyperplanes will be added for $d_{i_0} x^T = b_{i_0}$.

Now we will show that if $\Delta_{i_0,2}^{(\delta)} < 1$ and $\Delta_{i_0,1}^{(\delta)} > 1$ then $\Delta_{i_0,2}^{(\delta)}$ will stop changing. If $\Delta_{i_0,2}^{(\delta)} < 1$ then since

$$d_{i_0} c^T - b_{i_0} \leq \Delta_{i_0,2}^{(\delta)} < 1$$

we must have $d_{i_0} c^T = b_{i_0}$. Then

$$d_{i_0} u^T = \frac{d_{i_0} c^T + d_{i_0} a^T}{2} \geq d_{i_0} c^T = b_{i_0}$$

and therefore $\Delta_{i_0,1}^{(\delta)}$ will be changed.

Therefore, by (1) and (2) the number of iterations that change $\Delta_{i,j}^{(\delta)}$ is finite and the number of iterations that do not change $\Delta_{i,j}^{(\delta)}$ is bounded by the number of components of the hypothesis which is also finite. Therefore the algorithm must stop with the target concept.

5.2 The Complexity of the Algorithm

By (1) and (2) the number of iterations in the algorithm to find one hyperplane is

$$\lceil 2\log(\gamma)) \rceil = O(\|S\| + \log n).$$

The number of hyperplanes is $m = C_S(G)$ and at each iteration we add $s = |S| \leq \|S\|$ hyperplanes to the hypothesis. Therefore the number of hyperplanes generated in the algorithm is $O(C_S(G)\|S\|(\|S\| + \log n))$. By lemma 1 the number of connected components of the hypothesis is at most

$$O((C_S(G)\|S\|(\|S\| + \log n))^d).$$

The number of iterations that do not change $\Delta_{i,j}^{(\delta)}$ is bounded by the number of connected components. Since by lemma 1 this number does not exceed $O((C_S(G)\|S\|(\|S\| + \log n))^d)$, the complexity of the above algorithm is

$$O((C_S(G)\|S\|(\|S\| + \log n))^d).$$

5.3 *l*-liar teacher

If the equivalence oracle lies on l counterexamples then we add to the set of slopes S the elementary vectors $\{e_i\}$. Then those faulty points will be bounded by hyperplanes and at the end the oracle must give the real value of those faulty points. This changes $C_S(G)$ to $C_S(G)+dl$, $\|S\|$ to $\|S\|+d$ and the complexity to

$$O(((C_S(G)+dl)(\|S\|+d)(\|S\|+d+\log n))^d).$$

6 The PAC-learning Algorithm

In this section we sketch the PAC-learning algorithm when the example oracle lies on the labels of the examples with a fixed probability $p \le \frac{1}{2}-\frac{1}{r}$.

Let C_i be t disjoint components of a geometric concept G and $C_i^+ \subseteq C_i$ be the set of all positive points in C_i. If each component is learned with approximation ϵ and probability greater than $1-\delta/t$ then with probability greater than $1-\delta$ we get an ϵ-approximation of $\cup_{i=1}^t C_i^+$. Formally, if we learn $H_i \subseteq C_i$ such that $Pr_D[x \in H_i \Delta C_i^+ | x \in C_i] \le \epsilon$ then since $H_i \subset C_i$ are disjoint

$$\begin{aligned} Pr_D\left[x \in \left(\bigcup_{i=1}^t H_i\right) \Delta \left(\bigcup_{i=1}^t C_i^+\right) \middle| x \in \bigcup_{i=1}^t C_i\right] &= \\ \sum_{i=1}^t Pr_D\left[x \in H_i \Delta C_i^+ \middle| x \in \bigcup_{i=1}^t C_i\right] &= \\ \sum_{i=1}^t \left(Pr_D\left[x \in H_i \Delta C_i^+ | x \in C_i\right] \times\right. & \\ \left. Pr_D\left[x \in C_i \middle| x \in \bigcup_{i=1}^t C_i\right]\right) &\le \\ \epsilon \sum_{i=1}^t Pr_D\left[x \in C_i \middle| x \in \bigcup_{i=1}^t C_i\right] &= \epsilon. \end{aligned}$$

Let $C \subseteq \mathcal{N}_n^d$ be a component in some hypothesis H of the target geometric concept G. Let C^+ (resp. C^-) be the set of all positive points (resp. negative points) in C for G. Suppose D is a distribution on $\mathcal{N}_n^d$. If the example oracle EX lies with a fix probability p then

$$Pr_D[EX(x)=1|x \in C] = pPr_D[x \in C^-|x \in C] +$$
$$(1-p)Pr_D[x \in C^+|x \in C].$$

This implies

$$Pr_D[x \in C^+|x \in C] = \frac{Pr_D[EX(x)=1|x \in C]-p}{1-2p}.$$

Suppose the learner knows p. If the EX oracle gives "enough" examples from C then using Chernoff bound the learner can estimate $Pr_D[EX(x)=1|x \in C]$ up to error $\epsilon' = 2\epsilon/r$ with high probability (exponentially close to 1). Now since $1-2p \ge \frac{2}{r}$ this gives $\epsilon/2$ approximation of $q = Pr_D[x \in C^+|x \in C]$. If $q \ge 1-\epsilon$ then we can assume that all the points in C in the target are positive and if $q \le \epsilon$ then we can assume that all the points in C in the target are negative. Otherwise, we have

$$Pr_D[x \in C^+|x \in C] \ge \epsilon \quad \text{or} \quad Pr_D[x \in C^-|x \in C] \ge \epsilon.$$

In this case we randomly choose one positive point a and one negative point c and make the split as in the online algorithm. We call this split "success" if a is really a positive point and c is really a negative point. The probability of success is

$$Pr[\text{ success}] =$$

$$Pr[a \text{ is pos. \& } b \text{ is neg.}] = Pr[a \text{ is pos.}]Pr[b \text{ is neg.}].$$

and

$$\begin{aligned} Pr[a \text{ is pos.}] &= \\ Pr_D[x \in C^+|EX(x)=1 \wedge x \in C] &= \\ \frac{Pr_D[EX(x)=1|x \in C^+]Pr_D[x \in C^+|x \in C]}{Pr_D[EX(x)=1|x \in C]} &= \\ \frac{(1-p)Pr_D[x \in C^+|x \in C]}{(1-2p)Pr_D[x \in C^+|x \in C]+p} &= \\ \frac{(1-p)}{(1-2p)+\frac{p}{Pr_D[x \in C^+|x \in C]}} &\ge \\ \frac{1}{2\left(1+\frac{1}{Pr_D[x \in C^+|x \in C]}\right)} \ge \frac{Pr_D[x \in C^+|x \in C]}{4}&. \end{aligned}$$

Therefore,

$$\begin{aligned} Pr[\text{success}] &\ge \frac{Pr_D[x \in C^+|x \in C]}{4}\frac{Pr_D[x \in C^-|x \in C]}{4} \\ &\ge \frac{\epsilon}{32}. \end{aligned}$$

This shows that the average number of hyperplanes generated in the hypothesis is $32/\epsilon$ times the number of the hyperplanes generated by the online algorithm.

If the learner cannot get enough points in C then the total distribution of the component C is "small" and we can assign random value to C. Therefore, the algorithm generates in average $O(msd(1/\epsilon)\log n)^d$ connected components.

Now what happen if the learner does not know p. First, it can be shown that if the learner uses p'

"enough" close to p then the algorithm can still find a good approximation of the target formula. If the learner uses p' that is not close to p, say $|p - p'| > \eta$ then when $Pr_D[x \in C^+ | x \in C] = 1$ (or close to 1) then $Pr_D[EX(x) = 1 | x \in C] = 1 - p$ and then

$$\begin{aligned}\frac{Pr_D[EX(x) = 1 | x \in C] - p'}{1 - 2p'} &= \frac{1 - p - p'}{1 - 2p'} \\ &= 1 - \frac{p - p'}{1 - 2p'} \\ &= \begin{cases} 1 - \frac{\eta}{1-2p'} & p > p' \\ 1 + \frac{\eta}{1-2p'} & p < p' \end{cases}\end{aligned}$$

So the algorithm will run forever when $p' < p$ and will give probability not from $[0, 1]$ if $p' > p$. Therefore, a good approximation of p can be found using a binary search.

7 The Bounded Injury Priority Method

In a typical finite injury priority argument (see, Soare [S]), one needs to achieve a "goal". One divides this "goal" A into a sequence of infinitely many "requirements" $\{R_i\}_{i \in N}$ such that A is achieved if all the requirements R_i are met or satisfied. Normally, one also assigns priority to the requirements. If $n < m$, then the requirement R_n is assigned priority over R_m, and we say that R_n has higher priority than R_m. One then constructs a procedure which runs in stages. At each stage, one will take certain action to satisfy some requirement(s). However, actions taken at some stage s for satisfying R_m may at a later stage $t > s$ be undone when action is taken for satisfying R_n. In this case, we say that R_m is injured at stage t. The crucial feature of all finite injury priority methods is that each requirement is injured finitely often and so comes to a limit.

Consider designing a learning algorithm for a given concept class $\mathbf{C}$ over a domain $\mathbf{X}$. For each target concept $C_t \in \mathbf{C}$, the goal of the learner is to identify C_t. One can design a sequence of requirements $\{R_i\}$ such that C_t is learned if all the requirements R_i are satisfied. However, substantial difficulties exist in this approach. First, one would like the complexity of the algorithm A to be bounded by a reasonable function $f(n, |C_t|, m)$ (usually, f is a polynomial, or at worst exponential), where n is the size of the domain, $|C_t|$ is the size of the target concept, and m is the size of the largest CE's received by the learner. Thus, the number of requirements designed by the learner must be bounded by $f(n, |C_t|, m)$. Similarly, the number of injuries received by each requirements must also be bounded by $f(n, |C_t|, m)$. Second, in the finite injury priority method, the domain is infinite. Once a requirement is injured, one can search the domain to find a new element with which to *"remedy"* the injury, i.e., to satisfy the requirement again. However, in designing the learning algorithm A, the domain $\mathbf{X}$ is finite. Hence, once a requirement is injured, it is more difficult to remedy the injury within the domain $\mathbf{X}$. Finally, during the construction of a finite injury priority method, one can decide whether a requirement is injured at each stage, because one can simply trace the entire (but finite) construction up to the current stage. However, this may not be true in designing a learning algorithm because of the desired complexity bound on the algorithm.

8 The Design of Algorithms with the Bounded Injury Priority Method

In this section we begin with several basic results whose proofs exploit the ideas of an injury construction. The methods and techniques used indicate how the problems mentioned here can be dealt with by injury arguments in this setting. Furthermore, they form the basis for the more complex constructions in section 9 where priority is essentially needed.

$\forall i, j \in \mathcal{N}$, we use $[i, j]$ to denote the set $\{i, \ldots, j\}$ when $i \leq j$ or ϕ otherwise. Let $BOX_n^d = \{\prod_{i=1}^d [a_i, b_i] | 0 \leq a_i \leq b_i \leq n-1, \forall i \in [1, d]\} \cup \{\phi\}$. We consider the following concept class of unions of rectangles over the domain $X = [0, n-1]^d$

$$\bigcup_k BOX_n^d = \{C_1 \cup \cdots \cup C_k | i \in [1, k], C_i \in BOX_n^d\}.$$

Consider the following interval concept class

$$HEAD(n) = \{[0, j] | 0 \leq j \leq n - 1\}$$

over the domain $[0, n-1]$. We now design an algorithm S to learn any target concept $C_t = [0, j] \in HEAD(n)$. Here, we consider an extended environment which may responds with an NCE outside C_t (called a true NCE) or an NCE in C_t (called a false NCE) for a given hypothesis. Algorithm S runs in stages. At each stage s, S decides a value for j, denoted by $RS(s)$, and issues a hypothesis $H_s = [0, RS(s)]$. Let x_s be the counterexample received by the learner at stage s for the hypothesis H_s. Define

$$PS(s) = max(\{0\} \cup \{x_r | r \in [1, s], x_r \ is \ a \ PCE\}),$$

$NS(s) = min(\{n\} \cup \{x_r | r \in [1,s], x_r \ is \ an \ NCE$
$and \ x_r > PS(s)\})$.

In order to learn C_t, one only needs to satisfy the following requirement

$$R: \ \exists s \ \ (\forall s' \geq s \ \ (RS(s') = j)).$$

We say that R is *injured* (or receives an *injury*) at stage s, if $PS(s) < NS(s) \leq j$. When R receives an injury at stage s, the learner has received false NCE's and will be *fooled* by them until he proves that they are false by receiving a PCE greater than their maximum.

Learning Algorithm S

Stage 1. Set $RS(1) = 0$. Ask an equivalence query for the hypothesis $H_1 = [0, RS(1)]$. If yes then stop. Otherwise, one receives a PCE x_1.

Stage $s+1 \geq 2$. We consider three cases: (1) If x_s is an NCE and $x_s \leq PS(s)$, then set $RS(s+1) = RS(s)$. (2) If x_s is an NCE but $x_s > PS(s)$, then set $RS(s+1) = PS_s + \lfloor \frac{NS(s)-PS(s)}{2} \rfloor$. (3) If x_s is a PCE, then set $RS(s+1) = min(\{NS(s) - 1\} \cup \{RS(r) | 1 \leq r \leq s \ \& \ PS(s) \leq RS(r) < NS(s)\})$. Finally, one asks an equivalence query for the hypothesis $H_{s+1} = [1, RS(s+1)]$. If yes then stop, otherwise one receives a CE x_{s+1}.

For an NCE x_s, we say that x_s is an *invalid* NCE if $x_s \leq PS(s)$, otherwise we say that x_s is a *valid* NCE[3]. The following theorems are slightly improvements on two results in [CM92].

Theorem 8.1. *Assume that R has received i injuries and g invalid NCE's have been received during a learning process of S. Then, at most $\log n$ true NCE's and at most $2(\log n + g + i) + i + 1$ CE's occur in this learning process.*

Theorem 8.2. *Assume f false NCE's are received during a learning process for S. Then, requirement R will receive at most $3f$ injuries.*

We can design a learning algorithm S^* using the similar injury construction for the concept class $TAIL(n) = \{[j, n-1] | j \in [0, n-1]\}$ over the domain $X = [0, n-1]$ such that analogous versions of Theorem 8.1 and 8.2 hold for S^*. With direct transformation, we know that, $\forall a, b$, Algorithm S and S^* works on the concept classes $HEAD(a,b) = \{[a,j] | j \in [a,b]\}$ and $TAIL(a,b) = \{[j,b] | j \in [a,b]\}$, respectively. For convenience, we will use S and S^* (or with subscriptions in most cases) throughout this paper to stand respectively for copies of Algorithm S and S^*. We also use $RS(s), NS(s), PS(s)$ and, $RS^*(s), NS^*(s)$, and $PS^*(s)$ (again, with subscriptions in most cases) to denote the corresponding parameters as defined in the constructions of Algorithm S and S^*.

For any $S \subseteq [0, n-1]^d$, let $min_i(S) = min\{x_i | x \in S \ \& \ x = (x_1, \ldots, x_d)\}$, $max_i(S) = max\{x_i | x \in S \ \& \ x = (x_1, \ldots, x_d)\}$. Define $rec(S) = \prod_{i=1}^{d} [min_i(S), max_i(S)]$. Given $C_t \in \bigcup_k BOX_n^d$, for any example $r \in [0, n-1]^d$ and $S \subseteq [0, n-1]^d$, we say that (r, S) is a *witness* for C_t if and only if, (1) $r \notin C_t$ and $S \subseteq C_t$, and (2) $r \in rec(S)$. It is obvious that $\forall C_t \in \bigcup_k BOX_n^d$, $C_t \notin BOX_n^d$ if and only if there is a witness (r, S) for it.

Theorem 8.3. *Given $k \geq 2$. For any $C_t \in \bigcup_k BOX_n^d$, with $O(d^2 \log n)$ CE's one learns it if it is a rectangle or finds a witness otherwise.*

Proof Sketch. For any target concept $C_t \in \bigcup_k BOX_n^d$, we assume that it is a rectangle $\prod_{i=1}^{d} [a_i, b_i]$ (of course, this may not true). Then, $\forall i \in [1, d]$, at the i-th dimension we use two procedures S_i and S_i^* to search for the parameters b_i and a_i, respectively. Our learning algorithm LR runs in stages. At stage s, the learner issues the hypothesis

$$H(s) = \textstyle\prod_{i=1}^{d} [RS_i^*(s), RS_i(s)].$$

Let $x_s = (x_{s1}, \ldots, x_{sd})$ be the CE received for $H(s)$[4], $W(s)$ be the set of all CE's received by the end of stage s and, $P(s)$ be the set of all PCE's in $W(s)$. In order to learn C_t, one only needs to satisfy the following $2d$ requirements, $i \in [1, d]$,

$$R(i,1): \ \exists s \ (\forall s' \geq s \ (RS_i(s') = b_i)),$$
$$R(i,2): \ \exists s \ (\forall s' \geq s \ (RS_i^*(s') = a_i)),$$

We say that $R(i,1)$ is injured at stage s, if either $PS_i(s) < NS_i(s) \leq b_i$, or $b_i < PS_i(s)$. In the first case, we say that $R(i,1)$ receives a negative injury. In the latter case, we say that $R(i,1)$ receives a positive injury. Similarly, we define injury, positive injury and negative injury for requirement $R(i,2)$.

Learning Algorithm LR

Stage 0. Set $H(0) = \phi$. Ask an equivalence query. If yes then stop. Otherwise one receives a PCE x_0. Let $W(0) = \{x_0\}$.

Stage $s+1 \geq 1$. Decide whether $\exists r_s \in (W(s) - P(s))$ such that $(r_s, PS(s))$ is a witness. If yes

[3]Obviously, the extended environment cannot cheat the learner by giving an invalid NCE. In other words, Requirement R will never be injured by an invalid NCE. We consider invalid NCE's for the completeness of Algorithm S. In this paper, however, invalid NCE's can be technically ignored in all the applications of Algorithm S.

[4]One should note that, although x_s is a CE for the hypothesis H_s, x_{si} may not necessarily a CE for the hypothesis issued by procedure S_i or S_i^*, $i \in [1, d]$. If x_{si} is an NCE, it may be false. However, x_{si} will always true, if it is a PCE.

then output it and stop. Otherwise, on the received CE x_s. $\forall i \in [1,d]$, do: (1) Search for b_i using S_i in the domain $[x_{0i}, n]$, i.e., if x_{si} is a CE for the current hypothesis $RS_i(s)$ of S_i, then S_i issues a new hypothesis $RS(s+1)$, or sets $RS(s+1) = RS(s)$ otherwise. (2) Search for a_i using S_i^* in the domain $[1, x_{0i}]$ with the same manner as that in (1).

Ask an equivalence query for H_{s+1}. If yes then stop, otherwise a CE x_{s+1} is received. Set $W(s+1) = W(s) \cup \{x_{s+1}\}$.

Claim 8.4. *Assume that $C_t \in BOX_n^d$. Then, $\forall i \in [1,d]$, neither $R(i,1)$ nor $R(i,2)$ will receive a positive-injury. Furthermore, each of the procedures S_i and S_i^* receives at most $2(d-1)\log n$ invalid NCE's, and each of $R(i,1)$ and $R(i,2)$ receives at most $6(d-1)\log n$ negative injuries.*

With Claim 8.4 and Theorem 8.1 and 8.2, one can show the following: If $C_t \in BOX_n^d$, then with at most $4d\log n + 44d(d-1)\log n + 2d$ CE's the learner learns C_t. If $C_t \notin BOX_n^d$. then with at most $4d\log n + 44d(d-1)\log n + 2d + 1$ CE's, the learner will find a witness.

We now present three theorems which depend on the techniques and the proofs of Theorems 8.1, 8.2 and 8.3. Due to the limited space, we only give proof sketch for Theorem 8.7.

Theorem 8.5. *There is an algorithm for learning unions of two disjoint rectangles over the domain $[0, n-1]^d$ by unions of two rectangles with $O(d^4 \log^2 n)$ equivalence queries and in time $poly(d, \log n)$.*

Theorem 8.6. *There is an algorithm for learning unions of $k \geq 3$ rectangles over the domain $[0, n-1]^d$ whose projections on some dimension are pairwise-disjoint with $O(k^2 d^3 \log^3 n)$ equivalence queries and in time $poly(k, d, \log n)$, where the hypotheses are unions of at most $(k-1)(\log n + 1)$ rectangles.*

Theorem 8.7. *There is an algorithm for learning unions of k rectangles over the domain $[0, n-1]^d$ using $O((4kd\log n)^{d+1} d^2 \log n)$ equivalence queries and in time $poly(k^d, d^d, \log^d n)$, where the hypotheses are unions of at most rectangles $((4kd(\log n - 1) - 2k(\log n - 3))^d)$ rectangles.*

Proof Sketch of Theorem 8.7. Given $C_t = \cup_{i=1}^k C_i \in \bigcup_k BOX_n^d$. Let $C_i = \prod_{j=1}^d [a_{i,j}, b_{i,j}]$, $\forall i \in [1,k]$. Our learning algorithm LUR runs in stages. At stage s, $\forall i \in [1,d]$, we divide the i-th dimension into a set $I(i,s)$ of pairwise-disjoint intervals such that their union is $[0, n-1]$. Based on $I(i,s)$, we divide the domain $[0, n-1]^d$ into a set of pairwise-disjoint subdomains $D(s) = \prod_{i=1}^d I(i,s)$. Inside each $B \in D(s)$, $\forall i \in [1,d]$, at the i-th dimension, we use $S_{B,i}$ and $S_{B,i}^*$ to search for the right and left parameters $r_{B,i}$ and $l_{B,i}$ of $C_t \cap B$, respectively. Here, we assume that $C_t \cap B$ is a rectangle (again, this may not be true). At stage s, $\forall B \in D(s)$, we issue a hypothesis

$$H_B(s) = \prod_{i=1}^d [RS_{B,i}^*(s), RS_{B,i}(s)],$$

and the hypothesis issued for the target concept is

$$H(s) = \bigcup \{H_B(s) | B \in D(s)\}.$$

In order to learn C_t, we only need to satisfy the following requirements, $j \in [1,d]$,

$$R(0) : \exists s\ (\forall s' \geq s\ (\forall B \in D(s')\ (C_t \cap B \in BOX_n^d))),$$
$$R(j,1) : \exists s\ (\forall s' \geq s\ (\forall B \in D(s')\ (RS_{B,j}(s') = r_{B,j}))),$$
$$R(j,2) : \exists s\ (\forall s' \geq s\ (\forall B \in D(s')\ (RS_{B,j}^*(s') = l_{B,j}))),$$

The priority of the requirements is

$$R(0) > R(j_1, 1) = R(j_2, 2), \quad \forall j_1, j_2 \in [1,d].$$

We say that $R(0)$ is injured at stage s, if $\exists B \in D(s)$ $(C_t \cap B \notin BOX_n^d)$. $\forall (j,k) \in [1,d] \times [1,2]$, in the same manner as we did in the construction of Algorithm LR, we define injury, positive injury and negative injury for $R(j,k)$. Let $W(s)$ be the set of all the CE's received by the end of stage s, and $P(s)$ be the set of all PCE's in $W(s)$. Fix an integer-pairing function $<\cdot,\cdot>$.

Learning Algorithm LUR

Stage $<0,0>$. $\forall i \in [1,d]$, set $I(i, <0,0>) = \{[0, n-1]\}$. Let $H(<0,0>) = W(<0,0>) = \phi$. Go to stage $<1,0>$.

Stage $<s,m> > 0$. Ask an equivalence query for $H(<s-1,m>)$. If yes then stop, otherwise one receives a CE $x_{<s-1,m>}$. Set $Z_1 = W(<s-1,m>) \cup \{x_{<s-1,m>}\}$ and Z_2 be the set of all PCE's in Z_1. Decide whether $\exists B = \prod_{i=1}^d [e_i, f_i] \in D(<s-1,m>)$ such that, $\exists r_s \in (Z_1 - Z_2) \cap B$ such that $(r_s, Z_2 \cap B)$ is a witness. If yes then execute Case (i), or execute Case (ii).

Case (i). $\forall i \in [1,d]$, let $z_i = e_i + \lfloor \frac{f_i - e_i}{2} \rfloor$. If $e_i < f_i$ then set $I(i, <s,m>) = (I(i, <s-1,m>) - \{[e_i, f_i]\}) \cup \{[e_i, z_i], [z_i + 1, f_i]\}$, or set $I(i, <s,m>) = I(i, <s-1,m>)$. Set $W(<s,m>) = Z_1$. Undo all procedures by setting $H_E(<s,m>) = rec(E \cap P(<s,m>))$, $\forall E \in D(<s,m>)$. Go to stage $<s+1,m>$.

Case (ii). Within $B = \prod_{i=1}^d [e_i, f_i]$ with $x_{<s-1,m>} \in B$, in the same manner as one did in Algorithm LR, $\forall i \in [1,d]$, search for

$r_{B,i}$ and $l_{B,i}$ using $S_{B,i}$ and $S^*_{B,i}$, respectively. No actions will be taken for all the other subdomains in $D(<s-1,m>)$. Set $W(<s-1,m+1>) = Z_1$ and, $I(i,<s-1,m+1>) = I(i,<s-1,m>)$. Go to stage $<s,m+1>$.

Theorem 8.7 follows from Theorem 8.3 and the following claims

Claim 8.8. *During the learning process of Algorithm LUR, Case (i) can be executed at most* $2kd(\log n - 1)$ *times, and* $|I(i,<s,m>)| \leq 4k + 2k(\log n - 1) + 4k(d-1)(\log n - 1)$, $\forall s \geq 1, m \geq 1$.

Claim 8.9. *Assume that Case (i) was executed at stage* $<s,m'>$ *and* $<s+1,m''>$. *Then,* $m'' - m' = O((4kd\log n)^d d^2 \log n)$.

Claim 8.10. *Requirement* $R(0)$ *will be satisfied after receiving* $O((4kd\log n)^{d+1} d^2 \log n)$ *CE's.*

Acknowledgments. The authors thank Sally Goldman for her helpful comments on the draft of this paper. The second and third authors thank Wolfgang Maass for his insights into linking the finite injury priority method to the problem of learning unions of rectangles, Ming Li for valuable discussions on this topic, and William Gasarch for his encouraging comments on the use of the bounded injury priority method in this setting.

References

[A88] D. Angluin. Queries and concept learning. Machine Learning, 2 (4), (1988), 319-342.

[AL88] D. Angluin and P. Laird. Learning from noisy examples. Machine Learning, 2 (4), (1988), 471-479.

[AU93] P. Auer. On-line learning of rectangles in noisy environment. COLT 1993, 253-261.

[B90] A. Blum. Separating Distribution-Free and Mistake-Bound Learning Models over the Boolean Domain, FOCS 1990, 211-218.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. JACM, 36 (4), (1989), 929-965.

[Bs93] N. Bshouty. Exact learning via the monotone theory. FOCS 1993.

[BGGM94] N. H. Bshouty, P. W. Goldberg, S. A. Goldman and H. D. Mathias. Exact learning of discretized geometric concepts. TR. Washington university, WUCS-94-19.

[C93] Z. Chen. Learning union of two rectangles in the plane with equivalence queries. COLT 1993, 243-252.

[CHa93] Z. Chen, S. Homer. Learning unions of rectangles with queries. TR BUCS-93-10, Boston University, 1993.

[CHb94] Z. Chen, S. Homer. The bounded injury priority method and the learnability of unions of rectangles. TR BUCS-94-9, Boston University, 1994.

[CM92] Z. Chen and W. Maass. On-line learning of rectangles. COLT 1992, 16-27.

[GGM94] P. W. Goldberg, S. A. Goldman and H. D. Mathias. Learning union of rectangles with membership and equivalent queries. COLT 1994, 198-207.

[L88] N. Littlestone. Learning when irrelevant attributes abound: A new linear threshold algorithm. Machine Learning, 2, (1988), 285-318.

[LW90] P. M. Long and M. K. Warmuth. Composite geometric concepts and polynomial predictability. COLT 1990, 273-287.

[MT90] W. Maass and G. Turan. On the complexity of learning from counterexamples. FOCS 1989, 262-267.

[MT91] W. Maass and G. Turan. Algorithm and lower bounds for on-line learning of geometric concept. TR IIG-Report 316, Technische Universitat Graz, (1991).

[MT92] W. Maass and G. Turan. Lower bound methods and separation results for online learning models. Machine Learning, 9, (1992), 107-145.

[MW94] W. Maass, M. Warmuth. Efficient learning methods with virtual threshold gates. Tech Report 395 of the Institutes for Information Processing Graz (August 94).

[So87] R. Soare. Recursively Enumerable Sets and Degrees. Springer-Verlag, 1987.

[V84] L. Valiant. A theory of the learnable. JACM, 27, (11), (1984), 1134-1142.