Algorithmica (1998) 22: 76–90

Algorithmica

Learning Boxes in High Dimension[1]

A. Beimel[2] and E. Kushilevitz[3]

Abstract. We present exact learning algorithms that learn several classes of (discrete) boxes in $\{0, \ldots, \ell - 1\}^n$. In particular we learn: (1) The class of unions of $O(\log n)$ boxes in time poly$(n, \log \ell)$ (solving an open problem of [16] and [12]; in [3] this class is shown to be learnable in time poly(n, ℓ)). (2) The class of unions of disjoint boxes in time poly$(n, t, \log \ell)$, where t is the number of boxes. (Previously this was known only in the case where all boxes are disjoint in one of the dimensions; in [3] this class is shown to be learnable in time poly(n, t, ℓ).) In particular our algorithm learns the class of decision trees over n variables, that take values in $\{0, \ldots, \ell - 1\}$, with comparison nodes in time poly$(n, t, \log \ell)$, where t is the number of leaves (this was an open problem in [9] which was shown in [4] to be learnable in time poly(n, t, ℓ)). (3) The class of unions of $O(1)$-degenerate boxes (that is, boxes that depend only on $O(1)$ variables) in time poly$(n, t, \log \ell)$ (generalizing the learnability of $O(1)$-DNF and of boxes in $O(1)$ dimensions). The algorithm for this class uses only equivalence queries and it can also be used to learn the class of unions of $O(1)$ boxes (from equivalence queries only).

Key Words. Boxes, Discrete geometric objects, Decision trees, Exact learning, Multiplicity automata, Hankel matrices.

1. Introduction.

The learnability (under various learning models) of geometric concept classes was studied in many papers (e.g., [8], [11], [13], and [5]). Particular attention was given to the case of discrete domains of points (i.e., $\{0, \ldots, \ell - 1\}^n$) and concept classes which are defined as unions of boxes in this domain (e.g., [22], [23], [15], [2], [16], [17], and [24]).

One of the reasons that unions of boxes seem to be interesting concepts is that they naturally extend DNF formulae (in other words, in the case $\ell = 2$ any union of t boxes is equivalent to a DNF formula with t-terms). That is, a box can be viewed as a conjunction of nonboolean properties of the form "the attribute x_i is in the range between a_i and b_i." Similar to the special case of DNF functions, the learnability of unions of boxes in time poly$(n, t, \log \ell)$ (where t is the number of boxes in the union) is an open problem in all models of learning. Note that to represent such a function $\Theta(t \cdot n \cdot \log \ell)$ bits are required. Hence efficiency is defined as polynomial in $t, n,$ and $\log \ell$. Research on the problem of learning the class of unions of boxes (again, with similarity to the case of

---

[1] A preliminary version of this paper appeared in the proceedings of the EuroCOLT '97 conference, published in volume 1208 of Lecture Notes in Artificial Intelligence, pages 3–15, Springer-Verlag, New York, 1997. Part of the research by A. Beimel was done while he was a Ph.D. student at the Technion. The research by E. Kushilevitz was supported by Technion V.P.R. Fund 120-872 and by the Japan Technion Society Research Fund.

[2] DIMACS Center, Rutgers University, P.O. Box 1179, Piscataway, NJ 08855, USA. beimel@dimacs.rutgers.edu. http://dimacs.rutgers.edu/∼beimel.

[3] Department of Computer Science, Technion, Haifa 32000, Israel. eyalk@cs.technion.ac.il. http://www.cs.technion.ac.il/∼eyalk.

Received January 19, 1997; revised June 4, 1997. Communicated by P. Auer and W. Maass.

DNF formulae) attempts to learn subclasses of this class. There are two main directions: (1) Subclasses in which the number of dimensions, n, is limited to $O(1)$. In this case unions of boxes (and more general geometric concept classes) are known to be learnable in the PAC model [13] and even in the weaker on-line model [5]. (2) Subclasses in which the number of boxes in the union is limited to $O(1)$ (but the number of dimensions is not restricted). Again, this subclass is learnable in the PAC model [21] and in the on-line model [24].

In this work we generalize some of the state-of-the-art results in *exact learning* of DNF formulae [6], [4], hence strengthening several results in direction (2) above. In particular we show:

1. The class of all unions of $O(\log n)$ boxes can be learned in time poly$(n, \log \ell)$ (solving an open problem of [16] and [12]; in [3] this class is shown to be learnable in time poly(n, ℓ)). This generalizes a similar result for the class of $O(\log n)$-term DNF [7], [9], [10], [19], [4].
2. The class of all unions of disjoint boxes (and, more generally, unions of boxes in which each point belongs to at most $O(1)$ boxes) can be learned in time poly$(n, t, \log \ell)$, where t is the number of boxes (in [3] this class is shown to be learnable in time poly(n, t, ℓ)). This generalizes similar results for learning disjoint DNF and satisfy-$O(1)$ DNF [6], [4]. (Previously such a result was known only in the case where all boxes are disjoint in one of the dimensions [11], [14]; in this case in fact equivalence queries are sufficient.) In particular our algorithm learns the class of decision trees with comparison nodes in time poly$(n, t, \log \ell)$, where t is the number of leaves (the learnability of this class was an open problem in [9]; in [4] it was shown that this class is learnable in time poly(n, t, ℓ)).
3. The class of all unions of $O(1)$-degenerate boxes, that is boxes that depend only on $O(1)$ variables, can be learned in time poly$(n, t, \log \ell)$, where t is the number of boxes. In this case only equivalence queries are used, i.e. we learn this class in the on-line model [20]. This result generalizes the learnability of $O(1)$-DNF and of boxes in $O(1)$ dimensions. The class of k-degenerate boxes was previously considered in [17], [18], and [24]. Our algorithm for this class also learns the class of unions of $O(1)$ boxes from equivalence queries only.

The first two results are obtained in two steps: First, in Section 3, we show how to learn these classes but with complexity which is polynomial in ℓ (and the other parameters of the problem). This result appears in [3] and is a straightforward generalization of results in [4]. It is included in this paper for the sake of completeness. Then, in Section 4, we prove the main result of this paper: we show how to (adaptively) select a "small" subset of the domain $\{0, 1, \ldots, \ell - 1\}$ which is sufficient for the learning. Hence, we give a reduction that converts *any* algorithm for learning unions of boxes whose complexity is polynomial in ℓ to an algorithm with complexity which is polynomial only in $\log \ell$ (and the other parameters of the problem). Finally, in Section 5 we show how to convert a simple poly(n, t, ℓ) algorithm that learns unions of $O(1)$-degenerate boxes into a poly$(n, t, \log \ell)$ algorithm. In this case the poly(n, t, ℓ) algorithm that we start with does not use membership queries. We use a refined conversion which also does not use memberships queries; hence we get an algorithm with equivalence queries only. This conversion uses specific properties of the poly(n, t, ℓ) algorithm.

2. Preliminaries

2.1. *The Learning Model.* The learning model we use is the *exact learning* model [1]: Let $\mathcal{C}$ be a class of functions and let $f \in \mathcal{C}$ be a *target* function. A learning algorithm may propose, in each step, a hypothesis function h by making an *equivalence query* (EQ) to an oracle. If h is logically equivalent to f, then the answer to the query is YES and the learning algorithm succeeds and halts. Otherwise, the answer to the EQ is NO and the algorithm receives a *counterexample*—an assignment z such that $f(z) \neq h(z)$. The learning algorithm may also query an oracle for the value of the function f on a particular assignment z by making a *membership query* (MQ) on z. The response to such a query is the value $f(z)$. We say that the learning algorithm *learns* a class of functions $\mathcal{C}$, if for every function $f \in \mathcal{C}$ the learning algorithm outputs a hypothesis h that is logically equivalent to f and does so in time polynomial in the "size" of a shortest representation of f.

We sometimes restrict the learning algorithm to use EQs only. This is equivalent to Littlestone's *on-line* model of learning [20].

2.2. *Classes of Boxes.* In this section we define the classes of boxes that we learn in this paper. We consider unions of n-dimensional boxes in $[\ell]^n$ (where $[\ell]$ denotes the set $\{0, 1, \ldots, \ell - 1\}$). Formally, a box in $[\ell]^n$ is defined by two corners $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ (in $[\ell]^n$) as follows:

$$B_{a_1,\ldots,a_n,b_1,\ldots,b_n} = \{(x_1, \ldots, x_n) \ : \ \forall i,\ a_i \le x_i \le b_i\ \}.$$

We view such a box as a boolean function that gives 1 for every point in $[\ell]^n$ which is inside the box and 0 to each point outside the box. More generally, the boolean function corresponding to the *union* of boxes $B_1, \ldots, B_t$ is defined to be 1 for every point in $[\ell]^n$ which is inside (at least) one of the t boxes and 0 otherwise.

Denote by BOX_t the set of all functions that correspond to unions of (at most) t boxes and by DISJ-BOX_t the set of all functions that correspond to unions of (at most) t *disjoint* boxes. A k-degenerate box is a box that depends only on k variables, where a box *depends* on the ith variable if either $a_i \neq 0$ or $b_i \neq \ell - 1$. Denote by $k - \text{DBOX}_t$ the set of all functions that correspond to unions of (at most) t k-degenerate boxes. For the case $\ell = 2$ the class BOX_t corresponds to the class of t-term DNF, the class DISJ-BOX_t corresponds to the class (t-term) disjoint DNF, and the class $k - \text{DBOX}_t$ corresponds to the class (t-term) k-DNF. Note that the number of boxes, t, for a function in $k - \text{DBOX}_t$ can be at most $\binom{n}{k} \cdot \ell^{2k}$. This is $\text{poly}(n, \ell)$, for $k = O(1)$, but may be much larger than $\log \ell$.

Notation. We end this section with some useful notation. Given a set $L \subseteq [\ell]$ (such that $0 \in L$) and a letter $a \in [\ell]$ we denote by $\lfloor a \rfloor$ the largest value in L which is at most a, that is, $\lfloor a \rfloor \triangleq \max\{\sigma \in L : \sigma \le a\}$. Similarly, we denote $\lceil a \rceil \triangleq \min\{\sigma \in L : \sigma > a\}$ (if $\sigma \le a$ for every $\sigma \in L$, then $\lceil a \rceil \triangleq \ell$). Whenever we use these notations the ground set L will be clear from the context. Denote by f_L the function f restricted to the range L^n, that is, $f_L \colon L^n \to \{0, 1\}$ and $f_L(x) = f(x)$ for every $x \in L^n$. Note that if f is a union of at most t boxes, then so is f_L.

3. Learning Boxes Using Hankel Matrices. We consider the learnability, in the exact-learning model, of the classes DISJ-BOX$_t$ and BOX$_{O(\log n)}$. Our starting point is a recent algorithm of [4] and [3] for learning multiplicity automata. In [3] it is shown that this algorithm can be used to learn DISJ-BOX$_t$ and BOX$_{O(\log n)}$ in time polynomial in ℓ and the other parameters of the problem (this is a straightforward generalization of results in [4]). For the sake of completeness we prove this result in this section. In addition, in the Appendix we briefly sketch the algorithm of [4] and [3]. Then, in the next section, we show how to reduce the complexity to be polynomial in $\log \ell$.

We start with some notations. Let $\mathcal{K}$ be a field, let Σ be an alphabet, and let $f: \Sigma^n \to \mathcal{K}$ be a function. The *Hankel matrix* corresponding to f, denoted F, is defined as follows: each row of F is indexed by a string $x \in \Sigma^{\leq n}$; each column of F is indexed by a string $y \in \Sigma^{\leq n}$; the (x, y) entry of F contains the value of $f(x \circ y)$ (where $\circ$ denotes concatenation) if $x \circ y$ is of length (exactly) n and the value 0 otherwise.

THEOREM 3.1 [4]. *There is an algorithm* LEARN_HANKEL *that learns every function* $f: \Sigma^n \to \mathcal{K}$ *with time* (*and query*) *complexity* $\text{poly}(n, \text{rank}(F), |\Sigma|)$, *where* rank *is defined with respect to the field* $\mathcal{K}$. (*See the Appendix*.)

By the above theorem, to prove the learnability of a concept class it is sufficient to give an upper bound on $\text{rank}(F)$ for the matrices corresponding to functions in the class. For convenience (and efficiency), we fix $\mathcal{K}$ to be GF(2) (although the next lemma holds for any field).

LEMMA 3.2. *Let* $B_1, \ldots, B_t$ *be t boxes in* $[\ell]^n$ *such that there is no point* $x \in [\ell]^n$ *which belongs to more than s boxes and let* f *be the function corresponding to the union of these boxes* (*e.g., for* $s = 1$ *we get the functions in* DISJ-BOX$_t$). *Then*

$$\text{rank}(F) \leq (n+1) \cdot \sum_{i=1}^{s} \binom{t}{i}.$$

PROOF. Let F^d denote the submatrix of F whose rows are indexed by strings $x \in \Sigma^d$ and whose columns are indexed by strings $y \in \Sigma^{n-d}$ (see Figure 1). Note that by the definition of F all entries which are not in one of the submatrices F^d are zeros. Hence, by linear algebra, $\text{rank}(F) = \sum_{d=0}^{n} \text{rank}(F^d)$. Therefore, it is sufficient to prove, for every d, that $\text{rank}(F^d) \leq \sum_{i=1}^{s} \binom{t}{i}$.

Let B be any box and denote the two corners of B by $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$. Define functions (of a single variable) $p_j(z_j): [\ell] \to \{0, 1\}$ to be 1 if $a_j \leq z_j \leq b_j$ $(1 \leq j \leq n)$. Let $g: [\ell]^n \to \{0, 1\}$ be defined by $\prod_{j=1}^{n} p_j(z_j)$ (i.e., $g(z_1, \ldots, z_n)$ is 1 if and only if $(z_1, \ldots, z_n)$ belongs to the box B). Let G be the Hankel matrix corresponding to g and let G^d be its corresponding submatrix. Every row of G^d is indexed by $x \in \Sigma^d$, and its yth coordinate can be written as

$$G_x^d(y) = g(x \circ y) = \left(\prod_{j=1}^{d} p_j(x_j) \right) \left(\prod_{j=1}^{n-d} p_{j+d}(y_j) \right),$$

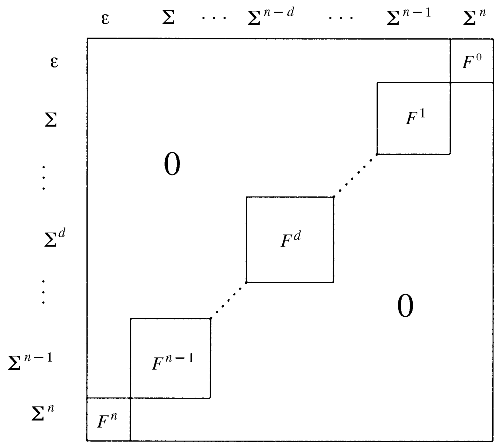

Fig. 1. The Hankel matrix F.

where $x = x_1 \cdots x_d$ and $y = y_1 \cdots y_{n-d}$. Now, for every x, the term $\prod_{j=1}^{d} p_j(x_j)$ is just a constant $\alpha_x \in \{0, 1\}$. Thus, every row $G^d_x(y)$ is just a constant times the vector whose yth coordinate is $\prod_{j=1}^{n-d} p_{j+d}(y_j)$. This implies that $\mathrm{rank}(G^d) \leq 1$. Finally, note that if g_i is the function corresponding to the box B_i, then f can be expressed as

$$\begin{aligned} f &= 1 - \prod_{i=1}^{t}(1 - g_i) \\ &= \sum_i g_i - \sum_{i,j}(g_i \wedge g_j) + \cdots + (-1)^{t+1} \sum_{|S|=t} \bigwedge_{i \in S} g_i \\ &= \sum_i g_i - \sum_{i,j}(g_i \wedge g_j) + \cdots + (-1)^{s+1} \sum_{|S|=s} \bigwedge_{i \in S} g_i, \end{aligned}$$

where the last equality is by the assumption that no point belongs to more than s boxes. Also note that each term of the form $h = \bigwedge_{i \in S} g_i$ is an intersection of boxes which is a box by itself. Hence, by the above, the rank of the matrix H^d corresponding to each of these terms is at most 1. Since we wrote f as a linear combination of $\sum_{i=1}^{s} \binom{t}{i}$ such terms we get, by linear algebra, that $\mathrm{rank}(F^d) \leq \sum_{i=1}^{s} \binom{t}{i}$. □

Combining the above lemma with Theorem 3.1 we get:

COROLLARY 3.3. *The class* $\mathrm{BOX}_{O(\log n)}$ *can be learned in time* $\mathrm{poly}(n, \ell)$.

COROLLARY 3.4. *The class* DISJ-BOX *can be learned in time* $\mathrm{poly}(n, t, \ell)$ (*where* t *is the number of boxes in the target functions*).

In [19] it is shown how to learn $O(\log n)$-term DNF using deterministic automata. The algorithm of [19] can also be modified to learn the class $\mathrm{BOX}_{O(\log n)}$ in time $\mathrm{poly}(n, \ell)$, yielding a different proof for Corollary 3.3.

4. Reducing the Dependency on ℓ. In this section we reduce the dependency of our algorithm on ℓ. For this, we define the notion of *sensitive* letters:

DEFINITION 4.1. A letter $\sigma \in [\ell]$ is called *i-sensitive* with respect to f if there exist letters $c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n \in [\ell]$ such that

$$f(c_1, \ldots, c_{i-1}, \sigma - 1, c_{i+1}, \ldots, c_n) \neq f(c_1, \ldots, c_{i-1}, \sigma, c_{i+1}, \ldots, c_n).$$

A letter σ is called *sensitive* with respect to f if σ is i-sensitive for some i.

LEMMA 4.2. *Let f be a function in* BOX_t. *Then there are at most $2nt$ sensitive letters with respect to f.*

PROOF. If $f(c_1 \cdots c_{i-1}, \sigma - 1, c_{i+1} \cdots c_n) = 0$ and $f(c_1 \cdots c_{i-1}, \sigma, c_{i+1} \cdots c_n) = 1$, then for some box B_j the letter σ is the ith coordinate of the lower corner. If $f(c_1, \ldots, c_{i-1}, \sigma - 1, c_{i+1}, \ldots, c_n) = 1$ and $f(c_1, \ldots, c_{i-1}, \sigma, c_{i+1}, \ldots, c_n) = 0$, then for some box B_j the letter $\sigma - 1$ is the ith coordinate of the upper corner. Since there are at most t boxes, in n dimensions, and each is defined by its two corners, the lemma follows. $\square$

In Figure 2 we describe an algorithm to learn the classes $\text{BOX}_{O(\log n)}$ and DISJ-BOX. The idea behind the algorithm is to learn the set of sensitive letters with respect to f as part of learning the function f. At each stage, with the set of current letters, denoted L, the algorithm tries to learn the function using the algorithm LEARN_HANKEL as a black box. Either the algorithm succeeds, or it finds another sensitive letter and starts a new execution of the algorithm LEARN_HANKEL.

While executing the algorithm LEARN_HANKEL as a black box, the algorithm simulates the MQs and EQs to f_L asked by this black box, using its membership and equivalence oracles for f. To answer an MQ about f_L, it simply uses the oracle for f (this is correct since $f_L(x) = f(x)$ for every $x \in L^n$). To simulate an equivalence query $\text{EQ}(h)$ to f_L, using the corresponding oracle for f, the hypothesis h is extended to a hypothesis $h'\colon [\ell]^n \to \{0, 1\}$ by $h'(x_1, \ldots, x_n) = h(\lfloor x_1 \rfloor, \ldots, \lfloor x_n \rfloor)$. (The intuition behind this definition is that if L contains all sensitive letters, then $f(x_1, \ldots, x_n) = f_L(\lfloor x_1 \rfloor, \ldots, \lfloor x_n \rfloor)$.) If $\text{EQ}(h')$ returns a counterexample y, then there are two cases. If $f(y_1, \ldots, y_n) = f(\lfloor y_1 \rfloor, \ldots, \lfloor y_n \rfloor)$, then $(\lfloor y_1 \rfloor, \ldots, \lfloor y_n \rfloor)$ is a counterexample to h, and the algorithm passes this counterexample to the black box. Otherwise, if $f(y_1, \ldots, y_n) \neq f(\lfloor y_1 \rfloor, \ldots, \lfloor y_n \rfloor)$, then there must be a sensitive letter σ in the interval $\{\lfloor y_i \rfloor + 1, \ldots, y_i\}$ for some index i. This sensitive letter is found with $\log n + \log \ell$ MQs using a binary search. Then this sensitive letter is added to L and a new execution of LEARN_HANKEL is started.

Using the algorithm LEARN_SENSITIVE we prove the following results:

THEOREM 4.3. *The class* $\text{BOX}_{O(\log n)}$ *can be learned in time* $\text{poly}(n, \log \ell)$.

PROOF. We use the algorithm LEARN_SENSITIVE to learn the class $\text{BOX}_{O(\log n)}$. As remarked, for every L the function f_L is also a union of $O(\log n)$ boxes. By Corollary 3.3,

ALGORITHM LEARN_SENSITIVE

1. $L \leftarrow \{0\}$
2. Learn the function f_L using the algorithm LEARN_HANKEL.
   To answer membership queries about f_L simply use the oracle for f.
   To simulate an equivalence query EQ(h) to f_L:
   (a) Define $h'\colon [\ell]^n \to \{0,1\}$ as $h'(x_1,\ldots,x_n) = h(\lfloor x_1 \rfloor,\ldots,\lfloor x_n \rfloor)$.
   Ask whether $h' \equiv f$.
   If the answer is "YES" halt with output h'.
   Otherwise, we have a counterexample $(y_1,\ldots,y_n) \in [\ell]^n$.
   (b) If $f(y_1,\ldots,y_n) = f(\lfloor y_1 \rfloor,\ldots,\lfloor y_n \rfloor)$ (this is checked using an MQ) then $(\lfloor y_1 \rfloor,\ldots,\lfloor y_n \rfloor)$ is a counterexample to h—pass this counterexample to the algorithm for learning f_L and continue its execution.
   Otherwise, proceed to Step (3).
3. $(f(y_1,\ldots,y_n) \neq f(\lfloor y_1 \rfloor,\ldots,\lfloor y_n \rfloor).)$
   Find an index $1 \le i \le n$ such that

$$f(\lfloor y_1 \rfloor \cdots \lfloor y_{i-1} \rfloor, y_i \cdots y_n) \neq f(\lfloor y_1 \rfloor \cdots \lfloor y_{i-1} \rfloor, \lfloor y_i \rfloor, y_{i+1} \cdots y_n)$$

   (this is found with $O(\log n)$ MQs using a binary search).
   Then find a letter σ such that $\lfloor y_i \rfloor + 1 \le \sigma \le y_i$ and

$$f(\lfloor y_1 \rfloor \cdots \lfloor y_{i-1} \rfloor, \sigma - 1, y_{i+1} \cdots y_n) \neq f(\lfloor y_1 \rfloor \cdots \lfloor y_{i-1} \rfloor, \sigma, y_{i+1} \cdots y_n)$$

   (this is found with $O(\log \ell)$ MQs using a binary search).
   Set $L \leftarrow L \cup \{\sigma\}$ and start Step (2) again.

Fig. 2. An algorithm for learning the sensitive letters.

learning the function f_L (Step (2)) ends within time poly$(n, |L|)$. It ends either by identifying the target function f, or by adding a new sensitive letter to L (Step (3)). In addition, once L contains all sensitive letters then, for every point $(y_1,\ldots,y_n) \in [\ell]^n$,

$$f(y_1,\ldots,y_n) = f(\lfloor y_1 \rfloor,\ldots,\lfloor y_n \rfloor). \tag{1}$$

At this point, since there are no more sensitive letters, the algorithm for f_L will find a hypothesis $h \equiv f_L$ which by (1) implies $h' \equiv f$. By Lemma 4.2 the size of L, and hence the number of times Step (3) is executed, is $O(n \log n)$. Each time a new sensitive letter is inserted to L we spend poly$(n, \log \ell)$ time searching for such a letter. To conclude, algorithm LEARN_SENSITIVE learns the class $\mathrm{BOX}_{O(\log n)}$ in time poly$(n, \log \ell)$. $\square$

The above theorem solves an open problem of [16] and [12]. The following theorem shows the learnability of disjoint boxes in time poly$(n, t, \log \ell)$. This significantly improves over [11] and [14] where a similar result was shown for the case where there exists a dimension in which all the boxes are disjoint.

THEOREM 4.4. *The class* DISJ-BOX *can be learned in time* $\mathrm{poly}(n, t, \log \ell)$ (*where* t *is the number of boxes in the target function*).

PROOF. Again, we use the algorithm LEARN_SENSITIVE to learn this class. By Lemma 4.2 the size of L, and hence the number of times Step (3) is executed, is $O(nt)$. Each time a new sensitive letter is inserted to L we spend $\mathrm{poly}(n, \log \ell)$ time searching for such a letter. By Corollary 3.4 learning the function f_L (Step (2)) takes time $\mathrm{poly}(n, t, |L|)$ (observe that the function f_L always consists of a union of at most t disjoint boxes). To conclude, algorithm LEARN_SENSITIVE learns any function in DISJ-BOX_t in time $\mathrm{poly}(n, t, \log \ell)$. $\square$

A special case of disjoint boxes are functions which are represented by decision trees where each node contains a boolean query of the form "Is $x_i \geq \theta$?" (and where the variables x_i and the constants θ take values in $[\ell]$). The learnability of this class was an open problem in [9]. In [4] it was shown that this class is learnable in time $\mathrm{poly}(n, t, \ell)$ (where t here denotes the number of leaves in the tree corresponding to f). Algorithm LEARN_SENSITIVE shows that this class can in fact be learned in time $\mathrm{poly}(n, t, \log \ell)$.

REMARK 4.5. Obviously, Corollary 3.4 and Theorem 4.4 can be easily extended to the case where no point x is contained in more than s boxes, for $s = O(1)$ (note that Lemma 3.2 is already formulated in a way that allows this extension). Also, Theorem 4.3 can be extended to learning decision trees of depth $O(\log n)$, where each node contains a boolean query of the form "does x belong to a box B?" (for this result we can use Corollary 4.11 from [4] to learn f_L, and slightly generalize Lemma 4.2). In particular this shows the learnability of *any* function of $O(\log n)$ boxes, and not only *unions* of boxes.

REMARK 4.6. An improved complexity can be obtained if instead of collecting all the sensitive letters in a single set L we would maintain a separate set L_i for the i-sensitive letters.

REMARK 4.7. A box is just the product of n intervals, one in each dimension. We can consider more general boxes which are products of (at most) m intervals in each dimension. Such a general box can be viewed as the union of m^n "regular" boxes. Nevertheless, it can be easily seen that our algorithms can be extended to this case as well, with a small increase in the complexity. For example, the union of t disjoint, general boxes can be learned in time which is $\mathrm{poly}(n, t, \log \ell, m)$. This is because Lemma 3.2 still holds, and since the number of sensitive letters is now bounded by $2mnt$.

5. Learning $O(1)$-Degenerate Boxes. In this section we show how to learn the class $k - \text{DBOX}_t$, for $k = O(1)$, using only EQs. This generalizes the learnability of unions of boxes in $O(1)$ dimensions and, as will be shown, can be used to learn unions of $O(1)$ boxes in $[\ell]^n$. Again, we start with an algorithm which is polynomial in ℓ and convert it into an algorithm which is polynomial in $\log \ell$. We use a refined transformation which

ALGORITHM ELIMINATE_BOXES

1. Make a list Q of all width one k-degenerate boxes.
2. Define a hypothesis h as the union of all boxes in the list Q.
3. Ask $\mathrm{EQ}(h)$.
   If the answer is "YES" halt with output h.
4. Otherwise, the answer is "NO" and y is a counterexample.
   Remove all the boxes in Q that contain y.
   Goto 2.

Fig. 3. A $\mathrm{poly}(n, \ell)$ algorithm for $O(1)$-degenerate boxes.

does not use MQs. However, this transformation is not general and uses specific properties of the algorithm that we start with.

We first define the class of *width one boxes* which is a certain class of boxes that we use in our algorithm.

DEFINITION 5.1. A *width one box* is a box in which for every dimension either the width is 1 (i.e., $a_i = b_i$) or the box does not depend on the ith variable (i.e., $a_i = 0$ and $b_i = \ell - 1$).

It is a simple observation that every k-degenerate box can be written as the union of at most ℓ^k width one k-degenerate boxes, and hence every function in $k - \mathrm{DBOX}_t$ can be written as the union of at most $t \cdot \ell^k$ width one k-degenerate boxes. In Figure 3 we describe a simple algorithm, ELIMINATE_BOXES, which learns the class $k - \mathrm{DBOX}_t$ with complexity which is polynomial in ℓ, for $k = O(1)$. This algorithm is a variant of the standard elimination algorithm for learning k-DNF [25].

Note that algorithm ELIMINATE_BOXES uses only EQs (i.e., no MQs). We start with a simple observation about the algorithm.

CLAIM 5.2. *Let f, the target function, be any function in $k - \mathrm{DBOX}_t$. In every step of algorithm* ELIMINATE_BOXES $f \le h$ (*i.e.*, $f(x) = 1$ *implies* $h(x) = 1$).

PROOF. As remarked, f can be represented as a union of width one k-degenerate boxes. Let $Q^\star$ be the set of these boxes. For proving the claim, it suffices to prove that at any time $Q^\star \subseteq Q$. This is obviously true at the beginning, since Q contains *all* width one k-degenerate boxes. Whenever a counterexample y is received, since $Q^\star \subseteq Q$ it must be that $h(y) = 1$ and $f(y) = 0$; hence, when the boxes that contain y are removed from Q, none of them is in $Q^\star$. Therefore, after Q is modified in Step (4) still $Q^\star \subseteq Q$. $\square$

We next prove that the algorithm is correct.

LEMMA 5.3. *Algorithm* ELIMINATE_BOXES *learns the class $k - \mathrm{DBOX}_t$, for $k = O(1)$, in time* (*and query*) *complexity* $\mathrm{poly}(n, \ell)$.

PROOF. By the code, if the algorithm halts, then its hypothesis is equivalent to f. We have to prove that the algorithm must halt within $\text{poly}(n, \ell)$ time. By Claim 5.2, for every counterexample, $h(y) = 1$ while $f(y) = 0$. Thus, every equivalence query removes at least one width one k-degenerate box from Q. The size of Q when the algorithm starts is the number of all width one k-degenerate boxes which is less than $\ell^k n^k$. Thus, algorithm ELIMINATE_BOXES uses at most $\ell^k n^k$ equivalence queries, and runs in time $\text{poly}(n^k, \ell^k)$. $\square$

Using the transformation of Section 4 together with algorithm ELIMINATE_BOXES we can learn the class $k - \text{DBOX}_t$ in time $\text{poly}(n, t, \log \ell)$ with EQs and MQs. However, our goal is an algorithm that does not use MQs and still has complexity which is polynomial in $\log \ell$. The idea is again to learn the sensitive letters adaptively. There are two problems in learning the sensitive letters without MQs. The first problem is that when algorithm LEARN_SENSITIVE (Figure 2) gets a counterexample $(y_1, \ldots, y_n)$ it decides, using an MQ, whether it can return $(\lfloor y_1 \rfloor, \ldots, \lfloor y_n \rfloor)$ as a counterexample to the algorithm that learns the restricted function f_L (Step (2b)). Since we know that every counterexample in the algorithm ELIMINATE_BOXES is a negative counterexample (that is, $f(y) = 0$ while $h(y) = 1$), we pass all the negative counterexamples to the algorithm for f_L, while every positive counterexample is used to look for a sensitive letter (we prove below that this strategy works). The second problem that we face is how to search for a sensitive letter without using MQs. We use an idea of [11] and [12]: if there is a sensitive letter in a set $\{a, a+1, \ldots, b\}$ where a and b are in L, then add $(a+b)/2$ to L (if $(a+b)/2$ is not an integer, then add $(a+b-1)/2$). In this case the set L of letters that is used by the algorithm will be a superset of the sensitive letters. However, the size of L will still be relatively small.

In Figure 4 we describe our algorithm FAST_ELIMINATION. Every time that this algorithm reaches Step (6) it adds at least one new letter to L. Furthermore, if L contains all the sensitive letters, then the algorithm finds a hypothesis that is equivalent to f. Thus, since $L \subseteq [\ell]$, algorithm FAST_ELIMINATION will eventually halt with the right answer. As in Lemma 5.3, if L is the set of letters when the algorithm halts, then the complexity of the algorithm is $\text{poly}(n^k, t^k, |L|^k)$. Thus, it remains to prove that when the algorithm FAST_ELIMINATION halts, L is small. This is based on the next claim.

CLAIM 5.4. *Every time algorithm* FAST_ELIMINATION *reaches Step* (6) *there exists an index i and a sensitive letter σ in* $\{\lfloor y_i \rfloor + 1, \ldots, \lceil y_i \rceil - 1\}$.

PROOF. If the algorithm reaches Step (6), then $f(y) = 1$, i.e., there exists some k-degenerate box $B = B_{a_1,\ldots,a_n,b_1,\ldots,b_n}$ in f that contains the counterexample y. Consider the box $B' = B_{a'_1,\ldots,a'_n,b'_1,\ldots,b'_n}$ where if $a_i = 0$ and $b_i = \ell - 1$, then $a'_i = 0$ and $b'_i = \ell - 1$, and otherwise $a'_i = b'_i = \lfloor y_i \rfloor$. This is a width one k-degenerate box over L that contains $(\lfloor y_1 \rfloor, \ldots, \lfloor y_n \rfloor)$. Since $h(\lfloor y_1 \rfloor, \ldots, \lfloor y_n \rfloor) = 0$, this box B' is not in Q. Let z be the counterexample that removed B' from Q in some previous execution of Step (5) after the last execution of Step (2) (thus, L has not changed since z removed B' from Q). For $1 \le i \le n$ define $w_i = y_i$ if B depends on the ith variable and $w_i = z_i$ otherwise. By definition, $f(w_1, \ldots, w_n) = 1$ while $f(z_1, \ldots, z_n) = 0$. Thus, there exists an index i

ALGORITHM FAST_ELIMINATION

1. $L \leftarrow \{0\}$.
2. Make a list Q of all width one k-degenerate boxes over the current set L.
3. Define a hypothesis h: $L^n \to \{0, 1\}$ as the union of all boxes in Q.
   Extend the hypothesis to $[\ell]^n$ by

$$h'(x_1, \ldots, x_n) \triangleq h(\lfloor x_1 \rfloor, \ldots, \lfloor x_n \rfloor).$$

4. Ask EQ(h').
   If the answer is "YES" halt with output h'.
   Otherwise, the answer is "NO" and y is a counterexample.
5. If $h'(y) = 1$ and $f(y) = 0$ then:
   Remove all boxes that contain $(\lfloor y_1 \rfloor, \ldots, \lfloor y_n \rfloor)$ from Q.
   Goto (3).
6. ($h'(y) = 0$ and $f(y) = 1$.)
   For $i = 1$ to n do:
   If $\lceil y_i \rceil + \lfloor y_i \rfloor$ is even then $L \leftarrow L \cup \{(\lceil y_i \rceil + \lfloor y_i \rfloor)/2\}$
   else $L \leftarrow L \cup \{(\lceil y_i \rceil + \lfloor y_i \rfloor - 1)/2\}$.
   Goto (2).

Fig. 4. A poly($n, t, \log \ell$) algorithm for $O(1)$-degenerate boxes.

such that

$$f(w_1, \ldots, w_{i-1}, w_i, z_{i+1}, \ldots, z_n) = 1$$

while

$$f(w_1, \ldots, w_{i-1}, z_i, z_{i+1}, \ldots, z_n) = 0.$$

This implies that there exists a sensitive letter σ such that either $z_i < \sigma \le y_i$ or $y_i < \sigma \le z_i$. Furthermore, $z_i \ne w_i$ and B depends on the ith variable. Therefore, $\lfloor z_i \rfloor = \lfloor y_i \rfloor$ (since $(\lfloor z_1 \rfloor, \ldots, \lfloor z_n \rfloor) \in B'$) which implies $\lfloor y_i \rfloor \le z_i < \lceil y_i \rceil$. To conclude, if $z_i < \sigma \le y_i$, then $\lfloor y_i \rfloor \le z_i < \sigma \le y_i < \lceil y_i \rceil$. Thus, the sensitive letter σ is in the interval $\{\lfloor y_i \rfloor + 1, \ldots, \lceil y_i \rceil - 1\}$. The case $y_i < \sigma \le z_i$ is similar. □

The next theorem completes the analysis of algorithm FAST_ELIMINATION.

THEOREM 5.5. *The class $k - \mathrm{DBOX}_t$, for $k = O(1)$, can be learned in time* (*and query*) *complexity* poly($n, t, \log \ell$) *using equivalence queries only.*

PROOF. We use the algorithm FAST_ELIMINATION to learn the class $k - \mathrm{DBOX}_t$, for $k = O(1)$. If L contains all sensitive letters, then, by Claim 5.4, we can get only counterexamples y such that $h'(y) = 1$ and after at most $|Q|$ such counterexamples the algorithm finds a hypothesis equivalent to f. Therefore, again by Claim 5.4, the algorithm reaches Step (6) at most $\log \ell$ times per each sensitive letter. By Claim 4.2, there are $O(nt)$

sensitive letters; hence, the algorithm reaches Step (6) only $O(nt \log \ell)$ times, and each time it adds at most n new letters to L. That is, $|L| = O(n^2 t \log \ell)$ and the number of boxes in Q (each time the algorithm reaches Step (2)) is $O(|L|^k n^k) = \mathrm{poly}(n^k, t^k, \log^k \ell)$. Each time algorithm FAST_ELIMINATION asks an EQ and does not reach Step (6) it holds that $h(y) = 1$, i.e., there is a box in Q that contains $(\lfloor y_1 \rfloor, \ldots, \lfloor y_n \rfloor)$ and the algorithm removes this box from Q. In other words, after at most $|Q| = \mathrm{poly}(n^k, t^k, \log^k \ell)$ EQs the algorithm reaches Step (6). That is, the running time is $\mathrm{poly}(n^k, t^k, \log^k \ell)$ which is $\mathrm{poly}(n, t, \log \ell)$ for $k = O(1)$. □

REMARK 5.6. Algorithm FAST_ELIMINATION can be used to learn the class of all unions of $O(1)$ boxes with only EQs. (This was an open problem in [15] that was later solved in [24].) The idea is that if we have a function that can be represented as a union of $O(1)$ boxes, then its negation can be represented as a union of $O(1)$-degenerate boxes. Let f be a union of k boxes, that is (using the notation of the proof of Lemma 3.2), there exist functions $p_{m,j}\colon [\ell] \to \{0, 1\}$ for $1 \le m \le k$ and $1 \le j \le n$ such that $f = \bigvee_{m=1}^{k} \prod_{j=1}^{n} p_{m,j}(x_j)$, and $p_{m,j}(x_j) = 1$ if x_j is in some interval. Therefore,

$$\overline{f} = \bigvee_{i_1,\ldots,i_k \in \{1,\ldots,n\}} \prod_{j=1}^{k} \overline{p_{j,i_j}(x_{i_j})}.$$

In words, a point x is not in the union of k boxes if there exist coordinates $i_1, \ldots, i_k$ such that for every j the variable x_{i_j} is not in the i_j-interval of the box B_j (i.e., $p_{j,i_j}(x_{i_j}) = 0$). Note that $\overline{p_{j,i_j}(x_{i_j})}$ is a union of (at most) two intervals. Thus, $\overline{f}$ can be represented as a union of at most n^k generalized boxes as defined in Remark 4.7. Each such generalized box can be represented as a union of at most 2^k (simple) k-degenerate boxes. Thus, $\overline{f}$ can be represented as a union of $O(k 2^k n^k)$ k-degenerate boxes, and algorithm FAST_ELIMINATION, as it is, learns the class of union of $O(1)$ boxes in time $\mathrm{poly}(n, \log \ell)$.

Acknowledgments. We would like to thank the EuroCOLT ‘97 program committee members and the anonymous referees for helpful comments.

Appendix. Learning Using Hankel Matrices. In this Appendix we briefly describe the algorithm of [4] and [3], which learns a function f using its Hankel matrix F (the definition of the Hankel matrix of a function appears in Section 3). The version described here is limited to functions $f\colon \Sigma^n \to \mathcal{K}$ which is what we need for the current paper (whereas the original algorithm works for the more general case of functions $f\colon \Sigma^\star \to \mathcal{K}$). Also we do not describe the most efficient version of the algorithm that exploits the specific structure of the Hankel matrix corresponding to functions of the form $f\colon \Sigma^n \to \mathcal{K}$ (as described in Figure 1). For full details (including a formal proof) the reader is referred to [4], [3].

The idea is as follows: at any given step the algorithm will have a matrix $\widehat{F}$ which is an $r \times r$ submatrix of F, the Hankel matrix corresponding to f. The rows of $\widehat{F}$ will be indexed by strings $X = \{x_1, \ldots, x_r\}$ and the columns of $\widehat{F}$ will be indexed by strings

$Y = \{y_1, \ldots, y_r\}$ (note that based on X and Y the matrix $\widehat{F}$ can be generated using r^2 MQs). We maintain the property that $\widehat{F}$ is of full rank (i.e., it has rank r). Obviously the rank of $\widehat{F}$ is bounded by the rank of F. The algorithm will start by asking EQ(0). If the target function is identically 0 we are done. Otherwise we will get a counterexample z such that $f(z) = 1$. We initialize $x_1 = \varepsilon, x_2 = z, y_1 = \varepsilon, y_2 = z$ so we get a 2×2 matrix $\widehat{F}$ of rank 2.

The structure of the algorithm is as follows. Given $\widehat{F}$ we show below how to construct a hypothesis h such that from a counterexample z satisfying $f(z) \neq h(z)$ we can always find a row and column such that adding them to $\widehat{F}$ increases the rank by 1. If we can do this, then after at most rank(F) iterations we are done. In addition all the computations that we describe can be performed efficiently. The total running time of the algorithm is poly$(n, \text{rank}(F), |\Sigma|)$. So it remains to show how we construct the hypothesis and how we process the counterexamples.

Constructing a Hypothesis. We need to define a procedure/function h that on input w efficiently computes a value $h(w)$. For every $x_i \in X$ and $\sigma \in \Sigma$ we look at the r-tuple corresponding to the entries of F in row $x_i \circ \sigma$ and columns in Y. Denote this r-tuple by $\widehat{F}_{x_i \circ \sigma}$ (the tuple $\widehat{F}_{x_i \circ \sigma}$ is generated using r MQs). Since $\widehat{F}$ is a full rank matrix we can find (efficiently) coefficients $a_{i,j,\sigma}$ such that

$$\widehat{F}_{x_i \circ \sigma} = \sum_{j=1}^{r} a_{i,j,\sigma} \widehat{F}_{x_j}.$$

Now, given w, we can compute $h(w)$ as follows. We use the coefficients of the above $r \cdot |\Sigma|$ linear dependencies to express (efficiently) every vector $\widehat{F}_w$ as a linear combination of $\widehat{F}_{x_1}, \ldots, \widehat{F}_{x_r}$ (this representation of $\widehat{F}_w$ is *not* necessarily correct). This is done by induction; $w = \varepsilon$ is easy since $x_1 = \varepsilon$. Now, if we already expressed $\widehat{F}_w = \sum_{i=1}^{r} a_i^w \widehat{F}_{x_i}$ then we write $\widehat{F}_{w \circ \sigma} = \sum_{i=1}^{r} a_i^w \widehat{F}_{x_i \circ \sigma}$. Notice that if a_i^w are the "correct" coefficients, i.e., $F_w = \sum_{i=1}^{r} a_i^w F_{x_i}$, then $F_{w \circ \sigma} = \sum_{i=1}^{r} a_i^w F_{x_i \circ \sigma}$ (since $F_{w \circ \sigma, v} = F_{w, \sigma \circ v}$). Using the coefficients $a_{i,j,\sigma}$ we get

$$\widehat{F}_{w \circ \sigma} = \sum_{i=1}^{r} a_i^w \left[\sum_{j=1}^{r} a_{i,j,\sigma} \widehat{F}_{x_j} \right] = \sum_{j=1}^{r} \left[\sum_{i=1}^{r} a_i^w a_{i,j,\sigma} \widehat{F}_{x_j} \right],$$

so the coefficient $a_j^{w \circ \sigma}$ is computed by $\sum_{i=1}^{r} a_i^w a_{i,j,\sigma}$. Finally, for every w, since we can compute $\widehat{F}_w$ in particular we can efficiently compute $h(w) \triangleq \widehat{F}_{w,\varepsilon}$ (note that if $\widehat{F}_w$ is "correct", i.e., $\widehat{F}_w$ indeed contains the correct values as in F, then in particular $\widehat{F}_{w,\varepsilon} = F_{w,\varepsilon} = f(w)$).

Processing a Counterexample. Suppose that z is such that $f(z) \neq h(z)$. We claim that z can be partitioned to $z = w \circ \sigma \circ v$ such that adding a row w to X and a column σy (for some $y \in Y$ that we will find) to Y will increase the rank of $\widehat{F}$ by 1. This partition can be found by considering all prefixes w of z (and in fact can be found more efficiently using the appropriate binary search). To see this, observe that there must exist a prefix w for which the coefficients a_i^w computed by the above algorithm are correct, i.e., $F_w = \sum_{i=1}^{r} a_i^w F_{x_i}$ (this is tested by the algorithm by verifying that $\widehat{F}_w = \sum_{i=1}^{r} a_i^w \widehat{F}_{x_i}$), but for these coefficients $\widehat{F}_{w \circ \sigma} \neq \sum_{i=1}^{r} a_i^w \widehat{F}_{x_i \circ \sigma}$. (For $w = \varepsilon$ the coefficients are certainly correct and if no such w exists it follows, by induction, that the coefficients of $\widehat{F_z}$ are also correct, which implies that $h(z) = f(z)$—a contradiction.) In particular, for such w, there exists $y \in Y$ such that $\widehat{F}_w(\sigma y) \neq \sum_{i=1}^{r} a_i^w \widehat{F}_{x_i}(\sigma y)$. It follows that if we add the column σy to Y, then the row $\widehat{F_w}$ must be independent of the previous rows $\widehat{F}_{x_1}, \ldots, \widehat{F}_{x_r}$, as needed (since if $\widehat{F_w}$ depends on $\widehat{F}_{x_1}, \ldots, \widehat{F}_{x_r}$ the coefficients must be $a_1^w, \ldots, a_r^w$ but adding σy to Y eliminates this possibility).

References

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

[2] P. Auer. On-line learning of rectangles in noisy environments. In *Proc*. 6*th Annual ACM Workshop on Computational Learning Theory*, pages 253–261, 1993.

[3] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. Submitted for publication, 1997. Preliminary version: On the applications of multiplicity automata in learning. In *Proc*. 37*th Annual IEEE Symp*. *on Foundations of Computer Science*, pages 349–358, 1996.

[4] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. On the applications of multiplicity automata in learning. In *Proc*. *the* 37*th Annual IEEE Symp*. *on Foundations of Computer Science*, pages 349–358, 1996.

[5] S. Ben-David, N. H. Bshouty, and E. Kushilevitz. A composition theorem for learning algorithms with applications to geometric concept classes. In *Proc*. 29*th Annual ACM Symp*. *on the Theory of Computing*, pages 324–333, 1997.

[6] F. Bergadano, D. Catalano, and S. Varricchio. Learning sat-*k*-DNF formulas from membership queries. In *Proc*. 28*th Annual ACM Symp*. *on the Theory of Computing*, pages 126–130, 1996.

[7] A. Blum and S. Rudich. Fast learning of *k*-term DNF formulas with queries. *Journal of Computer and System Sciences*, 51(3):367–373, 1995.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik–Chervonenkis dimension. *Journal of the ACM*, 36:929–965, 1989.

[9] N. H. Bshouty. Exact learning via the monotone theory. In *Proc*. 34*th Annual IEEE Symp*. *on Foundations of Computer Science*, pages 302–311, 1993. Journal version: *Information and Computation*, 123(1):146–153, 1995.

[10] N. H. Bshouty. Simple learning algorithms using divide and conquer. In *Proc*. 8*th Annual ACM Workshop on Computational Learning Theory*, pages 447–453, 1995.

[11] N. H. Bshouty, Z. Chen, and S. Homer. On learning discretized geometric concepts. In *Proc*. 35*th Annual IEEE Symp*. *on Foundations of Computer Science*, pages 54–63, 1994.

[12] N. H. Bshouty, P. W. Goldberg, S. A. Goldman, and H. D. Mathias. Exact Learning of Discretized Geometric Concepts. Technical Report WUCS-94-19, Washington University, 1994.

[13] N. H. Bshouty, S. A. Goldman, H. D. Mathias, S. Suri, and H. Tamaki. Noise-tolerant distribution-free learning of general geometric concepts. In *Proc*. 28*th Annual ACM Symp*. *on the Theory of Computing*, pages 151–160, 1996.

[14] Z. Chen and S. Homer. The bounded injury priority method and the learnability of unions of rectangles. *Annals of Pure and Applied Logic*, 77(2):143–168, 1996.

[15] Z. Chen and W. Maass. On-line learning of rectangles and unions of rectangles. *Machine Learning*, 17(2/3):23–50, 1994.

[16] P. W. Goldberg, S. A. Goldman, and H. D. Mathias. Learning unions of boxes with membership and equivalence queries. In *Proc*. 7*th Annual ACM Workshop on Computational Learning Theory*, 1994.

[17] J. C. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In *Proc*. 35*th Annual IEEE Symp*. *on Foundations of Computer Science*, pages 42–53, 1994.

[18] J. C. Jackson. The Harmonic Sieve: A Novel Application of Fourier Analysis to Machine Learning Theory and Practice. Ph.D. thesis, Technical Report CMU-CS-95-184, School of Computer Science, Carnegie Mellon University, 1995.

[19] E. Kushilevitz. A simple algorithm for learning $O(\log n)$-term DNF. *Information Processing Letters*, 61(6):289–292, 1997.

[20] N. Littlestone. Learning when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[21] P. M. Long and M. K. Warmuth. Composite geometric concepts and polynomial predictability. *Information and Computation*, 113(2):230–252, 1994.

[22] W. Maass and G. Turán. On the complexity of learning from counterexamples. In *Proc*. 30*th Annual IEEE Symp*. *on Foundations of Computer Science*, pages 262–273, 1989.

[23] W. Maass and G. Turán. Algorithms and lower bounds for on-line learning of geometrical concepts. *Machine Learning*, 14:251–269, 1994.

[24] W. Maass and M. K. Warmuth. Efficient learning with virtual threshold gates. In *Proc*. 12*th International Conf. on Machine Learning*, pages 378–386. Morgan Kaufmann, San Mateo, CA, 1995.

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