Learning Unions of High Dimensional Boxes over the Reals

Amos Beimel* Eyal Kushilevitz†

Abstract

In [4] an algorithm is presented that exactly learns (using membership queries and equivalence queries) several classes of unions of boxes in high dimension over finite discrete domains. The running time of the algorithm is polynomial in the logarithm of the size of the domain and other parameters of the target function (in particular, the dimension). We go one step further and present a PAC+MQ algorithm whose running time is independent of the size of the domain. Thus, we can learn such classes of boxes over *infinite* domains. Specifically, we learn unions of t disjoint n-dimensional boxes over the reals in time polynomial in n and t, and unions of $O(\log n)$ (possibly intersecting) n-dimensional boxes over the reals in time polynomial in n.

Keywords: Computational learning, PAC learning, Membership Queries, Boxes, Geometric objects.

1 Introduction

The learnability (under various learning models) of geometric concept classes was studied in many papers (e.g., [7, 8, 9, 5]). A particular attention was given to the case of continuous domains of points (i.e., $\Re^n$) and concept classes which are defined as boxes and unions of boxes in this domain (e.g., [7, 12, 15, 11, 2]).

An axes parallel box is a basic geometric object; it can also be thought of as a conjunction of properties of the form "the attribute x_i is in the range between a_i and b_i." Many natural functions can be represented as such boxes and more generally as unions of boxes. The number of attributes which define each box might be big, and so it is desirable to consider high dimensional boxes. Another reason why unions of high dimensional boxes are interesting is that they extend the widely studied class of DNF formulae (which are just unions of boxes over $\{0,1\}^n$). In this work we present a PAC+MQ algorithm that learns several subclasses of unions of (axes parallel) boxes over $\Re^n$: (1) unions of t disjoint n-dimensional boxes over the reals in time polynomial in n and t; and (2) unions of $O(\log n)$ (possibly intersecting) n-dimensional boxes over the reals in time polynomial in n.

Our starting point is that for simple geometric concepts like boxes "close" points are likely to be classified in a similar way. That is, although $\Re$ has infinitely many elements, for any union of boxes there is a partition of the real numbers into a "small" number of intervals such that each interval can be considered as an equivalence class of isomorphic points. (This partition can be found by projecting the boxes on each of the n axes.) A possible strategy for a learning algorithm is therefore to achieve the following two tasks: find the abovementioned partition of the reals into intervals; and learn the target function restricted to a grid defined by taking a representative from each interval.

*Division of Applied Sciences, Harvard University, 40 Oxford St., Cambridge, MA 02138. E-mail: beimel@deas.harvard.edu. http://www.deas.harvard.edu/~beimel. This research was supported by grants ONR-N00014-96-1-0550 and ARO-DAAL-03-92-G0115. Part of this research was done while the author was a Postdoctoral fellow at DIMACS.

†Department of Computer Science, Technion, Haifa 32000, Israel. E-mail: eyalk@cs.technion.ac.il. http://www.cs.technion.ac.il/~eyalk. This research was supported by a GIF fund, by Technion V.P.R. Fund 120-872, by Japan Technion Society Research Fund, and by the fund for the promotion of research at the Technion.

Our algorithm draws a sufficiently large sample of labeled points in $\Re^n$ and efficiently finds a hypothesis consistent with the sample using membership queries. We prove that we can choose the representatives of the intervals to be a subset of the coordinates of the sample points. Thus, we have reduced the problem of learning unions of boxes from an infinite domain to a finite discrete domain, and we can make use of an algorithm from [4], which learns these classes of unions of boxes in high dimension over finite discrete domains (in the exact learning model). The algorithm of [4] simultaneously finds the representatives and learns the function restricted to these representatives using an algorithm of [3], which learns the boxes using the so-called multiplicity automata representation. To analyze the size of the sample needed for our algorithm, we define the class of "interval multiplicity automata" and analyze its VC-dimension.

Our algorithm has some additional useful properties. The only operations it performs on elements from $\Re$ are comparisons and membership queries, and these involve only elements that appear as coordinates of the random examples. Thus, for instance, if all example points are rational and can be described by d bits then the number of bits needed to describe the numbers used by our algorithm is also at most d. Furthermore, our algorithm is strongly polynomial. That is, the number of arithmetic operations in our algorithm is independent of the size of numbers representing the target function and the examples.

It is interesting to note that the main issue in [4] is how to learn subclasses of unions of boxes in time polynomial in the logarithm of the size of the domain (and other parameters of the problem). However, here use the algorithm of [4] not for this property of the algorithm; we rather build on the property that it compresses the domain into a small number of intervals that is independent of the size of the domain. Without this property our transformation would not work. We do not know if there is a "generic" transformation of any algorithm that learns unions of boxes over finite domains into an algorithm that works over *infinite* domains.

There are many results on learning geometric objects (e.g., [9, 5, 7, 15, 16, 14, 4]). These results differ in the learning model employed, the number of dimensions (i.e., constant dimension versus high dimension), the type of domain (i.e., finite discrete domain versus infinite continuous domain), the number of geometric objects, the complexity of the objects (e.g., halfspaces, boxes, polyhedra, and objects defined by more complex curves). In a relevant work by Kwek and Pitt [14], it is shown how to learn the class of polyhedra (i.e., intersection of halfspaces) in high dimension over the reals in the PAC+MQ learning model. The running time of their algorithm depends on the size of the numbers that appear as coordinates in the random examples (as well as other parameters of the target function). They also learn the wider class of unions of disjoint polyhedra; however, in this case the running time and number of membership queries are polynomial in the distance between the polyhedra (not even the log of the distance!). In comparison, our algorithm learns only unions of axis parallel boxes but it is polynomial in all the required parameters (i.e., our algorithm is strongly polynomial).

2 Preliminaries

2.1 Learning Models

We consider learning in the *Probably Approximately Correct (PAC)* model with membership queries [17]. Let $\mathcal{X}$ be some domain, $\mathcal{C}$ be a class of functions from $\mathcal{X}$ to $\{0,1\}$, and $f \in \mathcal{C}$ be a *target* function. There is some unknown (but fixed) probability distribution $\mathcal{D}$ on the input domain $\mathcal{X}$. The learning algorithm is given two input parameters, the error probability ϵ and the confidence probability δ (both in the range $(0, 0.5)$). At any step, the algorithm may ask an *examples oracle* for a random example. The response to such a request is a labeled example $\langle x, f(x)\rangle$, where x is drawn randomly and independently according to the distribution $\mathcal{D}$. The learning algorithm may also query a *membership 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 $f(z)$. Each oracle response takes one time unit. The output of the algorithm is a hypothesis

$h \in \mathcal{H}$ (where $\mathcal{H}$ is some class of hypotheses). There is a basic difference between the examples oracle and the membership oracle. Getting a random example from the examples oracle is a passive action of the algorithm since the example is chosen by the oracle. On the other hand, querying the membership oracle is an active action of the algorithm since it needs to choose an example z such that the value $f(z)$ would give the algorithm some useful information. The classic PAC model gives access only to the examples oracle. We consider its extention, the PAC+MQ model (defined in [17] and studied extensively in the literature), that gives access to both oracles.

The error of a hypothesis h relative to a target function f, with respect to the distribution $\mathcal{D}$, is $\mathrm{ERROR}(h) \stackrel{\mathrm{def}}{=} \Pr[f(x) \neq h(x)]$, where the probability is taken over the input x chosen according to the distribution $\mathcal{D}$. A learning algorithm *learns* a class of functions $\mathcal{C}$ using a class of hypotheses $\mathcal{H}$, if for every function $f \in \mathcal{C}$, for every distribution $\mathcal{D}$, and for all $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, the learning algorithm outputs a hypothesis $h \in \mathcal{H}$ such that: $\Pr[\,\mathrm{ERROR}(h) \geq \epsilon\,] \leq \delta$. This probability is taken over the random examples drawn by the examples oracle according to the distribution $\mathcal{D}$. We require the algorithm to be efficient; i.e., its running time is polynomial in the "size" of a shortest representation of f,[1] and in $1/\epsilon$ and $1/\delta$. Notice that the distribution $\mathcal{D}$ has a dual role; it is used both for choosing the random examples and for measuring the error of the output hypothesis.

In this paper the domain $\mathcal{X}$ is $\Re^n$. We consider the common computation model of real numbers, e.g., [10, 6]. This model stores a single real number in a single memory location, and charges one unit of computation time for each basic arithmetic operation on two real numbers (e.g., comparison, addition, multiplication, or division).

We also need the definition of the *exact learning* model [1]: in this model the learning algorithm has access to a membership queries oracle (as above) and in addition to an equivalence queries oracle (EQ). An input to the equivalence query oracle is a function h (a hypothesis) and the output is either YES, meaning that h is equivalent to f, or NO together with a counterexample (i.e., y such that $h(y) \neq f(y)$). The goal of the learning algorithm in this model is to *exactly* identify the target function f; i.e., to find h such that $h \equiv f$.

2.2 VC-dimension

Next, we define the Vapnik-Chervonenkis (VC) dimension of a class of functions. This is a combinatorial measure for the "complexity" of a class which in particular characterizes the sample complexity required for an algorithm to learn a class.

Definition 2.1 [VC-dimension] *Let $\mathcal{F}$ be a class of functions from $\mathcal{X}$ to $\{0,1\}$. A set $S \subseteq \mathcal{X}$ is shattered by the class $\mathcal{F}$ if for every function $g : S \to \{0,1\}$ there is a function $f \in \mathcal{F}$ such that $g(x) = f(x)$ for every $x \in S$. (That is, $\mathcal{F}$ has all possible behaviors on S.) The* Vapnik-Chervonenkis (VC) dimension *of a class $\mathcal{F}$ is the maximum size of a set $S \subseteq \mathcal{X}$ that is shattered by $\mathcal{F}$.*

The following fundamental theorem is due to Blumer et al. [7]:

Theorem 2.2 [7] *Let $\mathcal{C}$ be a class of concepts, and $\mathcal{H}$ be a class of hypotheses of VC-dimension d. Let $\mathcal{B}$ be an algorithm that takes as input a set S of points labeled by some concept in $\mathcal{C}$, may ask polynomially-many membership queries, and produces as a output a hypothesis $h \in \mathcal{H}$ that is consistent with S (that is, $h(x) = f(x)$ for every $x \in S$). Then, for some constant $c_0 > 0$, algorithm $\mathcal{B}$ is a PAC+MQ learning algorithm for the class $\mathcal{C}$ provided that the number of random examples given to $\mathcal{B}$ is at least*

$$c_0 \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{d}{\epsilon} \log \frac{1}{\epsilon} \right).$$

[1] We consider some representation scheme of the functions in $\mathcal{C}$ and assume that there is a "natural" size function that maps representations to non-negative integers. For a more formal treatment of this point see [13, Chapter 1.2.2].

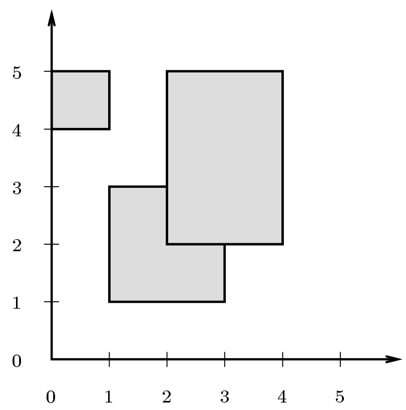

Figure 1: A simple example of a union of 2-dimensional boxes.

2.3 Classes of Boxes

Let L be an ordered set (the typical sets we consider are the real numbers, $\Re$, and the discrete domain $\{0, 1, \ldots, \ell - 1\}$ for some integer ℓ). An n-dimensional box in L^n is defined by two corners $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ (in L^n) as follows:

$$B_{a_1,\ldots,a_n,b_1,\ldots,b_n} = \{(x_1, \ldots, x_n) \in L^n \ : \ \forall i, \ a_i \leq x_i \leq b_i \ \}.$$

It is possible that $a_i = b_i$ for some indices i. We view such a box as a boolean function that gives 1 for every point in L^n which is inside the box and 0 for every point outside the box. More generally, the boolean function corresponding to the *union* of (n-dimensional) boxes $B_1, \ldots, B_t$ is defined to be 1 for every point in L^n which is inside (at least) one of the t boxes and 0 otherwise. In Figure 1 we show a simple example of union of three 2-dimensional boxes over L^2, where $L = \{0, 1, 2, 3, 4, 5\}$.

3 The VC-Dimension of Interval Multiplicity Automata

Theorem 2.2 implies that the sample complexity (and hence the time complexity) of a learning algorithm is related to the VC-dimension of the hypothesis class used by the learning algorithm. In this section we define the class of *interval multiplicity automata* and analyze its VC-dimension. These are the hypotheses used by our algorithm (Algorithm LEARN_BOXES described in Figure 3). A multiplicity automata is a generalization of deterministic finite automata, and an interval multiplicity automata extends the function computed by the automata to infinite domains.

We start with the definition of multiplicity automaton over GF(2).[2] There are several (equivalent) ways to define multiplicity automata. In this paper we view these automata as non-deterministic automata in which the acceptance criterion is changed; a word is accepted by such automaton if the number of accepting paths is odd.

Definition 3.1 [Multiplicity Automata] *A* multiplicity automaton *A is a 5-tuple $\langle Q, \Sigma, \delta, q_0, F\rangle$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. A path $p_0, p_1, \ldots, p_n$ is an accepting path for an input $w = (w_1, w_2, \ldots, w_n)$ if $p_0 = q_0$, for every $i = 1, \ldots, n$ it holds that $p_i \in \delta(p_{i-1}, w_i)$, and $p_n \in F$. The*

[2] Multiplicity automata can be defined over every field; see, e.g., [3]. In the rest of the paper whenever we refer to a multiplicity automaton it is over the field GF(2).

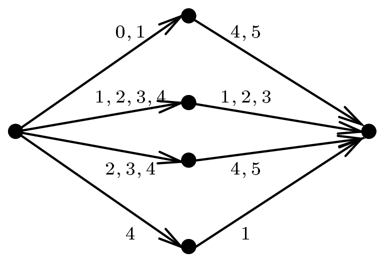

Figure 2: A simple example of a multiplicity automaton.

automaton A computes a function $f_A : \Sigma^* \to \{0, 1\}$, where $f_A(w) = 1$ if the number of accepting paths for w is odd and $f_A(w) = 0$ otherwise.[3]

In Figure 2, a simple multiplicity automaton is shown. It can be verified that this automaton computes the union of boxes described in Figure 1. For example, there is exactly one accepting path for the point $(1, 2)$, thus, $f_A(1, 2) = 1$. On the other hand, there are two accepting paths for the point $(4, 1)$, thus, $f_A(4, 1) = 0$.

It is not clear how to use such automata for, say, real numbers because there are infinitely many numbers. However, when we look at boxes we expect "close" points to behave in a similar way. Therefore, we can partition the reals into a small number of intervals and view each such interval as a single symbol. This intuition is used in the following definition:

Definition 3.2 [Interval Multiplicity Automata] *An* interval multiplicity automaton *is a pair $\langle A, C\rangle$ where A is a multiplicity automaton and C is a set. The alphabet of the automaton is $\Sigma = \{0, ..., \ell - 1\}$. The set $C = \{c_0, c_1, ..., c_{\ell-1}\}$ contains ℓ elements where $c_0 = -\infty$, the elements $c_1, ..., c_{\ell-1}$ are in $\Re$ and $c_0 < c_1 < c_2 < ... < c_{\ell-1}$. The index of an element $a \in \Re$, denoted by $\mathrm{ind}_C(a)$, is $\max\{i : c_i \leq a\}$. The function $f_{\langle A,C\rangle}$ computed by $\langle A, C\rangle$ is defined as follows: $f_{\langle A,C\rangle}$ on an input $(x_1, \ldots, x_n) \in \Re^*$ equals $f_A(\mathrm{ind}_C(x_1), \mathrm{ind}_C(x_2), \ldots, \mathrm{ind}_C(x_n))$; i.e., the value of the automaton on the indices of the n elements of the input.*

Notice that in the above definition the size of C is equal to the size of the alphabet of A. We prove that the VC-dimension of the class of interval multiplicity automata is small.

Lemma 3.3 *Let $\mathcal{F}$ be the class of interval multiplicity automata that accept only words of length n, have alphabet of size ℓ, and have at most r states. Then, the VC-dimension of $\mathcal{F}$ is $O(\ell(\log \ell + r^2))$.*

Proof: Let $S \subseteq \Re^n$ be a set of size m that is shattered by $\mathcal{F}$. We will show, by a simple counting argument, that if m is "too big" then it is not possible to obtain all the 2^m functions $g : S \to \{0, 1\}$.

Let $V = \{v_1, v_2, \ldots, v_k\}$ be the elements in $\Re$ that appear as coordinates in the points of S, where $v_1 < v_1 < \ldots < v_k$. There are at most nm elements in V, that is, $k \leq mn$. The main observation is that if we fix any multiplicity automaton $A \in \mathcal{F}$, and go over all possible sets C, there cannot be too many behaviors of $\langle A, C\rangle$ on S: First observe that for such an automaton A the value of $f_{\langle A,C\rangle}$ on the points of S does not depend on the exact values of the elements in C but rather on the set $\{\mathrm{ind}_C(v) : v \in V\}$. These values are determined by the number of elements in C that are in each of the $k + 2$ intervals $(-\infty, v_1), [v_1, v_2), \ldots, [v_{k-1}, v_k), [v_k, \infty)$. There are less than $(k+2)^\ell \leq (mn+2)^\ell$ ways to partition C into the k intervals. Thus, for a fixed A there are at most $(mn+2)^\ell$ behaviors of interval multiplicity automata $\langle A, C\rangle$ on S.

[3]The function f_A is the characteristic function of the language accepted by the multiplicity automaton.

The number of multiplicity automata with r states and an alphabet of size ℓ is $2^{r+\ell\cdot r^2}$ (since we can represent such automaton by ℓ boolean matrices of size $r \times r$ describing the transition function δ and an additional vector of size r describing the set of final states F). Furthermore, the number of states of A, i.e. r, is greater that n (since the automaton accepts only words of length n). All together, the number of behaviors of interval multiplicity automata on S is at most

$$(mn+2)^{\ell} \cdot 2^{O(\ell\cdot r^2)} \quad = \quad 2^{O(\ell(\log m + r^2))}.$$

Finally, if S is shattered by $\mathcal{F}$, then there are 2^m behaviors of interval multiplicity automata on S (where $m = |S|$). Thus, $m = O(\ell(\log m + r^2))$, which implies $m = O(\ell(\log \ell + r^2))$. □

4 The Learning Algorithm

In this section we present our algorithm, Algorithm LEARN_BOXES, which learns unions of n-dimensional boxes over infinite domains. The algorithm gets a labeled sample and uses the algorithm LEARN_ SENSITIVE of [4] to produce a hypothesis consistent with the sample. We then extend the hypothesis to $\Re^n$ in a way that the VC-dimension of the extended hypotheses is small. The details of LEARN_ SENSITIVE are not important for understanding LEARN_BOXES; the reader need only know that this is an *exact learning* algorithm that efficiently learns certain classes of unions of n-dimensional boxes over a discrete domain and uses interval multiplicity automata as its hypotheses.[4]

We first explain how to project a function to a discrete domain. Given a function $f : \Re^n \to \{0,1\}$ and a (sorted) set $C = \{-\infty, c_1, ..., c_{\ell-1}\}$ we define $f_C : \{0, 1, \ldots, \ell-1\} \to \{0,1\}$ as $f_C(\sigma_1, \ldots, \sigma_n) \stackrel{\text{def}}{=} f(c_{\sigma_1}, \ldots, c_{\sigma_n})$. If there is some i such that $\sigma_i = 0$ then $f_C(\sigma_1, \ldots, \sigma_n) \stackrel{\text{def}}{=} 0$.

Algorithm LEARN_BOXES:
Input: A sample S of labeled points in $\Re^n$
Output: An interval multiplicity automaton consistent with the sample.

1. Let V be the elements in $\Re$ that appear as coordinates in the points of S and $\{-\infty\}$.
2. Learn the function f_V using Algorithm LEARN_ SENSITIVE (from [4]).
   (a) To answer a membership query about $f_V(\sigma_1, \ldots, \sigma_n)$:
      - If there is some i such that $\sigma_i = 0$ then answer 0.
      - Otherwise, use the oracle for f and answer $f(v_{\sigma_1}, \ldots, v_{\sigma_n})$.

   (b) To simulate an equivalence query EQ(h_V) to f_V:
      - Define $h : \Re^n \to \{0,1\}$ as $h(x_1, \ldots, x_n) = h_V(\text{ind}_V(x_1), \ldots, \text{ind}_V(x_n))$.
        If h is equal to f on all the points in S then halt with the hypothesis h.
      - Otherwise, we get a counterexample $(y_1, \ldots, y_n) \in S$.
        Pass $(\text{ind}_V(y_1), \ldots, \text{ind}_V(y_n))$ as a counterexample to LEARN_ SENSITIVE and continue its execution.

Figure 3: An algorithm for learning high dimensional boxes over the reals.

The learning algorithm – Algorithm LEARN_BOXES – is described in Figure 3. Next, we prove its correctness.

[4]In fact in [4] the class of hypotheses used was not explicitely defined; the definition of *interval multiplicity automata* given in the current paper is new. Also note that [4] deals with a more restricted domain (which is large, yet finite).

Lemma 4.1 *Let S be a set of labeled examples $\Re^n$, and V be the elements in $\Re$ that appear as coordinates in the points of S. Algorithm* LEARN_BOXES *produces a hypothesis which is consistent with the sample set S. The running time of the algorithm is $\mathrm{poly}(n, |S|, r)$, where r is the number of states in the smallest multiplicity automaton for f_V (i.e., f restricted to V^n).*

Proof: We first claim that if $(y_1, \ldots, y_n)$ is a counterexample for f (with respect to hypothesis h), then $(\mathrm{ind}_V(y_1), \ldots, \mathrm{ind}_V(y_n))$ is a counterexample for f_V (with respect to hypothesis h_V). This is implied by

$$f_V(\mathrm{ind}_V(y_1), \ldots, \mathrm{ind}_V(y_n)) \;=\; f(y_1, \ldots, y_n) \;\neq\; h(y_1, \ldots, y_n) \;=\; h_V(\mathrm{ind}_V(y_1), \ldots, \mathrm{ind}_V(y_n)).$$

The first equality follows from the definition of f_V and the fact that $y_i \in V$ for every i, the second inequality follows from the fact that $(y_1, \ldots, y_n)$ is a counterexample for f, and the third equality follows from the definition of h in Step (2b) of the algorithm.

Algorithm LEARN_ SENSITIVE of [4] is an exact learning algorithm which is guaranteed that after asking $\mathrm{poly}(n, |S|, r)$ equivalence queries it finds a hypothesis that is exactly equivalent to the function f_V. We have shown that every counterexample that Algorithm LEARN_BOXES passes to LEARN_ SENSITIVE in response to an equivalence query is indeed a counterexample for f_V. Therefore, the only possibility that Algorithm LEARN_ SENSITIVE is not given enough counterexamples to learn f_V is if the hypothesis is already equivalent to the target function on all the sample points. In both cases the hypothesis is consistent on S with the target function f. That is, after at most $\mathrm{poly}(n, |S|, r)$ counterexamples Algorithm LEARN_BOXES passes to it, Algorithm LEARN_ SENSITIVE finds an interval multiplicity automaton equivalent to f_V. At this stage no additional counterexamples to f can be found in $S \subseteq V^n$, and Algorithm LEARN_BOXES terminates. The overhead of Algorithm LEARN_BOXES is the simulation of the equivalence queries, each one requires evaluating the hypothesis on the points of the sample S and comparing it to the label of the point. Each evaluation of the hypothesis is efficient; i.e., it requires time $\mathrm{poly}(n, |S|, r)$. All together the running time is $\mathrm{poly}(n, |S|, r)$. $\square$

Lemma 4.1 guarantees that if the target function f has a small multiplicity automaton then Algorithm LEARN_BOXES is an efficient algorithm that produces a hypothesis consistent with the sample. This is still not enough to guarantee learning since the hypotheses in Algorithm LEARN_BOXES depend on all elements of V (and therefore, when learning the class of small multiplicity automata, the VC-dimension of the hypothesis class is too big). However, in our case the target function f_V is a union of boxes and so the hypothesis h_V depends only on a small subset of $\{0, \ldots, \ell - 1\}$ and h is an interval multiplicity automaton with a small alphabet.

Theorem 4.2 *Algorithm* LEARN_BOXES *described in Figure 3 learns the class of unions of t disjoint boxes in $\Re^n$ in time $\mathrm{poly}(n, t, \frac{1}{\epsilon}, \log \frac{1}{\delta})$, provided its input is a random sample given by the examples oracle whose size is*

$$\Theta\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n^3 t^3}{\epsilon}\log\frac{1}{\epsilon}\right).$$

Algorithm LEARN_BOXES *also learns the class of unions of t (possibly intersecting) boxes in $\Re^n$ using*

$$\Theta\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n^3 t 2^{2t}}{\epsilon}\log\frac{1}{\epsilon}\right)$$

examples and its running time is $\mathrm{poly}(n, 2^t, \frac{1}{\epsilon}, \log \frac{1}{\delta})$ (e.g., for $t = O(\log n)$ the number of examples and running time is $\mathrm{poly}(n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$).

Proof: Consider the interval multiplicity automata produced as hypotheses by LEARN_ SENSITIVE. These automata use $\ell = O(nt)$ letters for both classes, $r = O(tn)$ states for unions of t disjoint boxes and $r = O(n2^t)$ states for unions of t boxes. By Lemma 3.3, the VC-dimension of this hypothesis class is $O(n^3t^3)$ for unions of t disjoint boxes and $O(n^3t2^{2t})$ for unions of t boxes. By Lemma 4.1, the hypothesis produced by Algorithm LEARN_BOXES is consistent with the sample. Thus, by Theorem 2.2, a sample size as in the theorem suffices. The running time analysis is according to Lemma 4.1. $\square$

Remark 4.3 The sample points in the algorithm are n-tuples in $\Re^n$. The only operations performed on real numbers in our algorithm are comparisons, and making membership queries (on points from V^n). Thus, if all sample points are rational then all queries are on rational points as well. Furthermore, Algorithm LEARN_BOXES can learn the same classes of boxes over any ordered domain.

References

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

[2] P. Auer, P. M. Long, and A. Srinivasan. Approximating hyper-rectangles: learning and pseudorandom sets. In *Proc. of 29th Symp. on the Theory of Computing*, pages 314–323, 1997. Journal version: *J. of Computer and System Sciences*, 57(3):376–388, 1998.

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

[4] A. Beimel and E. Kushilevitz. Learning boxes in high dimension. In S. Ben-David, editor, *3rd European Conf. on Computational Learning Theory (EuroCOLT '97)*, volume 1208 of *Lecture Notes in Artificial Intelligence*, pages 3–15. Springer, 1997. Journal version: *Algorithmica*, 22(1/2):76–90, 1998.

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

[6] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer, 1998.

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

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

[9] 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. of 28th Symp. on the Theory of Computing*, pages 151–160, 1996.

[10] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory.* Springer, 1997.

[11] M. Frazier, S. Goldman, N. Mishra, and L. Pitt. Learning from a consistently ignorant teacher. *J. of Computer and System Sciences*, 52(3):471–492, 1996.

[12] M. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proc. of 25th Symp. on the Theory of Computing*, pages 392–401, 1993. Journal version: *J. of the ACM*, 45(6):983–1006, 1998.

[13] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory.* MIT press, 1994.

[14] S. Kwek and L. Pitt. PAC learning intersections of halfspaces with membership queries. In *Proc. of 9th Conf. on Comput. Learning Theory*, pages 244–254, 1996. Journal version: *Algorithmica*, 22(1/2):53-75, 1998.

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

[16] W. Maass and M. K. Warmuth. Efficient learning with virtual threshold gates. *Information and Computation*, 141(1):66–83, 1998.

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