Learning Boolean Functions in an Infinite Attribute Space

Avrim Blum
School of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

This paper presents a theoretical model for learning Boolean functions in domains having a large, potentially infinite number of attributes. The model allows an algorithm to employ a rich vocabulary to describe the objects it encounters in the world without necessarily incurring time and space penalties so long as each *individual* object is relatively simple. We show that many of the basic Boolean functions learnable in standard theoretical models, such as conjunctions, disjunctions, K-CNF, and K-DNF, are still learnable in the new model, though by algorithms no longer quite so trivial as before. The new model forces algorithms for such classes to act in a manner that appears more natural for many learning scenarios.

Keywords: Theoretical models, concept learning, irrelevant attributes, large attribute spaces.

Running head: *Learning in an Infinite Attribute Space*

Email: avrim@theory.cs.cmu.edu.

1 Introduction

Suppose we planned to send a learning machine into a new, never-explored environment to learn about the objects it encounters. Since we may not know a priori what such objects would look like and what sort of features they might have, we would want to equip the machine with a large vocabulary of descriptive terms or *attributes* to use to describe the things it finds. We would like, however, not to be penalized for the large space of possible attributes if it happens that relatively few of these pertain to the objects actually seen. In addition, we would like to discourage the use of learning methods that involve creating a list of all possible descriptive terms and then deleting the ones deemed unnecessary.

In the standard theoretical models for learning Boolean functions from examples (Valiant, 1984) (Kearns et al., 1987) (Littlestone, 1989), we imagine there are n attributes that our algorithm is aware of and we are trying to learn some Boolean function f of these attributes. An example is some $\mathbf{x} \in \{0,1\}^n$ with the interpretation that the ith component of $\mathbf{x}$ is 1 if $\mathbf{x}$ has the ith attribute. The learning algorithm gains information about f by seeing examples $\mathbf{x}$ and their labels $f(\mathbf{x})$. The standard models differ in how examples are chosen and how successful learning is defined, but one feature they have in common is that the learning algorithm is allowed time and space polynomial in n and the description length of f.

A problem with this basic model is that representing each example as a bit vector is wasteful when the total number of possible attributes is much larger than the number of attributes any one example has. Suppose, for instance, we want to teach a machine what a pencil is by presenting to it examples of pencils and non-pencils. We *could* hard-wire into the machine only those attributes which appear in pencils, but then the machine would have trouble if we later wished to teach it some other concept such as "cup" or "book". Instead, we would like to provide the machine a broad set of possible attributes, expecting that few of these will occur in any *individual* object, though any one might potentially prove useful. For example, we might choose the set of attributes to be all English verbs and adjectives. A typical pencil might then have the attributes: orange, wooden, long, hexagonal, writes, and erases. One could likely think of several more one-word attributes that apply, but the total number of attributes the pencil does have is much smaller than the number it does *not* have: not only obvious ones such as it is not green or blue or chartreuse, but also the more absurd, such as that it does not fly or eat or breathe. One would rather not have to explicitly state whether or not each verb or adjective applies in order to describe a pencil, which is what one must do in the standard models. Another similar situation would be learning about research papers based on keywords. Here, each paper might be succinctly described by a list of the keywords it has, but standard models would require representing each paper as a bit vector over the space of all potential keywords.

In the model proposed in this paper, instead of representing each example as a bit vector, we represent it as simply a list of the attributes the example *has*, leaving unmentioned all attributes the example does *not* have. Since the learning machine no longer needs time proportional to the size of its vocabulary just to read in each example, we are now free to make the attribute space arbitrarily large and weaken the learning machine by allowing it only time polynomial in the size of the longest example seen (and the size of the target function) to do its learning. Thus, this model differs most from the standard ones in situations where each example has as attributes a very small fraction of the total attribute space.

What makes the model interesting is that even though the attribute space may be so large that the learning machine does not have time even to look through all the words in its "dictionary", we can still find algorithms to learn many of the basic learnable Boolean functions such as conjunctions, disjunctions, K-CNF, and K-DNF. Also, because the model no longer allows the learner to learn essentially by elimination, the algorithms are forced to act in what seems to be a more realistic manner. In addition, we can combine these techniques with the thresholding algorithms of Littlestone (1987), to learn especially quickly when the number of attributes relevant to the target function is much less than the size of the examples.

2 The proposed model

The domain of examples is defined by an attribute space $A \subseteq \Sigma^*$ whose elements are called *attributes*. The attribute space might be infinite (eg. $A = \Sigma^*$) or just large (eg. $A = \{\text{all English verbs and adjectives}\}$). For simplicity, we will name the attributes $a_1, a_2, a_3, \ldots$. An *example* or *instance* $\mathbf{x}$ is a finite subset of A and we define the instance space X to be the set of all possible examples: that is, the set of all finite subsets of A. An example will be presented to the learner as a list of its attributes; this list will have finite length since examples are finite sets. For $\mathbf{x}$ some finite subset of A, let the *size* of $\mathbf{x}$ be the number of attributes in $\mathbf{x}$, and the *length* of $\mathbf{x}$ be the length of the string formed by concatenating together the attributes of $\mathbf{x}$.

A concept is a Boolean function on X that depends on only a finite number of attributes. Formally, define a *concept* f to be a function from X to $\{0, 1\}$ such that for some minimal finite set $R_f \subseteq A$ of *relevant* attributes, $f(\mathbf{x}) = f(\mathbf{x} \cap R_f)$ for all examples $\mathbf{x}$. In other words, the value of f is determined by only the relevant attributes in $\mathbf{x}$. We call examples on which f is 1, *positive examples* of f and those on which f is 0, *negative examples* of f. A *concept class* is simply a collection of concepts.

In the learning scenario, we are given a concept class C and there is some unknown *target concept* $f_T \in C$ that we are trying to learn. (Throughout this paper, we will use f_T to denote the target concept, f to denote a generic concept, and h the current hypothesis of the learning algorithm when one exists.) Learning proceeds in a sequence of *stages*. In each stage, we are given some example $\mathbf{x}$, are asked to predict the value $f_T(\mathbf{x})$ and then are told whether our prediction was correct. In this paper we will use a mistake-bound model to describe how examples are chosen and how to define successful learning.

In the mistake-bound model, for a given concept class C and unknown target concept $f_T \in C$, an adversary chooses examples from X in any order it wishes to present during the learning stages. Each time the learning algorithm makes an incorrect prediction, we charge it one *mistake*. Let $\mathbf{size}(f_T)$ be some natural measure of the description length of f_T, and let n and $\tilde{n}$ be the maximum size and maximum length respectively of any example seen so far. We say that we *learn* C if for all $f_T \in C$ we can guarantee to make a number of mistakes at most some fixed polynomial in $\mathbf{size}(f_T)$ and n, using time in each stage polynomial in $\mathbf{size}(f_T)$ and $\tilde{n}$. (The use of $\tilde{n}$ here is simply to allow the algorithm time enough to read in the examples in case the attribute names happen to be long.) Note that this definition differs from the standard model where n refers to the total number of attributes in the attribute space and not just the size of the largest example seen.

For some of the learning algorithms, it will be convenient to assume that $\mathbf{size}(f_T)$ and n are known by the learner beforehand. These assumptions can be removed by standard "doubling" techniques in which, for instance, the learner's hypothesized value of $\mathbf{size}(f_T)$ is doubled when a certain mistake bound is exceeded.

Note that by the definition of "relevant," attributes that are missing from an example $\mathbf{x}$ can be considered philosophically as either actually missing from the object seen or else as unknown but not relevant. For instance, if "purple" is not relevant to f_T, and if $\mathbf{x}$ does not contain the attribute purple, then $f_T(\mathbf{x}) = f_T(\mathbf{x} \cup \{\text{purple}\})$ so for the purposes of learning it does not matter whether the "actual object seen" really might have been purple.

3 Learning algorithms

To illustrate the difference between the new model and the standard mistake-bound model, let us first consider learning the class of monotone disjunctions: that is, functions of the form $a_1 \vee a_2 \vee a_4$ but not $a_1 \vee a_2 \vee \overline{a_3}$. The typical algorithm for learning such concepts proceeds as follows. We keep at all times a *current hypothesis* h which is used to make predictions and begin by initializing h to the disjunction of all attributes in the space: $a_1 \vee \ldots \vee a_n$. When we make a mistake on a negative example, we simply remove from h all attributes present in that example. Since we only remove terms on negative examples, we never remove any terms that are present in the target concept (that is, we never remove any relevant attributes). Since we remove at least one term on every mistake, we can make at most n mistakes.

Clearly, this simple procedure will no longer work in the new model. In an infinite attribute space we will go on forever just using the negative examples to cross out the irrelevant attributes. Instead we will need to use both the positive and the negative examples to control more carefully the part of the space under consideration. A procedure that works for the new model is as follows.

LEARN-MONOTONE-DISJUNCTION

1. Predict "negative" until we see our first positive example $\mathbf{x}$. At this point, initialize h to be the disjunction of all attributes that appear in $\mathbf{x}$; h will be used to make all future predictions.

2. If we make a mistake on a positive example $\mathbf{x}$, add onto h all attributes in $\mathbf{x}$. Notice that since $\mathbf{x}$ is a positive example, it must contain some attribute appearing in f_T (the target concept) and since we mistakenly predicted "negative", it has no attributes in h. So, we add to h at least 1 attribute from f_T and at most $n-1$ attributes not from f_T.

3. If we make a mistake on a negative example $\mathbf{x}$, remove from h all attributes in $\mathbf{x}$. Since we mistakenly predicted "positive", we are removing at least 1 attribute from h and since $\mathbf{x}$ is negative, we are removing 0 attributes that appear in f_T.

On monotone disjunctions of r attributes, the above procedure makes at most r mistakes on positive examples and $r(n-1)$ mistakes on negative examples for a total of rn mistakes maximum.

3.1 Monotone and non-monotone K-CNF

We now turn to the problem of learning monotone K-CNF formulas. For instance, for $K = 3$, f_T might be $(a_1 \vee a_2 \vee a_5)(a_2 \vee a_3 \vee a_4)$. In the typical algorithm for this problem, we begin by setting h to be the conjunction of all clauses of size K over the attribute space. Each time a positive example is seen, we keep only the clauses consistent with that example and discard the rest. This algorithm makes no mistakes on negative examples.

In addition to the problem of initialization, the above algorithm will no longer work in the infinite-attribute model because there may possibly be an infinite number of clauses of size K that are consistent with any given positive example. Instead, we will use a procedure that "grows" each of the clauses out of "seeds" of size one and discards them if they get too large.

Define a *seed* of a monotone clause to be a disjunction of a subset of the attributes disjoined in the clause. So, for instance, a_1 and $a_1 \vee a_5$ are both seeds of the clause $a_1 \vee a_2 \vee a_5$. If an example satisfies the seed of a clause, then it satisfies the clause as well. In addition, seeds have the following property:

> If c is a seed of clause c_T, and example $\mathbf{x}$ satisfies c_T but not c, then $\mathbf{x}$ has at least one attribute in c_T that is not in c. $(*)$

The procedure below learns monotone K-CNF functions in the infinite-attribute model.

LEARN-MONOTONE-K-CNF

1. Predict "negative" until we see our first positive example $\mathbf{x}$. At this point, initialize h to the conjunction of all attributes appearing in $\mathbf{x}$. So, h is now a 1-CNF. Notice that h contains at least one seed for each clause in f_T, since any positive example has at least one attribute from every clause.

2. If we make a mistake on a positive example $\mathbf{x}$, consider each clause c in h *not* satisfied by the example. If c contains K attributes, discard it. Otherwise (it contains fewer literals), replace it with at most n new clauses: for each attribute in $\mathbf{x}$, create a new clause consisting of the disjunction of that attribute with the literals in c.

   Thus, for example, if h were $(a_1)(a_4)$ and we saw the positive example $\{a_3, a_4, a_5\}$, we would change h to equal $(a_1 \vee a_3)(a_1 \vee a_4)(a_1 \vee a_5)(a_4)$. (Actually, we do not need to include the clause $(a_1 \vee a_4)$ since it is implied by the clause (a_4)).

Theorem 1 *LEARN-MONOTONE-K-CNF makes at most* $(n+1)^K$ *mistakes on any monotone K-CNF formula.* (*Recall that n is the size of the largest example seen*).

Proof: LEARN-MONOTONE-K-CNF maintains in step 2 the invariant that every clause in f_T has some seed in h. The reason is that if clause c is the only seed in h of some clause c_T in f_T and is modified in step 2, then by property $(*)$ of seeds, at least one of the new clauses created in step 2 will also be a seed of c_T. This invariant implies that LEARN-MONOTONE-K-CNF never makes any mistakes on negative examples. (An example that satisfied h would satisfy a seed for each clause in f_T and therefore would satisfy f_T).

For each clause in h of m attributes place a "cost" of $(n+1)^{K-m}$ and let the *total cost* of h be the sum of the costs of all its clauses. Thus, we begin with a total cost of at most $n(n+1)^{K-1} < (n+1)^K$. Each time we replace a clause by up to n clauses of size one larger, we decrease the total cost by at least 1 since for $m < K$, we have $(n+1)^{K-m} \geq 1 + n(n+1)^{K-(m+1)}$. Also, each time we throw out a clause we decrease the total cost by 1. Thus, on each mistake, we decrease the cost by at least 1, and since the cost is never negative, the algorithm makes less than $(n+1)^K$ mistakes total.

The running time of this algorithm is clearly polynomial in $\mathbf{size}(f_T)$ and the length of the longest example seen. ∎

So far, we have considered only monotone functions. It may be, however, that we wish to learn a concept that depends on some attribute *not* being present—for instance, for the concept "penguin", it may be important that the bird *not* fly. In the standard model, any algorithm that learns a monotone function can be used to learn the non-monotone version as well because a non-monotone function can be thought of as a monotone one over the larger attribute space that has a_i and $\overline{a_i}$ as two different attributes for each original attribute a_i. It will just so happen that every example seen will have as an attribute exactly one of a_i and $\overline{a_i}$.

In the new model this transformation no longer works. Making $\overline{a_i}$ a new attribute for all i would cause each example to have a possibly infinite size. We will see, however, that by improving the algorithms, we *can* learn non-monotone K-CNF formulas—functions that depend on an attribute not being present—even though the examples are presented to us as a list of their positive attributes only. As intuition, if the concept "penguin" depended on the absence of the attribute "can fly", then even though this absence is never explicitly mentioned in the examples, we might still infer its importance through the *negative* examples that seem as though they ought to be penguins except for the fact that they *do* fly. In addition, we can use the algorithm that learns non-monotone K-CNF to learn non-monotone disjunctions and K-DNF functions f as well, by simply having it learn the K-CNF that is the complement of f, since we no longer need worry about monotonicity.

The main idea for general (non-monotone) K-CNF is that we will use the first positive example to give us the seeds for each monotone clause (as before), but then we will allow the possibility of making mistakes on negative examples and will use them to provide the seeds for the non-monotone clauses. The only additional complication is that we must modify the definition of "seed" for non-monotone clauses so that seeds still satisfy property $(*)$. The "seed = subset" definition above will no longer work because, for instance, example $\{a_2\}$ satisfies $c_T = (a_1 \vee \overline{a_2} \vee \overline{a_3})$ and not $c = (a_1 \vee \overline{a_2})$ but does not have any *attribute* in c_T that is not in c. The new definition is as follows.

Definition 1 *A* seed *of a clause is a disjunction of all negated attributes of the clause and any subset of the non-negated ones.*

For instance, any seed of the clause $(a_1 \vee \overline{a_2} \vee \overline{a_3})$ must contain $\overline{a_2}$ and $\overline{a_3}$. The new definition of "seed" reduces to the old one when all clauses are monotone, and satisfies property $(*)$. The procedure below learns general (non-monotone) K-CNF.

LEARN-K-CNF

1. Same as LEARN-MONOTONE-K-CNF. Predict "negative" until we see our first positive example $\mathbf{x}$ and then initialize h to the conjunction of all the attributes in $\mathbf{x}$.

2. Same as LEARN-MONOTONE-K-CNF. If we make a mistake on a positive example $\mathbf{x}$, for each clause c in h *not* satisfied by the example, do the following. Discard c, but if c contained fewer than K literals, then for each attribute in $\mathbf{x}$, create a new clause consisting of the disjunction of that attribute with the literals in c.

3. If we make a mistake on a negative example $\mathbf{x}$, then for each subset $\{a_{i_1}, \ldots, a_{i_r}\} \subseteq \mathbf{x}$ $(r \leq K)$ of at most K attributes of $\mathbf{x}$, add to h the clause $(\overline{a_{i_1}} \vee \ldots \vee \overline{a_{i_r}})$.

Theorem 2 *The procedure LEARN-K-CNF makes at most $(rK+1)(n+1)^K$ mistakes on any K-CNF formula of r clauses.*

Proof: Step 1 introduces to h a seed for each monotone clause in f_T. In step 2, if a clause c is the only seed in h of some c_T in f_T and is removed, then by property $(*)$, one of the new clauses created will be a seed of c_T. So, once a clause in f_T has a seed in h, it will continue to have one.

Step 3 adds to h a seed for some clause in f_T that previously had no seed in h. If we predicted "positive" on a negative example $\mathbf{x}$, then $\mathbf{x}$ must have satisfied all clauses in f_T for which some seed exists in h. Notice that this includes all monotone clauses of f_T. Since $\mathbf{x}$ is a negative example, it must *not* satisfy some other clause c_T in f_T. Because c_T is not a monotone clause, $\mathbf{x}$ must have as attributes all those negated in c_T and so in step 3, a seed for c_T will be put into h. For instance, the example might not satisfy the clause $(a_1 \vee \overline{a_3} \vee \overline{a_4})$ which would imply that the example *must* have the attributes a_3 and a_4. Thus, in step 3 we would add in to h a seed for that clause.

The algorithm makes at most r mistakes on negative examples since each one adds to h the seed to at least one new clause of f_T. In addition, each such mistake adds to h a total of at most $\binom{n}{m}$ clauses of m literals for each $m \leq K$. As in the proof for the monotone case, we put a cost of $(n+1)^{K-m}$ on each clause in h of m literals. So, each such mistake adds a total cost of at most

$$\sum_{m=1}^{K} \binom{n}{m}(n+1)^{K-m} \;\leq\; \sum_{m=1}^{K}(n+1)^K \;\leq\; K(n+1)^K.$$

Step 1 starts h with a total cost of at most $n(n+1)^{K-1}$ and step 2 decreases the total cost by at least 1 and the cost is always at least r. So, the maximum number of mistakes possible is

$$\begin{aligned} & r+1+r[K(n+1)^K]+n(n+1)^{K-1}-r \\ \leq & \ (rK+1)(n+1)^K. \end{aligned}$$

∎

4 Reducing the mistake bound

Littlestone (1987) presented remarkably efficient algorithms for learning the concept classes discussed in the previous sections in the standard model in situations where the number of relevant attributes r is much smaller than the total number of attributes in the space. These algorithms use a linear-threshold representation of the concepts and a clever method of performing weight updates, and yield mistake bounds on the order of $r \log n$ and time and space $n \log n$ when examples are chosen from $\{0,1\}^n$.

In the new model, the logarithm of the number of attributes may still be large or infinite. However, we can combine the ideas in Littlestone's algorithms with the learning algorithms described above to produce procedures that have mistake bounds on the order of $r \log n$ where n is the size of the largest example seen instead of the size of the attribute space. Thus, these new algorithms work well when we have a three-stage hierarchy: a small number of relevant attributes, a larger number of attributes that appear in each example, and an enormous number of possible attributes in the universe.

Littlestone's threshold algorithm "Winnow1" for learning monotone disjunctions works as follows.[1] Associate a weight w_i to each attribute and initialize it to 1; let $\vec{w}$ be the vector of weights. On example $\vec{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$, predict "positive" if $\vec{v} \cdot \vec{w} > n$; otherwise predict "negative". If a mistake is made on a positive example $\vec{v}$, then double weights w_i for which $v_i = 1$. If a mistake is made on a negative example $\vec{v}$, then for all $v_i = 1$, set w_i to 0. Littlestone proves that this algorithm makes at most $r \log n$ mistakes in the standard mistake-bound model (Littlestone, 1987). The algorithm can also be used for non-monotone disjunctions or K-DNF by using the standard transformations of these problems to monotone disjunctions.

Since the mistake bound of Winnow1 grows so slowly with n, we can actually use a "brute-force" conversion to apply it to the infinite attribute case. The basic idea is as follows. We begin with a small set of attributes S, say those in the first positive example, and run Winnow1 ignoring all attributes seen not in this set. If we modify Winnow1 to handle the concepts "true" and "false", then if we make more than $r \log |S|$ mistakes, we *know* that some example on which a mistake is made has some relevant attribute that we ignored. (Otherwise all such examples would be consistent with some monotone disjunction of just attributes in S or "true" or "false"). So, we can include all the at most $nr \log |S|$ attributes we ignored that appeared in those examples and restart Winnow1 on the larger attribute set and so on. We will make at most $O(r^2 \log(rn))$ mistakes total since each new iteration adds at least 1 relevant attribute and at most $nr \log(\text{poly}(r,n))$ irrelevant ones.

We can also, however, learn disjunctions and K-DNF using a more direct approach, merging the idea of Winnow1 with the algorithms described in the previous section, that yields better bounds. For this approach, we must limit the number of attributes under consideration and deal directly

[1] Littlestone actually allows for a variety of thresholds and multipliers. Only the simplest form of his algorithm is described here.

with the asymmetry between negated and non-negated attributes. To illustrate ideas for both these problems, let us consider learning general (non-monotone) disjunctions in the new model.

In the following algorithm, S will be the set of literals currently under consideration. For convenience, we make the following definition.

Definition 2 *Given an example* $\mathbf{x}$ *and a set of terms* S*, let* $\mathbf{x}(S)$ *be the set of those terms in* S *satisfied by* $\mathbf{x}$*.*

For instance, if $S = \{a_1, a_2, \overline{a_3}\}$ and $\mathbf{x} = \{a_1, a_4\}$, then $\mathbf{x}(S) = \{a_1, \overline{a_3}\}$. In addition, we will make the simplifying assumption in this section that all examples seen have the same number of attributes n. In general, this may not be the case and one can modify the algorithm to work correctly when the example sizes vary.

Theorem 3 *EFFICIENT-LEARN-DISJUNCTION below makes at most* $O(r \log n)$ *mistakes on any disjunction of* r *literals.*

EFFICIENT-LEARN-DISJUNCTION

1. Predict "positive" until we see our first negative example $\mathbf{x}$. Initialize S to the set of all $\overline{a_i}$ for which $a_i \in \mathbf{x}$, and for each term $t \in S$ initialize a weight w_t to 1. Notice that since $\mathbf{x}$ is a negative example, we must have $a_i \in \mathbf{x}$ for every $\overline{a_i}$ disjoined in f_T, so S now contains every negated attribute in f_T.

2. Given an example $\mathbf{x}$, if $\sum_{t \in \mathbf{x}(S)} w_t > n$, then predict "positive"; otherwise predict "negative".

3. If we make a mistake on a *positive* example $\mathbf{x}$, then:
   - For each $t \in \mathbf{x}(S)$, let $w_t \leftarrow 2w_t$.
   - For each $a_i \in \mathbf{x}$ such that $a_i \notin S$, put a_i into S and initialize w_{a_i} to 1.

4. If we make a mistake on a *negative* example $\mathbf{x}$, then:
   - For each $t \in \mathbf{x}(S)$, let $w_t \leftarrow 0$.

Proof of theorem: Let l be some literal disjoined in f_T. We know that w_l will never be set to zero in step 4 since no negative example can satisfy l. In addition, if w_l ever reaches a value greater than n, we will never again make a mistake on a (positive) example satisfying l since for any such example $\mathbf{x}$, we will have $\sum_{t \in \mathbf{x}(S)} w_t > n$.

If $l = \overline{a_i}$ for some a_i, then l is placed in S and w_l set to 1 in step 1. Otherwise, l will be put into S and w_l set to 1 the first time a mistake is made on a (positive) example satisfying l. Each subsequent mistake on a positive example satisfying l doubles the value of w_l. So, the maximum number of mistakes we can make on positive examples satisfying l is $1 + (\log n + 1) = 2 + \log n$; after this number of mistakes, we are guaranteed that $w_t > n$. Since there are r literals disjoined in f_T, the total number of mistakes possible on positive examples is $r(2 + \log n)$.

Let s be the sum of all weights w_t for $t \in S$. After step 1, s is at most n. Each mistake made on a positive example increases s by at most $2n$: at most n for the weights doubled in step 3 and at most n for the weights of the new terms added to S. Each mistake make on a negative example decreases s by at *least* n. Also, all weights are non-negative. Thus, the total number of mistakes possible on negative examples is

$$\frac{n + r(2 + \log n)(2n)}{n} = 1 + 2r(2 + \log n).$$

So, the maximum number of mistakes make by this algorithm is $1 + 3r(2 + \log n)$. ∎

One can extend the above technique to learn K-DNF formulas by combining the ideas of the above algorithm with those of LEARN-K-CNF. Recently, however, Blum, Hellerstein, and Littlestone (Blum et al., 1991) have found a method that uses a different approach and achieves similar bounds but with much simpler analysis. Therefore, we shall not present the more complicated method here, and instead refer the reader to that paper for details.

5 Allowing membership queries

One natural way to increase the power of a learner, studied by Angluin (1986) (1988), Valiant (1984) and others (Angluin et al., 1989), is to provide the learner with the ability to make membership queries. In a membership query, the learner selects an example and is then told its proper classification. One may incorporate membership queries into the mistake bound model as follows. We allow the learner at each stage to choose whether to receive an example from the adversary and make a prediction as before, or else to make a membership query. We now require for successful learning that both the number of mistakes and the number of membership queries be polynomial in n and the size of the target concept.

We show that any algorithm that learns a concept class in the standard mistake bound model with membership queries can be transformed into one that learns it in the infinite attribute model with membership queries, with only a small additional mistake and time penalty. To make this statement more precise, we first need a method to relate concept classes over $\{0,1\}^m$ with those in the infinite attribute model. There are several ways one might do this and we choose one here. For most "natural" classes C, this method is the same as simply restricting A to the first m attributes.

Definition 3 *Given a set $S = \{a_{i_1}, \ldots, a_{i_m}\} \subseteq A$, $(i_1 \leq i_2 \leq \ldots \leq i_m)$ and a vector $\vec{v} = (v_1, \ldots, v_m) \in \{0,1\}^m$, let $\tau_S(\vec{v}) = \{a_{i_j} \mid v_j = 1\}$.*

So, τ_S maps a bit vector into its indicated subset of S, and for $\mathbf{x} \subseteq S$, $\tau_S^{-1}(\mathbf{x})$ is the indicator vector of $\mathbf{x}$ in S. Thus, if f is a concept in the infinite attribute model, then $f \circ \tau_S$ is a function from $\{0,1\}^{|S|}$ to $\{0,1\}$. We can now define the concept class $C(m)$ over $\{0,1\}^m$ associated to the concept class C.

Definition 4 *For concept class C, let $C(m) = \{f \circ \tau_S \mid f \in C, S \subseteq A, \text{and } |S| = m\}$.*

That is, $C(m)$ contains every function from $\{0,1\}^m$ to $\{0,1\}$ that for some set S of m attributes and some concept f in C, treats its input as an indicator vector for a subset of S and applies f to that subset. So, for instance, if C is the class of monotone disjunctions, then $C(m)$ is all monotone disjunctions over $\{0,1\}^m$, including the concept "false". If C is a bizarre class like $\{f \mid \text{if } a_{37} \in R_f,$ then f is a disjunction, else f is a conjunction$\}$, then $C(m)$ is the class of all conjunctions and disjunctions over $\{0,1\}^m$. For most "natural" concept classes (sometimes called "naming invariant" classes (Kearns, 1989)) where the actual attribute names are not important to the definition of the class, the definition of $C(m)$ is the same as if the set S used were fixed to $\{a_1, \ldots, a_m\}$.

Theorem 4 *If for all m, $C(m)$ is learnable in the standard mistake-bound model with membership queries using at most M_m mistakes+ queries (and let us assume $M_l \leq M_m$ for $l \leq m$), then any $f_T \in C$ can be learned in the infinite-attribute mistake-bound model with membership queries using at most $2rM_{nr}$ mistakes+ queries, where $r = |R_{f_T}|$.*

Proof: We will keep a set S of attributes under consideration and initialize S to $\{\}$. Let m denote the size of S and label the elements of S as $a_{i_1}, \ldots, a_{i_m}$. Let P_m be an algorithm to learn $C(m)$ in the standard model that makes at most M_m mistakes + queries. We may assume that P_m is a conservative algorithm; that is, it only modifies its state after a mistake or membership query (Angluin, 1988), (Littlestone, 1987).

1. Initialize algorithm P_m.
2. Run one step of algorithm P_m.
   (a) If P_m makes a membership query on example $\vec{v}$, then make a membership query on $\tau_S(\vec{v})$ and return the result to P_m.
   (b) If P_m asks to receive an example from the adversary, then
      i. Get some example $\mathbf{x}$ from the adversary and feed to P_m the example $\tau^{-1}(\mathbf{x} \cap S)$; that is, $(v_1, \ldots, v_m)$ where $v_j = 1$ iff $a_{i_j} \in \mathbf{x}$.
      ii. Return the prediction of P_m as our prediction.
      iii. If our prediction was correct, then give the response $f_T(\mathbf{x})$ to P_m and go back to 2. Note that $f_T(\mathbf{x})$ may not equal $f_T(\mathbf{x} \cap S)$, but we do no harm since P_m is conservative.
      iv. If our prediction was incorrect, then make a membership query on example $\mathbf{x} \cap S$. If $f_T(\mathbf{x} \cap S) = f_T(\mathbf{x})$, then give $f_T(\mathbf{x})$ to P_m and go back to 2. Otherwise, we know there exists some relevant attribute in $\mathbf{x} - S$, so let $S \leftarrow S \cup \mathbf{x}$, let $m = |S|$, and go back to 1.

We know any given P_m will cause us to make at most M_m mistakes and M_m queries before S gets updated, since by definition of $C(m)$ there exists some $f_m \in C(m)$ that is consistent with the target function f_T over attribute space S. Each time we update S, we add at least 1 relevant attribute and at most n total attributes to S, so the largest number of attributes ever considered is nr, where $r = |R_{f_T}|$. Since there are at most r different P_m used, this procedure makes at most $2rM_{nr}$ mistakes + queries total. ∎

6 The Halving Algorithm

We now consider learning when computational constraints are ignored. Littlestone (1987) defines $opt(C)$ to be the best possible worst-case mistake bound achievable by any (not necessarily polynomial-time) algorithm for learning class C. If C_m is a concept class over $\{0,1\}^m$, then we have $opt(C_m) \leq \log_2 |C_m|$. This can be seen by using the standard "Halving Algorithm" which works as follows. Let $H = C_m$. On input $\vec{v}$, take a majority vote of all $f \in H$ and predict accordingly. If a mistake is made throw out all $f \in H$ that predicted incorrectly. Thus, each mistake reduces H by at least a factor of 2, so at most $\log_2 |C_m|$ mistakes total are made.

In the infinite-attribute model, the size of a concept class C may be infinite and the question arises: might it be that, $opt(C(m))$ is polynomial in m, but that even ignoring computational constraints, the class C cannot be learned with a number of mistakes polynomial in n and the size of the target concept? The answer to this question is "no" in the following sense.

Theorem 5 *If C is a concept class in the infinite-attribute model, then there exists a (non polynomial-time) algorithm for C that makes at most $O\left(r^2 \log(rn|C(r)|)\right)$ mistakes where r is the number of relevant attributes of the target concept and n is the size of the largest example seen.*

We will prove this by using a modified version of the Halving Algorithm called "Sequential Halving" or "SH" below. Let us first make the following definitions.

Definition 5 *For $f \in C$, let $f|_S(\mathbf{x}) = f(\mathbf{x} \cap S)$. We will say that $f|_S$ is f* restricted *to S.*

Definition 6 *Given a set $S \subseteq A$, let $C(r, S) = \{f|_S : f \in C, |R_f| \leq r\}$.*

So, the size of $C(r, S)$ is the number of functions in C on at most r relevant attributes that differ over examples whose attributes come from the set S.

We now describe the algorithm SH. For simplicity, we will assume that in addition to the concept class C, the number of relevant attributes r is given to the algorithm at the beginning.

SH(C, r)

1. To start, let $S \leftarrow \{\}$.
2. Let $H = C(r, S)$ and $T = \{\}$. Note that even if S is empty, H will have at least one concept in it (assuming C is non-empty) at this stage.
3. On example $\mathbf{x}$, predict according to a majority vote of the functions in H. If the prediction was wrong remove from H all functions that predicted incorrectly and let $T \leftarrow T \cup \mathbf{x}$. Continue with this step until H is empty.
4. If H is empty, let $S \leftarrow S \cup T$ and go back to step 2.

Proof of Theorem 5: Let f be the target function. In algorithm SH, each time step 4 is reached, a new attribute of R_f is added to S. The reason is that for all examples $\mathbf{x}$ with no relevant attributes outside of S, we have $f|_S(\mathbf{x}) = f(\mathbf{x})$ by definition of R_f. So, if all examples on which a mistake is made have no attributes in $R_f - S$, then $f|_S$ will never be removed from H in step 3. So, each time step 4 is reached, we add at least one relevant attribute and at most $n \log_2 |C(r, S)|$ attributes total.

If $|S| = r$, then by definitions 4 and 6 we have $|C(r, S)| \leq |C(r)|$. So, for $|S| = m > r$, we have $|C(r, S)| \leq \binom{m}{r}|C(r)|$ since every function in $C(r, S)$ is also an element of $C(r, S')$ for some $S' \subseteq S$ of size r. Thus, each time step 4 is reached, the number of attributes we add to S is at most

$$\begin{aligned} n \log |C(r, S)| &\leq n \log \left[\binom{|S|}{r} |C(r)| \right] \\ &\leq nr \log \left(|S| \, |C(r)| \right). \end{aligned}$$

Since we reach step 4 at most r times, if m_{max} is the size of the largest set S used, then m_{max} is at most $nr^2 \log(m_{max} |C(r)|)$. Solving for m_{max} gives

$$m_{max} = O\left(nr^2 \log(nr|C(r)|)\right).$$

Thus, if S_{max} is the largest set S used, the number of mistakes made is at most

$$\begin{aligned} & r \log |C(r, S_{max})| \\ &\leq r \log \left[\binom{m_{max}}{r} |C(r)| \right] \\ &= O\left(r^2 \log \left[m_{max} |C(r)| \right]\right) \\ &= O\left(r^2 \log \left[nr^2 |C(r)| \log(nr|C(r)|) \right]\right) \\ &= O\left(r^2 \log(nr|C(r)|)\right). \end{aligned}$$

∎

Note that we can get rid of the assumption that r is known beforehand by using a standard "doubling trick."

7 Conclusions

This paper presents a model for learning Boolean functions in a large or infinite attribute space when the size of each individual example is small. The model attempts to capture the notion that often the description of any individual object is much smaller than the size of one's total vocabulary. By allowing the attribute space to be infinite, the model no longer allows the standard methods for learning K-CNF and K-DNF which essentially list all the attributes in the space and cross them off as they are seen. We show that these concept classes remain learnable however, by new, and in some sense more "realistic" algorithms.

Some concept classes that are easy to learn in the standard model seem hard to learn in the proposed one. In particular, the class of decision lists (Rivest, 1987) is learnable in the standard model, but it is an open problem whether it can be learned in the infinite-attribute model. Note that by the results of section 6, decision lists *can* be learned in the infinite-attribute model if computational issues are ignored, so any attempt to show decision lists are *not* learnable would likely require some sort of complexity assumption.

Acknowledgements

I would like to thank Lisa Hellerstein, Michael Kearns, Nick Littlestone, Ron Rivest, and Rob Schapire for many helpful discussions and suggestions. The research presented here was conducted primarily while the author was at MIT and supported by an NSF Graduate Fellowship, NSF grant CCR-8914428 and the Siemens Corporation. Currently supported by an NSF Postdoctoral Fellowship. A preliminary version of this paper appears in (Blum, 1990).

References

Angluin, D. (1986). Learning regular sets from queries and counter-examples. Technical Report YALEU/DCS/TR-464, Yale University Department of Computer Science.

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

Angluin, D., Hellerstein, L., and Karpinski, M. (1989). Learning read-once formulas with queries. Technical Report UCB/CSD 89/528, University of California Berkeley Computer Science Division.

Blum, A. (1990). Learning boolean functions in an infinite attribute space. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 64–72, Baltimore, Maryland.

Blum, A., Hellerstein, L., and Littlestone, N. (1991). Learning in the presence of finitely or infinitely many irrelevant attributes. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, Santa Cruz, California. Morgan Kaufmann.

Kearns, M. (1989). *The Computational Complexity of Machine Learning.* PhD thesis, Harvard University Center for Research in Computing Technology. Technical Report TR-13-89. Also published by MIT Press as an ACM Distinguished Dissertation.

Kearns, M., Li, M., Pitt, L., and Valiant, L. (1987). Recent results on boolean concept learning. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 337–352, University of California, Irvine.

Littlestone, N. (1987). Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318.

Littlestone, N. (1989). *Mistake bounds and logarithmic linear-threshold learning algorithms*. PhD thesis, U. C. Santa Cruz.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, 2(3):229–246.

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