Toward Attribute Efficient Learning of Decision Lists and Parities

Adam R. Klivans*[1] and Rocco A. Servedio[2]

[1] Divsion of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
klivans@eecs.harvard.edu
[2] Department of Computer Science
Columbia University
New York, NY 10027, USA
rocco@cs.columbia.edu

Abstract. We consider two well-studied problems regarding attribute efficient learning: learning decision lists and learning parity functions.
First, we give an algorithm for learning decision lists of length k over n variables using $2^{\tilde{O}(k^{1/3})} \log n$ examples and time $n^{\tilde{O}(k^{1/3})}$. This is the first algorithm for learning decision lists that has both subexponential sample complexity and subexponential running time in the relevant parameters. Our approach is based on a new construction of low degree, low weight polynomial threshold functions for decision lists. For a wide range of parameters our construction matches a lower bound due to Beigel for decision lists and gives an essentially optimal tradeoff between polynomial threshold function degree and weight.
Second, we give an algorithm for learning an unknown parity function on k out of n variables using $O(n^{1-1/k})$ examples in poly(n) time. For $k = o(\log n)$ this yields the first polynomial time algorithm for learning parity on a superconstant number of variables with sublinear sample complexity. We also give a simple algorithm for learning an unknown size-k parity using $O(k \log n)$ examples in $n^{k/2}$ time, which improves on the naive n^k time bound of exhaustive search.

1 Introduction

An important goal in machine learning theory is to design *attribute efficient* algorithms for learning various classes of Boolean functions. A class $\mathcal{C}$ of Boolean functions over n variables $x_1, \ldots, x_n$ is said to be *attribute-efficiently learnable* if there is a poly(n) time algorithm which can learn any function $f \in C$ using a number of examples which is polynomial in the "size" (description length) of the function f to be learned, rather than in n, the number of features in the domain over which learning takes place. (Note that the running time of

* Supported by a National Science Foundation Mathematical Sciences Postdoctoral Research Fellowship.

the learning algorithm must in general be at least n since each example is an n-bit vector.) Thus an attribute efficient learning algorithm for e.g. the class of Boolean conjunctions must be able to learn any Boolean conjunction of k literals over $x_1, \ldots, x_n$ using $\text{poly}(k, \log n)$ examples, since $k \log n$ bits are required to specify such a conjunction.

A longstanding open problem in machine learning, posed first by Blum in 1990 [4, 5, 7, 8] and again by Valiant in 1998 [32], is whether or not there exist attribute efficient algorithms for learning *decision lists*, which are essentially nested "if-then-else" statements (we give a precise definition in Section 2). One motivation for considering the problem comes from the *infinite attribute model* introduced in [4]. Blum *et al.* [7] showed that for many concept classes (including decision lists) attribute efficient learnability in the standard n-attribute model is equivalent to learnability in the infinite attribute model. Since simple classes such as disjunctions and conjunctions are attribute efficiently learnable (and hence learnable in the infinite attribute model), this motivated Blum [4] to ask whether the richer class of decision lists is thus learnable as well. Several researchers [5, 8, 10, 25, 28] have since considered this problem; we summarize this previous work in Section 1.2. More recently, Valiant [32] relates the problem of learning decision lists attribute efficiently to questions about human learning abilities.

Another outstanding challenge in machine learning is to determine whether there exist attribute efficient algorithms for learning *parity functions*. The parity function on a set of 0/1-valued variables $x_{i_1}, \ldots, x_{i_k}$ takes value $+1$ or -1 depending on whether $x_{i_1} + \cdots + x_{i_k}$ is even or odd. As with decision lists, a simple PAC learning algorithm is known for the class of parity functions but no attribute efficient algorithm is known.

1.1 Our Results

We give the first learning algorithm for decision lists that is subexponential in both sample complexity (in the relevant parameters k and $\log n$) and running time (in the relevant parameter k). Our results demonstrate for the first time that it is possible to simultaneously avoid the "worst case" in both sample complexity and running time, and thus suggest that it may perhaps be possible to learn decision lists attribute efficiently. Our main learning result for decision lists is:

Theorem 1. *There is an algorithm which learns length-k decision lists over $\{0,1\}^n$ with mistake bound $2^{\tilde{O}(k^{1/3})} \log n$ and time $n^{\tilde{O}(k^{1/3})}$.*

This bound improves on the sample complexity of Littlestone's well-known Winnow algorithm [20] for all k and improves on its runtime as well for $k = \Omega(\log^{3/2} n)$; see Section 1.2.

We prove Theorem 1 in two parts; first we generalize the Winnow algorithm for learning linear threshold functions to learn *polynomial threshold functions* (PTFs). In recent work on learning DNF formulas [18], intersections of halfspaces [17], and Boolean formulas of superconstant depth [26], PTFs of degree d have been learned in time $n^{O(d)}$ by using polynomial time linear programming

algorithms such as the Ellipsoid algorithm (see e.g. [18]). In contrast, since we want to achieve low sample complexity as well as an $n^{O(d)}$ runtime, we use a generalization of the Winnow algorithm to learn PTFs. This generalization has sample complexity and running time bounds which depend on the degree and the total magnitude of the integer coefficients (i.e. the weight) of the PTF:

Theorem 2. *Let $\mathcal{C}$ be a class of Boolean functions over $\{0,1\}^n$ with the property that each $f \in \mathcal{C}$ has a PTF of degree at most d and weight at most W. Then there is an online learning algorithm for $\mathcal{C}$ which runs in n^d time per example and has mistake bound $O(W^2 \cdot d \cdot \log n)$.*

This reduces the decision list learning problem to a problem of representing decision lists with PTFs of low weight and low degree. To this end we prove:

Theorem 3. *Let L be a decision list of length k. Then L is computed by a polynomial threshold function of degree $\tilde{O}(k^{1/3})$ and weight $2^{\tilde{O}(k^{1/3})}$.*

Theorem 1 follows directly from Theorems 2 and 3. We emphasize that Theorem 3 does *not* follow from previous results [18] on representing DNF formulas as PTFs; the PTF construction from [18] in fact has exponentially larger weight ($2^{2^{\tilde{O}(k^{1/3})}}$ rather than $2^{\tilde{O}(k^{1/3})}$) than the construction in this paper.

Our PTF construction is essentially optimal in the tradeoff between degree and weight which it achieves. In 1994 Beigel [3] gave a lower bound showing that any degree d PTF for a certain decision list must have weight $2^{\Omega(n/d^2)}$. For $d = n^{1/3}$, Beigel's lower bound implies that our construction in Theorem 3 is essentially best possible.

For parity functions, we give an $O(n^4)$ time algorithm which can PAC learn an unknown parity on k variables out of n using $\tilde{O}(n^{1-1/k})$ examples. To our knowledge this is the first algorithm for learning parity on a superconstant number of variables with sublinear sample complexity. Our algorithm works by finding a "low weight" solution to a system of m linear equations (corresponding to a set of m examples). We prove that with high probability we can find a solution of weight $O(n^{1-1/k})$ irrespective of m. Thus by taking m to be only slightly larger than $n^{1-1/k}$, standard arguments show that our solution is a good hypothesis.

We also describe a simple algorithm, due to Dan Spielman, for learning an unknown parity on k variables using $O(k \log n)$ examples and $\tilde{O}(n^{k/2})$ time. This gives a square root runtime improvement over a naive $O(n^k)$ exhaustive search.

1.2 Previous Results

In previous work several algorithms with different performance bounds (runtime and sample complexity) have been given for learning length-k decision lists.

- Rivest [27] gave the first algorithm for learning decision lists in Valiant's PAC model of learning from random examples. Littlestone [5] later gave an analogue of Rivest's algorithm in the online learning model. The algorithm can learn any decision list of length k in $O(kn^2)$ time using $O(kn)$ examples.

– A brute-force approach is to maintain the set of all length-k decision lists which are consistent with the examples seen so far, and to predict at each stage using majority vote over the surviving hypotheses. This "halving algorithm" (proposed in various forms in [1, 2, 23]) can learn decision lists of length k using only $O(k \log n)$ examples, but the running time is $n^{O(k)}$.
– Several researchers [5, 32] have observed that Winnow can learn length-k decision lists from $2^{O(k)} \log n$ examples in time $2^{O(k)} n \log n$. This follows from the fact that any decision list of length k can be expressed as a linear threshold function with integer coefficients of magnitude $2^{\Theta(k)}$.
– Finally, several researchers have considered the special case of learning a length-k decision list in which the output bits of the list have at most D alternations. Valiant [32] and Nevo and El-Yaniv [25] have given refined analyses of Winnow's performance for this case (see also Dhagat and Hellerstein [10]). However, for the general case where D can be as large as k, these results do not improve on the standard Winnow analysis described above.

Note that all of these earlier algorithms have an exponential dependence on the relevant parameter(s) (k and $\log n$ for sample complexity, k for running time) for either the running time or the sample complexity.

Little previous work has been published on learning parity functions attribute efficiently in the PAC model. The standard PAC learning algorithm for parity (based on solving a system of linear equations) is due to Helmbold *et al.* [15]; however this algorithm is not attribute efficient since it uses $\Omega(n)$ examples regardless of k. Several authors have considered learning parity attribute efficiently in a model where the learner is allowed to make membership queries. Attribute efficient learning is easier in this framework since membership queries can help identify relevant variables. Blum et al. [7] give a randomized polynomial time membership-query algorithm for learning parity on k variables using only $O(k \log n)$ examples, and these results were later refined by Uehara *et al.* [31].

In Section 2 we give necessary background. In Section 3 we show how to reduce the decision list learning problem to a problem of finding suitable PTF representations of decision lists (Theorem 2). In Section 4 we give our PTF construction for decision lists (Theorem 3). In Section 5 we discuss the connection between Theorem 3 and Beigel's ODDMAXBIT lower bound. In Section 6 we give our results on learning parity functions, and we conclude in Section 7.

2 Preliminaries

Attribute efficient learning has been chiefly studied in the *on-line mistake-bound* model of concept learning which was introduced in [20, 22]. In this model learning proceeds in a series of trials, where in each trial the learner is given an unlabelled boolean example $x \in \{0,1\}^n$ and must predict the value $f(x)$ of the unknown target function f. After each prediction the learner is given the true value of $f(x)$ and can update its hypothesis before the next trial begins. The *mistake bound* of a learning algorithm on a target concept c is the worst-case number of mistakes

that the algorithm makes over all (possibly infinite) sequences of examples, and the mistake bound of a learning algorithm on a concept class (class of Boolean functions) C is the worst-case mistake bound across all functions $f \in C$. The running time of a learning algorithm A for a concept class C is defined as the product of the mistake bound of A on C times the maximum running time required by A to evaluate its hypothesis and update its hypothesis in any trial.

Our main interests are the classes of *decision lists* and *parity functions*. A decision list L of length k over the Boolean variables $x_1, \ldots, x_n$ is represented by a list of k pairs and a bit $(\ell_1, b_1), (\ell_2, b_2), \ldots, (\ell_k, b_k), b_{k+1}$ where each ℓ_i is a literal and each b_i is either -1 or 1. Given any $x \in \{0,1\}^n$, the value of $L(x)$ is b_i if i is the smallest index such that ℓ_i is made true by x; if no ℓ_i is true then $L(x) = b_{k+1}$. A parity function of length k is defined by a set of variables $S \subset \{x_1, \ldots, x_n\}$ such that $|S| = k$. The parity function $\chi_S(x)$ takes value 1 (-1) on inputs which set an even (odd) number of variables in S to 1.

Given a concept class C over $\{0,1\}^n$ and a Boolean function $f \in C$, let size(f) denote the description length of f under some reasonable encoding scheme. We say that a learning algorithm A for C in the mistake-bound model is *attribute-efficient* if the mistake bound of A on any concept $f \in C$ is polynomial in size(f). In particular, the description length of a length k decision list (parity) is $O(k \log n)$, and thus we would ideally like to have poly(n)-time algorithms which learn decision lists (parities) of length k with a mistake bound of poly$(k, \log n)$.

(We note here that attribute efficiency has also been studied in other learning models, namely Valiant's Probably Approximately Correct (PAC) model of learning from random examples. Standard conversion techniques are known [1, 14, 21] which can be used to transform any mistake bound algorithm into a PAC learning algorithm. These transformations essentially preserve the running time of the mistake bound algorithm, and the sample size required by the PAC algorithm is essentially the mistake bound. Thus, positive results for mistake bound learning, such as those we give for decision lists in this paper, directly yield corresponding positive results for the PAC model.)

Finally, our results for decision lists are achieved by a careful analysis of *polynomial threshold functions*. Let f be a Boolean function $f : \{0,1\}^n \to \{-1,1\}$ and let p be a polynomial in n variables with integer coefficients. Let d denote the degree of p and let W denote the sum of the absolute values of p's integer coefficients. If the sign of $p(x)$ equals $f(x)$ for every $x \in \{0,1\}^n$, then we say that p is a *polynomial threshold function (PTF) of degree d and weight W* for f.

3 Expanded-Winnow: Learning Polynomial Threshold Functions

Littlestone [20] introduced the online Winnow algorithm and showed that it can attribute efficiently learn Boolean conjunctions, disjunctions, and low weight linear threshold functions. Throughout its execution Winnow maintains a linear threshold function as its hypothesis; at the heart of the algorithm is an update rule which makes a multiplicative update to each coefficient of the hypothesis

each time a mistake is made. Since its introduction Winnow has been intensively studied from both applied and theoretical standpoints (see e.g. [6, 12, 16, 29]).

The following theorem (which, as noted in [32], is implicit in Littlestone's analysis in [20]) gives a mistake bound for Winnow for linear threshold functions:

Theorem 4. *Let $f(x)$ be the linear threshold function sign$(\sum_{i=1}^{n} w_i x_i - \theta)$ over inputs $x \in \{0,1\}^n$ where θ and $w_1, \ldots, w_n$ are integers. Let $W = \sum_{i=1}^{n} |w_i|$. Then Winnow learns $f(x)$ with mistake bound $O(W^2 \log n)$, and uses n time steps per example.*

We will use a generalization of the Winnow algorithm, which we call Expanded-Winnow, to learn *polynomial* threshold functions of degree at most d. Our generalization introduces $\sum_{i=1}^{d} \binom{n}{d}$ new variables (one for each monomial of degree up to d) and runs Winnow to learn a linear threshold function over these new variables. More precisely, in each trial we convert the n-bit received example $x = (x_1, \ldots, x_n)$ into a $\sum_{i=1}^{d} \binom{n}{d}$ bit expanded example (where the bits in the expanded example correspond to monomials over $x_1, \ldots, x_n$), and we give the expanded example to Winnow. Thus the hypothesis which Winnow maintains – a linear threshold function over the space of expanded features – is a polynomial threshold function of degree d over the original n variables $x_1, \ldots, x_n$. Theorem 2, which follows directly from Theorem 4, summarizes the performance of Expanded-Winnow:

Theorem 2 *Let $\mathcal{C}$ be a class of Boolean functions over $\{0,1\}^n$ with the property that each $f \in \mathcal{C}$ has a polynomial threshold function of degree at most d and weight at most W. Then Expanded-Winnow algorithm runs in n^d time per example and has mistake bound $O(W^2 \cdot d \cdot \log n)$ for $\mathcal{C}$.*

Theorem 2 shows that the degree of a polynomial threshold function strongly affects Expanded-Winnow's running time, and the weight of a polynomial threshold function strongly affects its sample complexity.

4 Constructing PTFs for Decision Lists

In previous constructions of polynomial threshold functions for computational learning theory applications [18, 17, 26] the sole goal has been to minimize the degree of the polynomials regardless of the size of the coefficients. As one example, the construction of [18] of $\tilde{O}(n^{1/3})$ degree PTFs for DNF formulae yields polynomials whose coefficients can be *doubly exponential* in the degree. In contrast, we must now construct PTFs that have low degree and low weight.

We give two constructions of PTFs for decision lists, each of which has relatively low degree and relatively low weight. We then combine these to achieve an optimal construction with improved bounds on both degree and weight.

4.1 Outer Construction

Let L be a decision list of length k over variables $x_1, \ldots, x_k$. We first give a simple construction of a degree h, weight $\frac{2k}{h} 2^{(k/h+h)}$ PTF for L which is based

on breaking the list L into sublists. We call this construction the "outer construction" since we will ultimately combine this construction with a different construction for the "inner" sublists.

We begin by showing that L can be expressed as a threshold of *modified decision lists* which we now define. The set $\mathcal{B}_h$ of modified decision lists is defined as follows: each function in $\mathcal{B}_h$ is a decision list $(\ell_1, b_1), (\ell_2, b_2), \ldots, (\ell_h, b_h), 0$ where each ℓ_i is some literal over $x_1, \ldots, x_n$ and each $b_i \in \{-1, 1\}$. Thus the only difference between a modified decision list $f \in \mathcal{B}_h$ and a normal decision list of length h is that the final output value is 0 rather than $b_{h+1} \in \{-1, +1\}$.

Without loss of generality we may suppose that the list L is $(x_1, b_1), \ldots, (x_k, b_k), b_{k+1}$. We break L sequentially into k/h blocks each of length h. Let $f_i \in \mathcal{B}_h$ be the modified decision list which corresponds to the i-th block of L, i.e. f_i is the list $(x_{(i-1)h+1}, b_{(i-1)h+1}), \ldots, (x_{(i+1)h}, b_{(i+1)h}), 0$. Intuitively f_i computes the ith block of L and equals 0 only if we "fall of the edge" of the ith block. We then have the following straightforward claim:

Claim. The decision list L is eqivalent to

$$\text{sign}\left(\sum_{i=1}^{k/h} 2^{k/h-i+1} f_i(x) \ + \ b_{k+1}\right). \tag{1}$$

Proof. Given an input $x \neq 0^k$ let $r = (i-1)h + c$ be the first index such that x_r is satisfied. It is easy to see that $f_j(x) = 0$ for $j < i$ and hence the value in (1) is $2^{k/h-i+1} b_r + \sum_{j=i+1}^{k/h} 2^{k/h-j+1} f_j(x) \ + \ b_{k+1}$, the sign of which is easily seen to be b_r. Finally if $x = 0^k$ then the argument to (1) is b_{k+1}. □

Note: It is easily seen that we can replace the 2 in formula (1) by a 3; this will prove useful later.

As an aside, note that Claim 4.1 can already be used to obtain a tradeoff between running time and sample complexity for learning decision lists. The class $\mathcal{B}_h$ contains at most $(4n)^h$ functions. Thus as in Section 3 it is possible to run the Winnow algorithm using the functions in $\mathcal{B}_h$ as the base features for Winnow. (So for each example x which it receives, the algorithm would first compute the value of $f(x)$ for each $f \in \mathcal{B}_h$, and would then use this vector of $(f(x))_{f \in \mathcal{B}_h}$ values as the example point for Winnow.) A direct analogue of Theorem 2 now implies that Expanded-Winnow (run over this expanded feature space of functions from $\mathcal{B}_h$) can be used to learn L_k in time $n^{O(h)} 2^{O(k/h)}$ with mistake bound $2^{O(k/h)} h \log n$.

However, it will be more useful for us to obtain a PTF for L. We can do this from Claim 4.1 as follows:

Theorem 5. *Let L be a decision list of length k. For any $h < k$ we have that L is computed by a polynomial threshold function of degree h and weight $4 \cdot 2^{k/h+h}$.*

Proof. Consider the first modified decision list $f_1 = (\ell_1, b_1), (\ell_2, b_2), \ldots, (\ell_h, b_h), 0$ in the expression (1). For ℓ a literal let $\tilde{\ell}$ denote x if ℓ is an unnegated variable

x and let $\tilde{\ell}$ denote $1-x$ if if ℓ is a negated variable $\overline{x}$. We have that for all $x \in \{0,1\}^h$, $f_1(x)$ is computed exactly by the polynomial

$$f_1(x) = \tilde{\ell}_1 b_1 + (1-\tilde{\ell}_1)\tilde{\ell}_2 b_2 + (1-\tilde{\ell}_1)(1-\tilde{\ell}_2)\tilde{\ell}_3 b_3 + \cdots + (1-\tilde{\ell}_1)\cdots(1-\tilde{\ell}_{h-1})\tilde{\ell}_h b_h.$$

This polynomial has degree h and has weight at most 2^{h+1}. Summing these polynomial representations for $f_1, \ldots, f_{k/h}$ as in (1) we see that the resulting PTF given by (1) has degree h and weight at most $2^{k/h+1} \cdot 2^{h+1} = 4 \cdot 2^{k/h+h}$. □

Specializing to the case $h = \sqrt{k}$ we obtain:

Corollary 1. *Let L be a decision list of length k. Then L is computed by a polynomial threshold function of degree $k^{1/2}$ and weight $4 \cdot 2^{2k^{1/2}}$.*

We close this section by observing that an intermediate result of [18] can be used to give an alternate proof of Corollary 1 with slightly weaker parameters; however our later proofs require the construction given in this section.

4.2 Inner Approximator

In this section we construct low degree, low weight polynomials which approximate (in the L_∞ norm) the modified decision lists from the previous subsection. Moreover, the polynomials we construct are exactly correct on inputs which "fall off the end":

Theorem 6. *Let $f \in \mathcal{B}_h$ be a modified decision list of length h (without loss of generality we may assume that f is $(x_1, b_1), \ldots, (x_h, b_h), 0$). Then there is a degree $2\sqrt{h}\log h$ polynomial p such that*

- *for every input $x \in \{0,1\}^h$ we have $|p(x) - f(x)| \leq 1/h$.*
- *$p(0^h) = f(0^h) = 0$.*

Proof. As in the proof of Theorem 5 we have that

$$f(x) = b_1 x_1 + b_2(1-x_1)x_2 + \cdots + b_h(1-x_1)\cdots(1-x_{h-1})x_h.$$

We will construct a lower (roughly $\sqrt{h}$) degree polynomial which closely approximates f. Let T_i denote $(1-x_1)\ldots(1-x_{i-1})x_i$, so we can rewrite f as

$$f(x) = b_1 T_1 + b_2 T_2 + \cdots + b_h T_h.$$

We approximate each T_i separately as follows: set $A_i(x) = h - i + x_i + \sum_{j=1}^{i-1}(1-x_j)$. Note that for $x \in \{0,1\}^h$, we have $T_i(x) = 1$ iff $A_i(x) = h$ and $T_i(x) = 0$ iff $0 \leq A_i(x) \leq h-1$. Now define the polynomial

$$Q_i(x) = q\left(A_i(x)/h\right) \qquad \text{where} \qquad q(y) = C_d\left(y\left(1+1/h\right)\right).$$

As in [18], here $C_d(x)$ is the dth Chebyshev polynomial of the first kind (a univariate polynomial of degree d) with d set to $\lceil\sqrt{h}\rceil$. We will need the following facts about Chebyshev polynomials [9]:

– $|C_d(x)| \leq 1$ for $|x| \leq 1$ with $C_d(1) = 1$;
– $C'_d(x) \geq d^2$ for $x > 1$ with $C'_d(1) = d^2$.
– The coefficients of C_d are integers each of whose magnitude is at most 2^d.

These first two facts imply that $q(1) \geq 2$ but $|q(y)| \leq 1$ for $y \in [0, 1 - \frac{1}{h}]$. We thus have that $Q_i(x) = q(1) \geq 2$ if $T_i(x) = 1$ and $|Q_i(x)| \leq 1$ if $T_i(x) = 0$. Now define $P_i(x) = \left(\frac{Q_i(x)}{q(1)}\right)^{2 \log h}$. This polynomial is easily seen to be a good approximator for T_i: if $x \in \{0,1\}^h$ is such that $T_i(x) = 1$ then $P_i(x) = 1$, and if $x \in \{0,1\}^h$ is such that $T_i(x) = 0$ then $|P_i(x)| < \left(\frac{1}{2}\right)^{2 \log h} < \frac{1}{h^2}$.

Now define $R(x) = \sum_{i=1}^{\ell} b_i P_i(x)$ and $p(x) = R(x) - R(0^h)$. It is clear that $p(0^h) = 0$. We will show that for every input $0^h \neq x \in \{0,1\}^h$ we have $|p(x) - f(x)| \leq 1/h$. Fix some such x; let i be the first index such that $x_i = 1$. As shown above we have $P_i(x) = 1$. Moreover, by inspection of $T_j(x)$ we have that $T_j(x) = 0$ for all $j \neq i$, and hence $|P_j(x)| < \frac{1}{h^2}$. Consequently the value of $R(x)$ must lie in $[b_i - \frac{h-1}{h^2}, b_i + \frac{h-1}{h^2}]$. Since $f(x) = b_i$ we have that $p(x)$ is an L_∞ approximator for $f(x)$ as desired.

Finally, it is straightforward to verify that $p(x)$ has the claimed degree. □

Strictly speaking we cannot discuss the weight of the polynomial p since its coefficients are rational numbers but not integers. However, by multiplying p by a suitable integer (clearing denominators) we obtain an integer polynomial with essentially the same properties. Using the third fact about Chebyshev polynomials from our proof above, we have that $q(1)$ is a rational number N_1/N_2 where N_1, N_2 are each integers of magnitude $h^{O(\sqrt{h})}$. Each $Q_i(x)$ for $i = 1, \ldots, h$ can be written as an integer polynomial (of weight $h^{O(\sqrt{h})}$) divided by $h^{\sqrt{h}}$. Thus each $P_i(x)$ can be written as $\tilde{P}_i(x)/(h^{\sqrt{h}} N_1)^{2 \log h}$ where $\tilde{P}_i(x)$ is an integer polynomial of weight $h^{O(\sqrt{h} \log h)}$. It follows that $p(x)$ equals $\tilde{p}(x)/C$, where C is an integer which is at most $2^{O(h^{1/2} \log^2 h)}$ and $\tilde{p}$ is a polynomial with integer coefficients and weight $2^{O(h^{1/2} \log^2 h)}$. We thus have

Corollary 2. *Let $f \in \mathcal{B}_h$ be a modified decision list of length h. Then there is an integer polynomial $p(x)$ of degree $2\sqrt{h} \log h$ and weight $2^{O(h^{1/2} \log^2 h)}$ and an integer $C = 2^{O(h^{1/2} \log^2 h)}$ such that*

– *for every input $x \in \{0,1\}^h$ we have $|p(x) - Cf(x)| \leq C/h$.*
– *$p(0^h) = f(0^h) = 0$.*

The fact that $p(0^h)$ is exactly 0 will be important in the next subsection when we combine the inner approximator with the outer construction.

4.3 Composing the Constructions

In this section we combine the two constructions from the previous subsections to obtain our main polynomial threshold construction:

Theorem 7. *Let L be a decision list of length k. Then for any $h < k$, L is computed by a polynomial threshold function of degree $O(h^{1/2} \log h)$ and weight $2^{O(k/h + h^{1/2} \log^2 h)}$.*

Proof. We suppose without loss of generality that L is the decision list $(x_1, b_1), \ldots, (x_k, b_k), b_{k+1}$. We begin with the outer construction: from the note following Claim 4.1 we have that

$$L(x) = \text{sign}\left(C \left[\sum_{i=1}^{k/h} 3^{k/h-i+1} f_i(x) \ + \ b_{k+1} \right] \right)$$

where C is the value from Corollary 2 and each f_i is a modified decision list of length h computing the restriction of L to its ith block as defined in Subsection 4.1. Now we use the inner approximator to replace each Cf_i above by p_i, the approximating polynomial from Corollary 2, i.e. consider $\text{sign}(H(x))$ where

$$H(x) = \sum_{i=1}^{k/h} (3^{k/h-i+1} p_i(x)) \ + \ Cb_{k+1}.$$

We will show that $\text{sign}(H(x))$ is a PTF which computes L correctly and has the desired degree and weight.

Fix any $x \in \{0,1\}^k$. If $x = 0^k$ then by Corollary 2 each $p_i(x)$ is 0 so $H(x) = Cb_{k+1}$ has the right sign. Now suppose that $r = (i-1)h + c$ is the first index such that $x_r = 1$. By Corollary 2, we have that

- $3^{k/h-j+1} p_j(x) = 0$ for $j < i$;
- $3^{k/h-i+1} p_i(x)$ differs from $3^{k/h-i+1} Cb_r$ by at most $C3^{k/h-i+1} \cdot \frac{1}{h}$;
- The magnitude of each value $3^{k/h-j+1} p_j(x)$ is at most $C3^{k/h-j+1}(1 + \frac{1}{h})$ for $j > i$.

Combining these bounds, the value of $H(x)$ differs from $3^{k/h-i+1} Cb_r$ by at most

$$C\left(\frac{3^{k/h-i+1}}{h} + \left(1 + \frac{1}{h}\right) \left[3^{k/h-i} + 3^{k/h-i-1} + \cdots + 3 \right] + 1 \right)$$

which is easily seen to be less than $C3^{k/h-i+1}$ in magnitude. Thus the sign of $H(x)$ equals b_r, and consequently $\text{sign}(H(x))$ is a valid polynomial threshold representation for $L(x)$. Finally, our degree and weight bounds from Corollary 2 imply that the degree of $H(x)$ is $O(h^{1/2} \log h)$ and the weight of $H(x)$ is $2^{O(k/h) + O(h^{1/2} \log^2 h)}$, and the theorem is proved. □

Taking $h = k^{2/3} / \log^{4/3} k$ in the above theorem we obtain our main result on representing decision lists as polynomial threshold functions:

Theorem 3 *Let L be a decision list of length k. Then L is computed by a polynomial threshold function of degree $k^{1/3} \log^{1/3} k$ and weight $2^{O(k^{1/3} \log^{4/3} k)}$.*

Theorem 3 immediately implies that Expanded-Winnow can learn decision lists of length k using $2^{\tilde{O}(k^{1/3})} \log n$ examples and time $n^{\tilde{O}(k^{1/3})}$.

4.4 Application to Learning Decision Trees

In 1989 Ehrenfeucht and Haussler [11] gave an a time $n^{O(\log s)}$ algorithm for learning decision trees of size s over n variables. Their algorithm uses $n^{O(\log s)}$ examples, and they asked if the sample complexity could be reduced to $\text{poly}(n, s)$. We can apply our techniques here to give an algorithm using $2^{\tilde{O}(s^{1/3})} \log n$ examples, if we are willing to spend $n^{\tilde{O}(s^{1/3})}$ time:

Theorem 8. *Let D be a decision tree of size s over n variables. Then D can be learned with mistake bound $2^{\tilde{O}(s^{1/3})} \log n$ in time $n^{\tilde{O}(s^{1/3})}$.*

The proof is omitted because of space limitations in these proceedings.

5 Lower Bounds for Decision Lists

Here we observe that our construction from Theorem 7 is essentially optimal in terms of the tradeoff it achieves between polynomial threshold function degree and weight.

In [3], Beigel constructs an oracle separating PP from P^{NP}. At the heart of his construction is a proof that any low degree PTF for a particular decision list, called the ODDMAXBIT_n function, must have large weights:

Definition 1. *The ODDMAXBIT_n function on input $x = x_1, \ldots, x_n \in \{0,1\}^n$ equals $(-1)^i$ where i is the index of the first nonzero bit in x.*

It is clear that the ODDMAXBIT_n function is equivalent to a decision list $(x_1, -1), (x_2, 1), (x_3, -1), \ldots, (x_n, (-1)^n), (-1)^{n+1}$ of length n. The main technical theorem which Beigel proves in [3] states that any polynomial threshold function of degree d computing ODDMAXBIT_n must have weight $2^{\Omega(n/d^2)}$:

Theorem 9. *Let p be a degree d PTF with integer coefficients which computes ODDMAXBIT_n. Then $w = 2^{\Omega(n/d^2)}$ where w is the weight of p.*

(As stated in [3] the bound is actually $w \geq \frac{1}{s} 2^{\Omega(n/d^2)}$ where s is the number of nonzero coefficients in p. Since $s \leq w$ this implies the result as stated above.)

A lower bound of $2^{\Omega(n)}$ on the weight of any linear threshold function ($d = 1$) for ODDMAXBIT_n has long been known [24]; Beigel's proof generalizes this lower bound to all $d = O(n^{1/2})$. A matching upper bound of $2^{O(n)}$ on weight for $d = 1$ has also long been known [24]. Our Theorem 7 gives an upper bound which matches Beigel's lower bound (up to logarithmic factors) for all $d = O(n^{1/3})$:

Observation 10 *For any $d = O(n^{1/3})$ there is a polynomial threshold function of degree d and weight $2^{\tilde{O}(n/d^2)}$ which computes ODDMAXBIT_n.*

Proof. Set $d = h^{1/2} \log h$ in Theorem 7. The weight bound given by Theorem 7 is $2^{O(\frac{n \log^2 d}{d^2} + d \log d)}$ which is $\tilde{O}(n/d^2)$ for $d = O(n^{1/3})$. □

Note that since the ODDMAXBIT$_n$ function has a polynomial size DNF, Beigel's lower bound gives a polynomial size DNF f such that any degree $\tilde{O}(n^{1/3})$ polynomial threshold function for f must have weight $2^{\tilde{\Omega}(n^{1/3})}$. This suggests that the Expanded-Winnow algorithm cannot learn polynomial size DNF in $2^{\tilde{O}(n^{1/3})}$ time from $2^{n^{1/3-\epsilon}}$ examples for any $\epsilon > 0$, and thus suggests that improving the sample complexity of the DNF learning algorithm from [18] while maintaining its $2^{\tilde{O}(n^{1/3})}$ running time may be difficult.

6 Learning Parity Functions

6.1 A Polynomial Time Algorithm

Recall that the standard algorithm for learning parity functions works by viewing a set of m labelled examples as a set of m linear equations over GF(2). Gaussian elimination is used to solve the system and thus find a consistent parity. Even though there exists a solution of weight at most k (since the target parity is of size k), Gaussian elimination applied to a system of m equations in n variables over $GF(2)$ may yield a solution of weight as large as $\min(m, n)$. Thus this standard algorithm and analysis give an $O(n)$ sample complexity bound for learning a parity of length at most k.

We now describe a simple poly(n)-time algorithm for PAC learning an unknown size-k parity using $\tilde{O}(n^{1-1/k})$ examples. As far as we know this is the first improvement on the standard algorithm and analysis described above.

Theorem 11. *The class of all parity functions on at most k variables is PAC learnable in $O(n^4)$ time using $O(n^{1-1/k} \log n)$ examples. The hypothesis output by the learning algorithm is a parity function on $O(n^{1-1/k})$ variables.*

Proof. If $k = \Omega(\log n)$ then the standard algorithm suffices to prove the claimed bound. We thus assume that $k = o(\log n)$.

Let H be the set of all parity functions of size at most $n^{1-1/k}$. Note that $|H| \leq n^{n^{1-1/k}}$ so $\log |H| \leq n^{1-1/k} \log n$. Consider the following algorithm:

1. Choose $m = 1/\epsilon(\log |H| + \log(1/\delta))$ examples. Express each example as a linear equation over n variables mod 2 as described above.
2. Randomly choose a set of $n - n^{1-1/k}$ variables and assign them the value 0.
3. Use Gaussian elimination to attempt to solve the resulting system of equations on the remaining $n^{1-1/k}$ variables. If the system has a solution, output the corresponding parity (of size at most $n^{1-1/k}$) as the hypothesis. If the system has no solution, output "FAIL."

If the simplified system of equations has a solution, then by a standard Occam's Razor argument this solution is a good hypothesis. We will show that the simplified system has a solution with probability $\Omega(1/n)$. The theorem follows by repeating steps 2 and 3 of the above algorithm until a solution is found (an expected $O(n)$ repetitions will suffice).

Let V be the set of k relevant variables on which the unknown parity function depends. It is easy to see that as long as no variable in V is assigned a 0, the resulting simplified system of equations will have a solution. Let $\ell = n^{1-1/k}$. The probability that in Step 2 the $n-\ell$ variables chosen do not include any variables in V is exactly $\binom{n-k}{n-\ell}/\binom{n}{\ell}$ which equals $\binom{n-k}{\ell-k}/\binom{n}{\ell}$. Expanding binomial coefficients we have

$$\frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}} = \prod_{i=1}^{k} \frac{\ell-k+i}{n-k+i} > \left(\frac{\ell-k}{n-k}\right)^k = \left(\frac{\ell}{n}\right)^k \left(\frac{1-\frac{k}{\ell}}{1-\frac{k}{n}}\right)^k$$
$$= \frac{1}{n} \cdot \left[\left(1-\frac{k}{\ell}\right)\left(1+\frac{2k}{n}\right)\right]^k > \frac{1}{n}\left(1-\frac{k}{\ell}\right)^k > \frac{1}{n}\left(1-\frac{k^2}{\ell}\right) > \frac{1}{2n}$$

which proves the theorem. □

6.2 An $\tilde{O}(n^{k/2})$ Time Attribute Efficient Algorithm

Dan Spielman [30] has observed that it is possible to improve on the n^k time bound of a naive search algorithm for learning parity using $k \log n$ examples:

Theorem 12 (Spielman). *The class of all size-k parity functions is PAC learnable in $\tilde{O}(n^{k/2})$ time from $O(k \log n)$ examples, using size-k parities as the hypothesis class.*

Proof. By Occam's Razor we need only show that given a set of $m = O(k \log n)$ labelled examples, a consistent size-k parity can be found in $\tilde{O}(n^{k/2})$ time.

Given a labelled example $(x_1, \ldots, x_n; y)$ we will view y as an $(n+1)$st attribute x_{n+1}. Thus our task is to find a set of $(k+1)$ attributes $x_{i_1}, \ldots, x_{i_{k+1}}$, one of which must be x_{n+1}, which sum to 0 in every example in the sample.

Let $(x^1; y_1), \ldots (x^m; y_m)$ be the labelled examples in our sample. Given a subset S of variables, let v_S denote the length-m binary vector $(\chi_S(x^1), \ldots, \chi_S(x^m))$ obtained by computing the parity function χ_S on each example in our sample.

We construct two lists, each containing $\binom{n}{k/2}$ vectors of length m. The first list contains all the vectors v_S where S ranges over all $k/2$-element subsets of $\{x_1, \ldots, x_n\}$. The second list contains all the vectors $v_{S \cup \{x_{n+1}\}}$ where S again ranges over all $k/2$-element subsets of $\{x_1, \ldots, x_n\}$.

After sorting these two lists of vectors, which takes $\tilde{O}(n^{k/2})$ time, we scan through them in parallel in time linear in the length of the lists and find a pair of vectors v_{S_1} from the first list and $v_{S_2 \cup \{x_{n+1}\}}$ from the second list which are the same. (Note that any decomposition of the target parity into two subsets S_1 and S_2 of $k/2$ variables each will give such a pair). The set $S_1 \cup S_2$ is then a consistent parity of size k. □

7 Future Work

An obvious goal for future work is to improve our algorithmic results for learning decision lists. As a first step, one might attempt to extend the tradeoffs

we achieve: is it possible to learn decision lists of length k in $n^{k^{1/2}}$ time from $\mathrm{poly}(k, \log n)$ examples?

Another goal is to extend our results for decision lists to broader concept classes. In particular, it would be interesting to obtain analogues of our algorithmic results for learning general linear threshold functions (independent of their weight). We note here that Goldmann *et al.* [13] have given a linear threshold function over $\{-1, 1\}^n$ for which any polynomial threshold function must have weight $2^{\Omega(n^{1/2})}$ regardless of its degree. Moreover Krause and Pudlak [19] have shown that any Boolean function which has a polynomial threshold function over $\{0, 1\}^n$ of weight w has a polynomial threshold function over $\{-1, 1\}^n$ of weight n^2w^4. These results imply that *representational* results akin to Theorem 3 for general linear threshold functions must be quantitatively weaker than Theorem 3; in particular, there is a linear threshold function over $\{0, 1\}^n$ with k nonzero coefficients for which any polynomial threshold function, regardless of degree, must have weight $2^{\Omega(k^{1/2})}$.

For parity functions many questions remain as well: can we learn parity functions on $k = \Theta(\log n)$ variables in polynomial time using a sublinear number of examples? Can we learn size-k parities in polynomial time using fewer than $n^{1-1/k}$ examples? Can we learn size-k parities from $O(k \log n)$ examples in time $\tilde{O}(n^{k/3})$? Progress on any of these fronts would be quite interesting.

8 Acknowledgements

We thank Les Valiant for his observation that Claim 4.1 can be reinterpreted in terms of polynomial threshold functions, and we thank Jean Kwon for suggesting the Chebychev polynomial. We thank Dan Spielman for allowing us to include his proof of Theorem 12.

References

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

[2] J. Barzdin and R. Freivald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

[3] R. Beigel. When do extra majority gates help? polylog(n) majority gates are equivalent to one. *Computational Complexity*, 4:314–324, 1994.

[4] A. Blum. Learning boolean functions in an infinite attribute space. In *Proceedings of the Twenty-Second Annual Symposium on Theory of Computing*, pages 64–72, 1990.

[5] A. Blum. On-line algorithms in machine learning. available at `http://www.cs.cmu.edu/~avrim/Papers/pubs.html`, 1996.

[6] A. Blum. Empirical support for winnow and weighted-majority algorithms: results on a calendar scheduling domain. *Machine Learning*, 26:5–23, 1997.

[7] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *Journal of Computer and System Sciences*, 50:32–40, 1995.

[8] A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[9] E. Cheney. *Introduction to approximation theory.* McGraw-Hill, New York, New York, 1966.

[10] A. Dhagat and L. Hellerstein. Pac learning with irrelevant attributes. In *Proceedings of the Thirty-Fifth Annual Symposium on Foundations of Computer Science*, pages 64–74, 1994.

[11] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

[12] A.R. Golding and D. Roth. A winnow-based approach to spelling correction. *Machine Learning*, 34:107–130, 1999.

[13] M. Goldmann, J. Hastad, and A. Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992.

[14] D. Haussler. Space efficient learning algorithms. Technical Report UCSC-CRL-88-2, University of California at Santa Cruz, 1988.

[15] D. Helmbold, R. Sloan, and M. Warmuth. Learning integer lattices. *SIAM Journal on Computing*, 21(2):240–266., 1992.

[16] J. Kivinen, M. Warmuth, and P. Auer. The perceptron algorithm vs. winnow: linear vs. logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, 97(1-2):325–343, 1997.

[17] A. Klivans, R. O'Donnell, and R. Servedio. Learning intersections and thresholds of halfspaces. In *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science*, 2002.

[18] A. Klivans and R. Servedio. Learning dnf in time $2^{\tilde{O}(n^{1/3})}$. In *Proceedings of the Thirty-Third Annual Symposium on Theory of Computing*, pages 258–265, 2001.

[19] M. Krause and P. Pudlak. Computing boolean functions by polynomials and threshold circuits. *Computational Complexity*, 7(4):346–370, 1998.

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

[21] N. Littlestone. From online to batch learning. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 269–284, 1989.

[22] N. Littlestone. *Mistake bounds and logarithmic linear-threshold learning algorithms.* PhD thesis, University of California at Santa Cruz, 1989.

[23] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[24] J. Myhill and W. Kautz. On the size of weights required for linear-input switching functions. *IRE Trans. on Electronic Computers*, EC10(2):288–290, 1961.

[25] Z. Nevo and R. El-Yaniv. On online learning of decision lists. *Journal of Machine Learning Research*, 3:271–301, 2002.

[26] R. O'Donnell and R. Servedio. New degree bounds for polynomial threshold functions. Proceedings of the 35th ACM Symposium on Theory of Computing, 2003.

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

[28] R. Servedio. Computational sample complexity and attribute-efficient learning. *Journal of Computer and System Sciences*, 60(1):161–178, 2000.

[29] R. Servedio. Perceptron, Winnow and PAC learning. *SIAM Journal on Computing*, 31(5):1358–1369, 2002.

[30] D. Spielman. Personal communication, 2003.

[31] R. Uehara, K. Tsuchida, and I. Wegener. Optimal attribute-efficient learning of disjunction, parity, and threshold functions. In *Proceedings of the Third European Conference on Computational Learning Theory*, pages 171–184, 1997.

[32] L. Valiant. Projection learning. *Machine Learning*, 37(2):115–130, 1999.