Toward Attribute Efficient Learning Algorithms

Adam R. Klivans*
Divsion of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
klivans@eecs.harvard.edu

Rocco A. Servedio
Department of Computer Science
Columbia University
New York, NY 10027
rocco@cs.columbia.edu

Abstract

We make the first progress on two important problems regarding attribute efficient learnability. 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 establishes a relationship between attribute efficient learning and polynomial threshold functions and 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 1994 lower bound due to Beigel for the ODDMAXBIT predicate and gives an essentially optimal tradeoff between polynomial threshold function degree and weight. Secondly, we give an algorithm for learning an unknown parity function on k out of n variables using $O(n^{1-1/k})$ examples in time polynomial in n. For k equal to $o(\log n)$ this yields a polynomial time algorithm with sample complexity $o(n)$. This is the first polynomial time algorithm for learning parity on a superconstant number of variables with sublinear sample complexity.

*Supported by an NSF Mathematical Sciences Postdoctoral Research Fellowship.

1 Introduction

1.1 Attribute Efficient Learning

A central goal in machine learning is to design efficient, effective algorithms for learning from small amounts of data. An obstacle to achieving this goal is that learning problems are often characterized by an abundance of *irrelevant information.* In many learning problems each data point is naturally viewed as a high dimensional vector of attribute values; as a motivating example, in a natural language domain a data point representing a text document may be a vector of word frequencies over a lexicon of 100,000 words (attributes). A newly encountered word in a corpus may typically have a simple definition which uses only a dozen or so words from the entire lexicon. One would like to be able to learn the meaning of such a word using a number of examples which is closer to a dozen (the actual number of relevant attributes) than to 100,000 (the total number of attributes).

Towards this end, 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 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, say, the class of Boolean conjunctions must be able to learn any Boolean conjunction of k literals over $x_1, \ldots, x_n$ using poly$(k, \log n)$ examples, since $k \log n$ bits are required to specify such a conjunction.

1.2 Decision Lists

A longstanding open problem in machine learning, posed first by Blum in 1990 [4, 6, 8, 10] and again by Valiant in 1998 [34], is to determine whether or not there exist attribute efficient algorithms for learning *decision lists.* A decision list is essentially a nested "if-then-else" statement (we give a precise definition in Section 2).

Attribute efficient learning of decision lists is of both theoretical and practical interest. Blum's motivation for considering the problem came from the *infinite attribute model* [4]; in this model there are infinitely many attributes but the concept to be learned depends on only a small number of them, and each example consists of a finite list of active attributes. Blum *et al.* [8] showed that for a wide range of 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.[1] Several researchers have subsequently considered this problem, see e.g. [6, 10, 12, 29, 32].

From an applied perspective, Valiant [34] relates the problem of learning decision lists attribute efficiently to the question "how can human beings learn from small amounts of data in the presence of irrelevant information?" He points out that since decision lists play an important role in various models of cognition, a first step in understanding this phenomenon would be to identify efficient algorithms which learn decision lists from few examples. Due to the lack of progress in developing such algorithms for decision lists, Valiant suggests that models of cognition should perhaps focus on "flatter" classes of functions such as projective DNF [34].

[1] Additional motivation comes from the fact that decision lists have such a simple algorithm in the PAC model.

1.3 Parity Functions

Another outstanding challenge in machine learning is the problem of learning parity functions attribute efficicently. The parity function on a set of variables $x_1, \ldots, x_k$ is defined to be 1 on any $\{0,1\}$ assignment where an even number of the variables take value 1, and -1 otherwise. Aside from decision lists, the concept class of parity functions is the most notorious concept class for which we have a polynomial time PAC learning algorithm but do not know how to attribute efficiently learn. [gag on that sentence] Learning parity functions has been studied extensively in computational learning theory and plays an important rule in Fourier learning methods [27] as well as decoding random linear codes [9]. Both A. Blum [6] and Y. Mansour [25] cite attribute efficient learning of parity functions as an important open problem.

Given a set of examples labelled according to an unknown parity function on k out of n variables, we wish to find an approximation to the unknown parity in polynomial time using as few examples as possible. The well known solution to this problem views these examples as a set of linear equations mod 2 in n variables and solves the set of equations to come up with a consistent hypothesis. Note, however, that we must take $\Omega(n)$ examples to achieve a solution which has good generalization error, as a solution to a system of m equations over n variables may contain $\min(m,n)$ non-zero entries. An attribute efficient algorithm for learning parity should require a number of examples polynomially related to k and $\log n$ (information theoretically we should only need $O(k \log n)$ examples).

1.4 Our Results: Decision Lists

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 indeed be possible to learn decision lists attribute efficiently.

Our main learning result is:

Theorem 1 *There is an algorithm for learning decision lists over $\{0,1\}^n$ which, when learning a decision list of length k, has mistake bound*[2] $2^{\tilde{O}(k^{1/3})} \log n$ *and runs in time* $n^{\tilde{O}(k^{1/3})}$.

We prove Theorem 1 in two parts; first we generalize Littlestone's well known Winnow algorithm [22] for learning linear threshold functions to learn *polynomial threshold functions.* In previous learning results, polynomial threshold functions are learned by applying techniques from linear programming: a Boolean function computed by a polynomial threshold function of degree d can be learned in time $n^{O(d)}$ by using polynomial time linear programming algorithms such as the Ellipsoid algorithm (see e.g. [20]). In contrast, we use the Winnow algorithm to learn polynomial threshold functions. Winnow learns using few examples in a small amount of time provided that the degree of the polynomial is low and the integer coefficients of the polynomial are not too large:

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 there is an online learning algorithm for $\mathcal{C}$ which runs in n^d time per example and has mistake bound $O(W^2 \cdot d \cdot \log n)$.*

[2]Throughout this section we use "sample complexity" and "mistake bound" interchangeably; as described in Section 2 these notions are essentially identical.

At this point we have reduced the problem of learning decision lists attribute efficiently to the problem of representing decision lists with polynomial threshold functions 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.

Polynomial threshold function constructions have recently been used to obtain the fastest known algorithms for a range of important learning problems such as learning DNF formulas [20], intersections of halfspaces [19], and Boolean formulas of superconstant depth [30]. For each of these learning problems the sole goal was to obtain fast learning algorithms, and hence the only parameter of interest in these polynomial threshold function constructions is their degree, since degree bounds translate directly into running time bounds for learning algorithms (see e.g. [20]). In contrast, for the decision list problem we are interested in both the running time and the number of examples required for learning. Thus we must bound both the degree and the *weight* (magnitude of integer coefficients) of the polynomial threshold functions which we use.

Our polynomial threshold function construction is essentially optimal in the tradeoff between degree and weight which it achieves. In 1994 Beigel gave a lower bound showing that any degree d polynomial threshold function for a particular decision list must have weight $2^{\Omega(n/d^2)}$. For $d = n^{1/3}$, Beigel's lower bound implies that the construction stated in Theorem 3 is essentially optimal. Furthermore, for any decision list L of length n and any $d \leq n^{1/3}$, we will in fact construct polynomial threshold functions of degree d and weight $2^{\tilde{O}(n/d^2)}$ computing L. Beigel's lower bound thus implies that our degree d polynomial threshold functions are of roughly optimal weight for all $d \leq n^{1/3}$, and hence strongly suggests that our analysis is the best possible for the algorithm we use.

1.5 Our Results: Parity Functions

We give an algorithm for learning an unknown parity function on k variables using $O(n^{1-1/k})$ examples which runs in time $O(n^3)$. For values of $k = o(\log n)$ the sample complexity of this algorithm is $o(n)$. 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 linear equations. A system of m equations over n variables may have a solution with m nonzero entries. If $k < n$, however, we prove that with high probability we can find a solution of weight $O(n^{1-1/k})$ irrespective of m. Hence, for Occam's Razor to apply, we need only choose m to be slightly more than $n^{1-1/k}$, and we obtain our improved bounds on sample complexity.

1.6 Previous Results: Decision Lists

In previous work several algorithms with different performance bounds (in terms of running time and number of examples used) have been given for learning decision lists.

- Rivest [31] gave the first algorithm for learning decision lists in Valiant's PAC model of learning from random examples. Littlestone [6] subsequently 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 to learning decision lists of length k is to maintain a collection of all such 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 by Barzdin and Freivald [2], Mitchell [26], and Angluin [1]) can learn decision lists of length k using only $O(k \log n)$ examples, but the running time is $n^{O(k)}$.

- Several researchers [6, 34] have observed that Littlestone's well-known Winnow algorithm [22] can learn decision lists of length k from $2^{O(k)} \log n$ examples in time $2^{O(k)} n \log n$. This follows from the observation that decision lists of length k can be viewed as linear threshold functions with integer coefficients of magnitude $2^{\Theta(k)}$. We note that our algorithm in this paper always has improved sample complexity over the basic Winnow algorithm, and for $k \geq (\log n)^{3/2}$ our approach improves on the time complexity of Winnow as well.

- Finally, several researchers have considered the special case of learning a decision list of length k over n variables in which the output bits of the decision list have at most D alternations. Valiant [34] and Nevo and El-Yaniv [29] have given refined analyses of Winnow's performance for this special case, and Dhagat and Hellerstein [12] have also studied this problem. However, for the general case in which D can be as large as k, the results thus obtained do not improve on the straightforward Winnow analysis described in the previous bullet.

These previous algorithmic results are summarized in Figure 1. We observe 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.

Reference:	Number of examples:	Running time:
Rivest / Littlestone	$O(kn)$	$O(kn^2)$
Halving algorithm	$O(k \log n)$	$n^{O(k)}$
Winnow algorithm	$2^{O(k)} \log n$	$2^{O(k)} n \log n$
This Paper	$2^{\tilde{O}(k^{1/3})} \log n$	$n^{\tilde{O}(k^{1/3})}$

Table 1: Comparison of known algorithms for learning decision lists of length k on n variables.

1.7 Previous Results: Parity Functions

Very little previous work exists with regard to learning parity functions attribute efficiently. The standard parity learning algorithm of solving a system of linear equations appears in work due to Helmbold et al.[17] and has sample complexity $\Omega(n)$. Several authors have considered learning parity attribute efficiently when the learner is allowed to make membership queries. Achieving attribute efficiency is simpler, however, if the learner is allowed membership queries, as they facilitate identifying relevant variables. In fact, Blum et al. [8] give a randomized, membership-query algorithm for learning parity on k variables using only $O(k \log n)$ examples in polynomial time.

1.8 Organization

In Section 2 we give the necessary background on online learning and polynomial threshold functions. In Section 3 we show how known results from learning theory enable us to reduce the decision list learning problem to a problem of finding suitable polynomial threshold function representations of decision lists. In Sections 4.1 and 4.2 we give two different proofs of a weak tradeoff between degree and weight for polynomial threshold function representations of decision lists, and in Section 4.3 we combine these techniques to prove Theorem 3. In Section 5 we show how to apply our techniques to give a tradeoff between sample complexity and running time for learning decision trees. In Section 6 we discuss the connection with Beigel's ODDMAXBIT lower bound and related issues. In Section 7 we give our new algorithm for learning parity functions, and in Section 8 we suggest directions for future work.

2 Preliminaries

Attribute efficient learning has been chiefly studied in the *on-line mistake-bound* model of concept learning which was introduced in [22, 24]. 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 measured by 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 in this paper 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 $|S| = k$. We define the parity function $\chi_S(x)$ to be 1 for an input setting an even number of variables in S to be 1 and -1 otherwise.

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. (Note that if f has r relevant variables then size(f) will be at least $r \log n$ since this many bits are required just to specify which variables are relevant). 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 $c \in C$ is polynomial in size(f). In particular, the description length of a length k decision list is $O(k \log n)$, and thus we would ideally like to have an algorithm which learns decision lists of length k with a mistake bound of poly$(k, \log n)$ and runs in time poly(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, 16, 23] which can be used to transform any mistake bound algorithm into a PAC learning algorithm. This transformation essentially preserves 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 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* of degree d and weight W for f.

3 Expanded-Winnow: Learning Polynomial Threshold Functions

Littlestone introduced the online Winnow algorithm in 1988 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 a novel update rule which makes a *multiplicative* update to each coefficient of the hypothesis (rather than an additive update as in the Perceptron algorithm) each time a mistake is made. Since its introduction Winnow has been intensively studied from both applied and theoretical standpoints (see e.g. [7, 14, 18, 33]) and multiplicative updates have become widespread in machine learning algorithms.

The following theorem (which, as noted in [34], is implicit in Littlestone's analysis in [22]) gives a mistake bound for Winnow when learning linear threshold functions:

Theorem 4 *Let $f(x)$ be the linear threshold function $sign(\sum_{i=1}^n w_i x_i - \theta)$ 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, called 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 corresponds to Expanded-Winnow's running time, and the weight of a polynomial threshold function corresponds to its sample complexity.

4 Constructing Polynomial Threshold Functions for Decision Lists

In previous constructions of polynomial threshold functions for computational learning theory applications [20, 19, 30] the sole goal has been to minimize the degree of the polynomials regardless of the size of the coefficients. As an extreme example, the construction of [20] of $\tilde{O}(n^{1/3})$ degree polynomial threshold functions for DNF formulae yields polynomials whose coefficients can be *doubly exponential* in the degree. In contrast, given Theorem 2 we must now construct polynomial threshold functions that have low degree and low weight.

We give two constructions of polynomial threshold functions for decision lists, each of which has relatively low degree and relatively low weight. We then combine these approaches 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 $\frac{2k}{h} 2^{(k/h+h)}$ polynomial threshold function 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 5 *The decision list L is eqivalent to*

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

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 5 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 polynomial threshold function for L. We can do this from Claim 5 as follows:

Theorem 6 *Let L be a decision list of length k. Then 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 polynomial threshold function 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 7 *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 [20] can be used to give an alternate proof of Corollary 7 with slightly weaker parameters; see Appendix A.

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 8 *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 6 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 [20], 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 [11]:

- $|C_d(x)| \leq 1$ 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 bound on 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 9 *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 10 *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 5 we have that

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

where C is the value from Corollary 9 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 9, i.e. consider $\mathrm{sign}(H(x))$ where

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

We will show that $\mathrm{sign}(H(x))$ is a polynomial threshold function 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 9 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 9, 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 $\mathrm{sign}(H(x))$ is a valid polynomial threshold representation for $L(x)$. Finally, our degree and weight bounds from Corollary 9 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 $n^{\tilde{O}(k^{1/3})}$.

5 Application to Learning Decision Trees

In 1989 Ehrenfeucht and Haussler [13] 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 $\mathrm{poly}(n, s)$. 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.

First we need to generalize Theorem 10 for higher order decision lists. An r-decision list is like a standard decision list but each pair is now of the form (C_i, b_i) where C_i is a conjunction of at most r literals and as before $b_i = \pm 1$. The output of such an r-decision list on input x is b_i where i is the smallest index such that $C_i(x) = 1$.

We have the following:

Corollary 11 *Let L be an r-decision list of length k. Then for any $h < k$, L is computed by a polynomial threshold function of degree $O(rh^{1/2} \log h)$ and weight $2^{r+O(k/h+h^{1/2} \log^2 h)}$.*

Proof: Let L be the r-decision list $(C_1, b_1), \ldots, (C_k, b_k), b_{k+1}$. By Theorem 10 there is a polynomial threshold function of degree $O(h^{1/2} \log h)$ and weight $2^{O(k/h + h^{1/2} \log^2 h)}$ over the variables $C_1, \ldots, C_k$. Now replace each variable C_i by the interpolating polynomial which computes it exactly as a function from $\{0,1\}^n$ to $\{0,1\}$. Each such interpolating polynomial has degree r and integer coefficients of total magnitude at most 2^r, and the corollary follows. □

Corollary 12 *There is an algorithm for learning r-decision lists over $\{0,1\}^n$ which, when learning an r-decision list of length k, has mistake bound $2^{\tilde{O}(r+k^{1/3})} \log n$ and runs in time $n^{\tilde{O}(rk^{1/3})}$.*

Now we can apply Corollary 12 to obtain a tradeoff between running time and sample complexity for learning decision trees:

Theorem 13 *Let D be a decision tree of size s over n variables. Then D can be learned using $2^{\tilde{O}(s^{1/3})} \log n$ examples in time $n^{\tilde{O}(s^{1/3})}$.*

Proof: Blum [5] has shown that any decision tree of size s is computed by a $(\log s)$-decision list of length s. Applying Corollary 12 we thus see that Expanded-Winnow can be used to learn decision trees of size s over $\{0,1\}^n$ with the claimed bounds on time and sample complexity. □

6 Lower Bounds for Decision Lists

Here we observe that our construction from Theorem 10 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 polynomial threshold function for a particular decision list, called the the $\mathrm{ODDMAXBIT}_n$ function, must have large weights:

Definition 14 *The* $\mathrm{ODDMAXBIT}_n$ *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 $\mathrm{ODDMAXBIT}_n$ function is equivalent to a decision list of length n:

$$(x_1, -1), (x_2, 1), (x_3, -1), \ldots, (x_n, (-1)^n), (-1)^{n+1}.$$

The main technical theorem which Beigel proves in [3] states that any polynomial threshold function of degree d computing $\mathrm{ODDMAXBIT}_n$ must have weight $2^{\Omega(n/d^2)}$:

Theorem 15 *Let p be a degree d polynomial threshold function with integer coefficients computing* $\mathrm{ODDMAXBIT}_n$. *Then $w = 2^{\Omega(n/d^2)}$ where w is the weight of p.*[3]

(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 $\mathrm{ODDMAXBIT}_n$ has long been known [28]; 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 [28]. Our Theorem 10 gives an upper bound which matches Beigel's lower bound (up to logarithmic factors) for all $d = O(n^{1/3})$:

[3] Beigel actually proves something stronger, namely that there must exists a coefficient whose absolute value is at least $2^{\Omega(n/d^2)}$.

Observation 16 *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* $\mathrm{ODDMAXBIT}_n$.

Proof: Set $d = h^{1/2} \log h$ in Theorem 10. The weight bound given by Theorem 10 is $2^{O(\frac{n \log^2 d}{d^2} + d \log d)}$ which is $\tilde{O}(n/d^2)$ for $d = O(n^{1/3})$. $\square$

Note that since the $\mathrm{ODDMAXBIT}_n$ function has a polynomial size DNF (see Appendix A), 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 [20] while maintaining its $2^{\tilde{O}(n^{1/3})}$ running time may be difficult.

7 Learning Parity Functions

Standard algorithms for learning parity functions proceed by viewing a set of examples as a set of linear equations over GF(2). A consistent hypothesis is found by solving this system of equations, and this algorithm requires $\Omega(n)$ samples to guarantee finding a hypothesis with good generalization error. In this section we present a simple polynomial-time algorithm for learning an unknown parity function on k variables using $O(n^{1-1/k})$ examples. We will need the following standard fact which gives a simple criterion for PAC learnability:

Fact 17 *Let C be a concept class and H a finite set of hypotheses. Set $m = 1/\epsilon(\log |H| + \log 1/\delta)$ where ϵ and δ are the standard PAC learning error parameters. Assume that there exists an algorithm running in time t which takes as input m examples classified by an element of C and outputs a hypothesis $h \in H$ consistent with the data. Then C is PAC learnable by H in time t.*

Note that an example labelled according to an unknown parity function χ corresponds to an equation modulo 2 in the obvious way: given an example E, let S equal the set of indices which contain a 1. Let $L = 1$ if the label of example E is 1 and let $L = 0$ otherwise. Then example E induces the equation $\sum_{i \in S} a_i = L \bmod 2$.

Now we can prove the main theorem of this section:

Theorem 18 *Let C be the class of parity functions on $k < n$ variables. Then C is learnable in polynomial-time using $O(n^{1-1/k})$ samples. Furthermore 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 our claimed bound on sample complexity equals $\Omega(n)$. Since we are only interested in algorithms which have sample complexity $o(n)$, we can assume without loss of generality that $k = o(\log n)$. Now consider the following algorithm:

1. Choose $m = 1/\epsilon(\log(n^{1-1/k}) + \log(1/\delta))$ examples. View each example as a linear equation over n variables mod 2.

2. Randomly choose a set of $n - n^{1-1/k}$ variables and assign them the value 0.

3. Output the solution to this simplified system of equations as the hypothesis. If no solution exists output "Fail."

If our algorithm outputs a solution to the simplified system of equations then by Fact 17 this solution is a good hypothesis. We will prove that with probability $1/n$ this occurs; by repeating steps 2 and 3 of the above algorithm, the statement of the theorem follows.

Let V be the set of k relevant variables on which the unknown parity function depends. It is easy to see that if we assign a variable not in V equal to 0, the resulting system of equations still has a solution. Now we calculate the probability of success: the probability of not randomly choosing *any* variables from V. Let $l = n^{1-1/k}$. The probability of success is $\binom{n}{l-k}/\binom{n}{l}$. A simple expansion of binomial coefficients shows this is equal to $\prod_{i=1}^{k}(l-k+i)/(n-l+i)$. Substituting $k = o(\log n)$ and $l = n^{1-1/k}$ this quantity is approximately $(n^{1-1/k}/n)^k = 1/n$.

$\square$

8 Future Work

An obvious goal for future work is to improve our algorithmic results for learning decision lists. The question still remains: can decision lists of length k be learned in poly(n) time from poly$(k, \log n)$ examples? 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 poly$(k, \log n)$ examples?

Another goal is to extend our results for decision lists to broader concept classes. In particular, since decision lists are a special case of linear threshold functions, 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.* [15] 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 [21] 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 $2^{\Omega(k^{1/2})}$.

With regard to learning parity functions, the obvious challenge is to learn parity functions on $k = O(\log n)$ variables in polynomial time using a sublinear number of examples.

9 Acknowledgements

We thank Les Valiant for his observation that Claim 5 can be reinterpreted in terms of polynomial threshold functions. We thank Jean Kwon for suggesting the Chebychev polynomial.

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. Rank-r decision trees are a subclass of r-decision lists. *Information Processing Letters*, 42(4):183–185, 1992.

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

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

[8] 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.

[9] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Proceedings of the Thirty-Second Annual Symposium on Theory of Computing*, 2000.

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

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

[12] 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.

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

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

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

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

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

[18] 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.

[19] 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.

[20] 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.

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

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

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

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

[25] Y. Mansour. Lecture notes # 5 from "machine learning: Foundations" course 2001/2002. available at `http://www.math.tau.ac.il/ mansour/ml-course-02`, 2001.

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

[27] E. Mossell, R. O'Donnell, and R. Servedio. Learning juntas. *Proceedings of the Thirty-Fifth Annual Symposium on Theory of Computing*, 2003.

[28] 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.

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

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

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

[32] R. Servedio. Computational sample complexity and attribute-efficient learning. In *Proceedings of the Thirty-First Annual Symposium on Theory of Computing*, pages 701–710, 1999.

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

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

A Alternate Proof of Corollary 7

The alternate proof of Corollary 7 is based on the observation that any decision list $L = (\ell_1, b_1), \ldots, (\ell_k, b_k), b_{k+1}$ of length k has a k-term DNF in which each term is a conjunction of at most k literals. To see this, note that we obtain a DNF for L simply by taking the OR of all terms $\overline{\ell}_1\overline{\ell}_2 \ldots \overline{\ell}_{i-1}\ell_i$ for each i such that $b_i = 1$. Now we use the following result from [20]:

Theorem 19 (Corollary 12 of [20]) *Let f be a DNF formula of s terms, each of length at most t. Then there is a polynomial threshold function for f of degree $O(\sqrt{t}\log s)$ and weight $t^{O(\sqrt{t}\log s)}$.*

Applying this result to the DNF representation for L, we immediately obtain that there is a polynomial threshold function for L which has degree $O(k^{1/2}\log k)$ and weight $2^{O(k^{1/2}\log^2 k)}$. (In Section 4.2, though, we need the construction given in our original proof of Corollary 7.)