On PAC learning algorithms for rich Boolean function classes

Rocco A. Servedio*

Department of Computer Science
Columbia University
New York, NY U.S.A.
rocco@cs.columbia.edu

Abstract. We survey the fastest known algorithms for learning various expressive classes of Boolean functions in the Probably Approximately Correct (PAC) learning model.

1 Introduction

Computational learning theory is the study of the inherent abilities and limitations of algorithms that learn from data. A broad goal of the field is to design computationally efficient algorithms that can learn Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$. A general framework within which this question is often addressed is roughly the following:

1. There is a fixed class C of possible target functions over $\{0,1\}^n$ which is *a priori* known to the learning algorithm. (Such function classes are often referred to as *concept classes*, and the functions in such classes are referred to as *concepts*.)
2. The learning algorithm is given some form of access to information about the unknown target concept $c \in C$.
3. At the end of its execution, the learning algorithm outputs a hypothesis $h\colon \{0,1\}^n \to \{0,1\}$, which ideally should be equivalent or close to c.

Different ways of instantiating (2) and (3) above – what form of access to c is the learner given? what is required of the hypothesis function h? etc. – give rise to different learning models. Within a given learning model, different choices of the Boolean function class C (i.e. different ways of instantiating (1) above) give rise to different learning problems such as the problem of learning an unknown conjunction, learning an unknown linear threshold function, an unknown decision tree, and so on.

In this brief survey we will focus exclusively on the widely studied Probably Approximately Correct (PAC) learning model introduced by Valiant [39]. In this learning model, which we define precisely in Section 2.1, the learning algorithm

* Supported in part by NSF CAREER award CCF-0347282, by NSF award CCF-0523664, and by a Sloan Foundation Fellowship.

is only given access to *independent random examples labelled according to c,* i.e. access to input-output pairs $(x, c(x))$ where each x is independently drawn from the same unknown probability distribution. Thus the learning algorithm has no control over the choice of examples used for learning. Such a model may be viewed as a good first-order approximation of commonly encountered scenarios in machine learning where one must learn from a given training set of examples generated according to some unknown random process.

(We note that a wide range of models exist in which the learning algorithm has other forms of access to the target function; in particular several standard models allow the learner to make black-box queries to the target function, which are often known as *membership queries*. Many powerful and elegant learning algorithms are known in various models that permit membership queries, see e.g. [1, 4, 10, 19, 28], but we will not discuss this work here. A rich body of results have also been obtained for the *uniform-distribution* variant of the PAC learning model, in which the learner need only succeed when given uniform random examples from $\{0,1\}^n$; see e.g. [40, 29, 12, 20, 34] for some representative work in this setting. Finally, we note that there also exist well-motivated and well-studied learning models in which the learning algorithm only has some more limited form of access to c than random labeled examples, see e.g. [5, 21].)

There are well-known polynomial-time PAC learning algorithms for concept classes consisting of simple functions such as conjunctions and disjunctions [39], decision lists [35], parity functions [15, 18], and halfspaces [9]. We give a concise overview of the current state of the art for learning richer concept classes consisting of more expressive Boolean functions such as decision trees, Disjunctive Normal Form (DNF) formulas, intersections of halfspaces, and various restricted classes of Boolean formulas. For each of these "rich" concept classes true polynomial-time algorithms are not (yet) known, but as we describe below, it is possible to give provable guarantees which improve substantially over naive exponential runtime bounds.

One perhaps surprising point which emerges from our survey is that a single linear programming based algorithm for learning *polynomial threshold functions* gives the current state-of-the-art results for learning a wide range of rich concept classes, including all those we will discuss in Section 3. We close the survey in Section 4 with a brief description of a very different approach to obtaining PAC learning algorithms, based on linear algebra rather than linear programming, which is also of interest. While to date this linear algebraic approach has not yielded as many results for learning rich concept classes as the polynomial threshold function approach, we feel that it presents an interesting direction for future study.

Throughout the survey we highlight various open questions, with an emphasis on problems where progress both would be of interest and (in the view of the author) would seem most likely to be feasible.

2 Distribution-Independent Learning

2.1 The learning model

In an influential 1984 paper Valiant introduced the *Probably Approximately Correct* (PAC) model of learning Boolean functions from random examples [39]. (See the book [22] for an excellent and detailed introduction to the model.) In the PAC model a learning algorithm has access to an *example oracle* $EX(c, \mathcal{D})$ which, when queried, provides a labeled example $(x, c(x))$ where x is drawn from a fixed but unknown distribution $\mathcal{D}$ over $\{0,1\}^n$ and $c \in C$ is the unknown target concept which the algorithm is trying to learn. Given Boolean functions h, c on $\{0,1\}^n$, we say that h is an *ϵ-approximator for c under $\mathcal{D}$* if $\Pr_{x \in \mathcal{D}}[h(x) = c(x)] \geq 1 - \epsilon$. The goal of a PAC learning algorithm is to output a hypothesis h which is an ϵ-approximator for the unknown target concept c with high probability.

More precisely, an algorithm A is a *PAC learning algorithm for concept class* C if the following condition holds: for any $c \in C$, any distribution $\mathcal{D}$ on $\{0,1\}^n$, and any $0 < \epsilon < \frac{1}{2}, 0 < \delta < 1$, if A is given ϵ, δ as input and has access to $EX(c, \mathcal{D})$, then A outputs (a representation of) some $h\colon \{0,1\}^n \to \{0,1\}$ which satisfies $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$ with probability at least $1 - \delta$. We say that *A PAC learns C in time $t = t(n, \epsilon, \delta, s)$* if A runs for at most t time steps and outputs a hypothesis h which can be evaluated on any point $x \in \{0,1\}^n$ in time t; here $s = \text{size}(c)$ is a measure of the "size" of the target concept $c \in C$. Note that no restriction is put on the form of the hypothesis h other than that it be efficiently evaluatable. In particular, h need not belong to the concept class C (i.e. we do not restrict ourselves to proper learning algorithms).

It is well known (see e.g. [22]) that the runtime dependence of a PAC learning algorithm on δ can always be made logarithmic in $\frac{1}{\delta}$. Moreover, for all the results we discuss the runtime dependence on ϵ is polynomial in $\frac{1}{\epsilon}$. Thus throughout this paper we discuss the running time of PAC learning algorithms as functions only of n and (when appropriate) the size parameter s.

2.2 The main technique: polynomial threshold functions

A polynomial threshold function is defined by a polynomial $p(x_1, \ldots, x_n)$ with real coefficients. The output of the polynomial threshold function on input $x \in \{0,1\}^n$ is 1 if $p(x_1, \ldots, x_n) \geq 0$ and is 0 otherwise. The *degree* of a polynomial threshold function is simply the degree of the polynomial p. A *linear threshold function* or *halfspace* is a polynomial threshold function of degree 1. Since we will only be concerned with the input space $\{0,1\}^n$, we may without loss of generality only consider polynomial threshold functions which correspond to multilinear polynomials.

It is well known that there are poly(n)-time PAC learning algorithms for the concept class of linear threshold functions over $\{0,1\}^n$; this follows from information-theoretic sample complexity arguments [8, 9] combined with the existence of polynomial-time algorithms for linear programming [23]. As various

authors have noted [7, 26], such algorithms can be run over an expanded feature space of $N = \sum_{i=1}^{d} \binom{n}{d}$ monomials of degree at most d to learn degree-d polynomial threshold functions in time poly(N). (This approach is closely related to using a Support Vector Machine with a degree-d polynomial kernel, see e.g. [36].) We thus have the following:

Fact 1 *Let C be a class of functions each of which can be expressed as an degree-d polynomial threshold function over $\{0,1\}^n$. Then there is a poly(N)-time PAC learning algorithm for C, where $N = \sum_{i=1}^{d} \binom{n}{i} \leq (\frac{en}{d})^d$.*

Thus, in order to get an upper bound on the runtime required to learn a concept class C, it is enough to bound the degree of polynomial threshold functions which represent the concepts in C. This approach has proved quite powerful as we now describe.

3 Known results on learning rich Boolean function classes

3.1 Decision Trees

A *Boolean decision tree* T is a rooted binary tree in which each internal node has two ordered children and is labeled with a variable, and each leaf is labeled with a bit $b \in \{-1, +1\}$. The *size* of a decision tree is the number of leaves. A decision tree T computes a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ in the obvious way: on input x, if variable x_i is at the root of T we go to either the left or right subtree depending on whether x_i is 0 or 1. We continue in this way until reaching a bit leaf; the value of this bit is $f(x)$.

Algorithms for learning decision trees have received much attention both from applied and theoretical perspectives. Ehrenfeucht and Haussler [14] gave a recursive algorithm which learns any size-s decision tree in $n^{O(\log s)}$ time; while no faster algorithms are known, various alternate algorithms with the same quasipolynomial runtime have since been given. Blum [6] showed that every size-s decision tree is equivalent to some $\log(s)$-*decision list*. (An r-decision list is a sequence of nested "if-then" rules where each "if" condition is a conjunction of at most r literals and each "then" statement is of the form "output bit b.") Since r-decision lists are PAC learnable in $n^{O(r)}$ time [35], this gives an equally efficient alternative algorithm to [14].

Blum's proof is easily seen to establish that any size-s decision tree is computed by a $\log(s)$-degree polynomial threshold function. Thus for decision trees we may use Fact 1 to obtain the fastest known algorithm, but as described above other equally fast algorithms are also known. However, for each of the concept classes discussed below in Sections 3.2 through 3.3, the Fact 1 approach is the only known way to achieve the current fastest runtimes.

3.2 DNF formulas

A *disjunctive normal form* formula, or DNF, is a disjunction $T_1 \vee \cdots \vee T_s$ of conjunctions of Boolean literals. An s-term DNF is one which has at most s

conjunctions (also known as terms). Learning s-term DNF formulas in time $\text{poly}(n, s)$ is a longstanding open question which goes back to Valiant's inception of the PAC learning model.

The first subexponential time algorithm for learning DNF was due to Bshouty [11] and learns any s-term DNF over n variables in time $2^{O((n \log s)^{1/2} \log^{3/2} n)}$. At the heart of Bshouty's algorithm is a structural result which shows that that any s-term DNF can be expressed as an $O((n \log n \log s)^{1/2})$-decision list; together with the aforementioned algorithm of [35] this gives the result. Subsequently Tarui and Tsukiji [38] gave a different algorithm for learning DNF with a similar runtime bound. Their algorithm adapted the machinery of "approximate inclusion/exclusion" developed by Linial and Nisan [30] in combination with hypothesis boosting [16] and learns s-term DNF in time $2^{O(n^{1/2} \log n \log s)}$.

In [26], Klivans and Servedio showed that any DNF formula with s terms can be expressed as a polynomial threshold function of degree $O(n^{1/3} \log s)$. By Fact 1 this yields an algorithm for learning s-term DNF in time $2^{O(n^{1/3} \log n \log s)}$, which is the fastest known time bound.

Several lower bounds on polynomial threshold function degree for DNFs are known which complement the $O(n^{1/3} \log s)$ upper bound of [26]. A well-known theorem of Minsky and Papert [31] shows that the "one-in-a-box" function (which is equivalent to an $n^{1/3}$-term DNF on n variables) requires polynomial threshold function degree $\Omega(n^{1/3})$. Minsky and Papert also proved that the parity function on k variables required polynomial threshold function degree at least k; since s-term DNF formulas can compute the parity function on $\log s$ variables, this gives an $\Omega(\log s)$ lower bound for s-term DNF as well. These known results motivate:

Question 1. Can we close the remaining gap between the $O(n^{1/3} \log s)$ upper bound and the $\max\{n^{1/3}, \log s\}$ lower bound on polynomial threshold function degree for s-term DNF?

Note that for decision trees no gap at all exists; Blum's approach gives a $\lfloor \log s \rfloor$ degree upper bound for size-s decision trees, and the parity function shows that this is tight.

3.3 Boolean Formulas

Known results on learning Boolean formulas of depth greater than two are quite limited. O'Donnell and Servedio [33] have shown that any unbounded fanin Boolean AND/OR/NOT formula of depth d and size (number of leaves) s is computed by a polynomial threshold function of degree $\sqrt{s}(\log s)^{O(d)}$. By Fact 1 this gives a $2^{\tilde{O}(n^{1/2+\epsilon})}$ time PAC learning algorithm for linear-size Boolean formulas of depth $o(\frac{\log n}{\log \log n})$.

It would be very interesting to weaken the dependence on either size or depth in the results of [33]:

Question 2. Does every AND/OR/NOT formula of size s have a polynomial threshold function of degree $O(\sqrt{s})$, independent of its depth?

An $O(\sqrt{s})$ degree bound would be the best possible since size-s formulas can express the parity function on $\sqrt{s}$ variables.

Question 3. Does every depth-3 AND/OR/NOT formula of size poly(n) have a polynomial threshold function of degree $o(n)$?

The strongest degree lower bound known for poly(n)-size formulas of small depth is $\Omega(n^{1/3}(\log n)^{2(d-2)/3})$ for formulas of depth $d \geq 3$ [33]. A lower bound of $\Omega(n^{2/5})$ for an explicit linear-size, depth-3 formula is conjectured in [33]. Some related results were proved by Krause and Pudlak [27], who gave an explicit depth-3 formula that requires any polynomial threshold function to have $2^{n^{\Omega(1)}}$ many monomials.

We note that there is some reason to believe that the class of arbitrary constant-depth, polynomial-size AND/OR/NOT Boolean formulas (e.g. the class of AC^0 circuits) is not PAC learnable in poly(n) time. Kharitonov [24] has shown that an $n^{(\log n)^{o(d)}}$-time algorithm for learning poly(n)-size, depth-d Boolean formulas for sufficiently large constant d would contradict a strong but plausible cryptographic assumption about the hardness of integer factorization (essentially the assumption is that factoring n-bit integers is 2^{n^ϵ}-hard in the average case for some absolute constant $\epsilon > 0$; see [24] for details).

3.4 Intersections of Halfspaces

In addition to the concept classes of Boolean formulas discussed in the previous sections, there is considerable interest in studying the learnability of various geometrically defined concept classes. As noted in Section 2.2, efficient algorithms are known which can learn a single halfspace over $\{0,1\}^n$. Algorithms for learning a single halfspace are at the heart of some of the most widely used and successful techniques in machine learning such as support vector machines [36] and boosting algorithms [16, 17]. Thus it is of great interest to obtain such algorithms for learning richer functions defined in terms of several halfspaces, such as intersections of two or more halfspaces.

A halfspace f has *weight* W if it can be expressed as $f(x) = \mathrm{sgn}(w_1 x_1 + \cdots + w_n x_n - \theta)$ where each w_i is an integer and $\sum_{i=1}^n |w_i| \leq W$. Well known results of Muroga *et al.* [32] show that any halfspace over $\{0,1\}^n$ is equivalent to some halfspace of weight $2^{O(n \log n)}$, and Håstad [37] has exhibited a halfspace which has weight $2^{\Omega(n \log n)}$. All of the current fastest algorithms for learning intersections of halfspaces have a significant runtime dependence on the weight W.

Using techniques of Beigel *et al.* [3], Klivans *et al.* [25] showed that any intersection of k halfspaces of weight W is computed by a polynomial threshold function of degree $O(k \log k \log W)$. By Fact 1, this gives a quasipolynomial-time ($n^{\mathrm{polylog}(n)}$) algorithm for learning an intersection of polylog(n) many polynomial-weight halfspaces. Since the "one-in-a-box" function on k^3 variables can be expressed as an intersection of k halfspaces each of weight $W = k^2$, we have that for $W = k^2$ there is an $\Omega(k)$ degree lower bound which nearly matches the

$O(k \log k \log w)$ upper bound. It is also shown in [25] that any intersection of k halfspaces of weight W can be expressed as a polynomial threshold function of degree $O(\sqrt{W} \log k)$; this gives a stronger bound in cases where W is small and k is large.

More generally, [25] showed that any Boolean function of k halfspaces of weight W is computed by a polynomial threshold function of degree $O(k^2 \log W)$. It follows that not just intersections, but in fact any Boolean function of polylog(n) many polynomial-weight halfspaces can be learned in quasipolynomial time.

While the above results are useful for intersections of halfspaces whose weights are not too large, in the general case they do not give a nontrivial bound even for an intersection of two halfspaces. A major open question is:

Question 4. Is there a $2^{o(n)}$ time algorithm which can PAC learn the intersection of two arbitrary halfspaces over $\{0,1\}^n$?

An affirmative answer to the above question would immediately follow from an affirmative answer to the following:

Question 5. Can every intersection of two halfspaces over $\{0,1\}^n$ be computed by a polynomial threshold function of degree $o(n)$?

The strongest known lower bound on polynomial threshold function degree for intersections of two halfspaces is quite weak; in [33] it is shown that an intersection of two majority functions (which are weight-n halfspaces) requires polynomial threshold function degree $\Omega(\frac{\log n}{\log \log n})$. Thus there is an exponential gap in our current knowledge of the answer to Question 5.

4 A different direction: linear algebraic approaches

We have seen that algorithms for learning polynomial threshold functions have broad utility in computational learning theory, yielding state-of-the-art PAC learning results for a wide range of rich concept classes. We note also that, as is well known, simple concept classes such as conjunctions, disjunctions, r-out-of-k threshold functions and decision lists can all be learned in poly(n) time using algorithms to learn linear threshold functions. Thus it is reasonable to ask at this point whether there are *any* natural concept classes over $\{0,1\}^n$ which require other techniques.

The answer is yes. The *parity* function defined by a set of variables $S \subseteq \{x_1, \ldots, x_n\}$ is the Boolean function which outputs $\sum_{x_i \in S} x_i$ mod 2. Polynomial threshold function based learning techniques are poorly suited for learning the concept class C consisting of all 2^n parity functions[1]; however, there are simple poly(n)-time learning algorithms for this class based on linear algebra [18, 15]. (Each example which is labeled according to a parity function gives

[1] Indeed, the parity function on all n variables and its negation are the only n-variable Boolean functions which require every polynomial threshold function representation to have degree as large as n [2].

a linear equation mod 2, and the system of linear equations obtained from a labeled sample can be solved efficiently to obtain a consistent parity hypothesis. Standard arguments [8] can be used to show that any parity function hypothesis which is consistent with a sufficiently large sample is probably approximately correct.)

As was the case with linear threshold learning algorithms, it is possible to run algorithms for learning parity functions over an expanded feature space of all degree-d monomials. Since multiplication corresponds to AND over GF_2 and addition corresponds to parity, we have the following analogue of Fact 1:

Fact 2 *Let C be a class of functions each of which can be expressed as an degree-d polynomial over GF_2. Then there is a poly(N)-time PAC learning algorithm for C, where $N = \sum_{i=1}^{d} \binom{n}{i} \leq (\frac{en}{d})^d$.*

Can Fact 2 can be used, either by itself or in conjunction with other techniques, to obtain interesting algorithms for learning rich Boolean function classes? One such result has been achieved by Bshouty *et al.* [13]. They showed that the class of *strict width two branching programs* (branching programs of width two with exactly two sinks) are PAC learnable in polynomial time, using an algorithm which combines parity learning with a decision list learning technique of Rivest [35]. The algorithm of [13] is of special interest because it provides an example where general linear threshold function learning algorithms do *not* supplant algorithms designed for restricted subclasses of linear threshold functions (in this case decision lists); while linear threshold function learning algorithms can learn decision lists, they cannot be combined with the parity learning component as required to obtain the results of [13]. Inspection shows that in fact it is possible to combine the more powerful approach of Ehrenfeucht and Haussler [14] for learning decision trees (a richer class of functions than decision lists) with parity (or more generally, GF_2 polynomial) learning algorithms in a similar way to [13]. Exploring the power of such an approach is an interesting direction for future work.

5 Acknowledgement

We thank Adam Klivans for helpful suggestions in preparing this survey.

References

[1] D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[2] J. Aspnes, R. Beigel, M. Furst, and S. Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):1–14, 1994.

[3] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. *Journal of Computer and System Sciences*, 50(2):191–202, 1995.

[4] A. Beimel, F. Bergadano, N. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.

[5] S. Ben-David and E. Dichterman. Learning with restricted focus of attention. *Journal of Computer and System Sciences*, 56(3):277–298, 1998.

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

[7] A. Blum, P. Chalasani, and J. Jackson. On learning embedded symmetric concepts. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 337–346, 1993.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[10] N. Bshouty. Exact learning via the monotone theory. *Information and Computation*, 123(1):146–153, 1995.

[11] N. Bshouty. A subexponential exact learning algorithm for DNF using equivalence queries. *Information Processing Letters*, 59:37–39, 1996.

[12] N. Bshouty and C. Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770, 1996.

[13] N. Bshouty, C. Tamon, and D. Wilson. On learning width two branching programs. *Information Processing Letters*, 65:217–222, 1998.

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

[15] P. Fischer and H.U. Simon. On learning ring-sum expansions. *SIAM Journal on Computing*, 21(1):181–192, 1992.

[16] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[17] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

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

[19] J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55:414–440, 1997.

[20] J. Jackson, A. Klivans, and R. Servedio. Learnability beyond AC^0. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 776–784, 2002.

[21] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998.

[22] M. Kearns and U. Vazirani. *An introduction to computational learning theory.* MIT Press, Cambridge, MA, 1994.

[23] L. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Dokl*, 20:1093–1096, 1979.

[24] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the Twenty-Fifth Annual Symposium on Theory of Computing*, pages 372–381, 1993.

[25] A. Klivans, R. O'Donnell, and R. Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer & System Sciences*, 68(4):808–840, 2004. Preliminary version in *Proc. of FOCS'02*.

[26] A. Klivans and R. Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. *Journal of Computer & System Sciences*, 68(2):303–318, 2004. Preliminary version in *Proc. STOC'01.*

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

[28] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM J. on Computing*, 22(6):1331–1348, 1993.

[29] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[30] N. Linial and N. Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365, 1990.

[31] M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, Cambridge, MA, 1968.

[32] S. Muroga, I. Toda, and S. Takasu. Theory of majority switching elements. *J. Franklin Institute*, 271:376–418, 1961.

[33] R. O'Donnell and R. Servedio. New degree bounds for polynomial threshold functions. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 325–334, 2003.

[34] R. O'Donnell and R. Servedio. Learning monotone decision trees in polynomial time. Submitted for publication, 2005.

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

[36] J. Shawe-Taylor and N. Cristianini. *An introduction to support vector machines*. Cambridge University Press, 2000.

[37] J. Håstad. On the size of weights for threshold gates. *SIAM Journal on Discrete Mathematics*, 7(3):484–492, 1994.

[38] J. Tarui and T. Tsukiji. Learning DNF by approximating inclusion-exclusion formulae. In *Proceedings of the Fourteenth Conference on Computational Complexity*, pages 215–220, 1999.

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

[40] K. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 314–326, 1990.