ELSEVIER

Theoretical Computer Science 137 (1995) 159–176

Theoretical Computer Science

Exact learning of linear combinations of monotone terms from function value queries

Atsuyoshi Nakamura*, Naoki Abe

Theory NEC Laboratory, Real World Computing Partnership, c/o C & C Research Laboratories, NEC Corporation, 4-1-1 Miyazaki Miyamae-ku, Kawasaki 216, Japan

Abstract

We investigate the problem of exactly identifying a real-valued function of $\{0,1\}^n$ represented by a weighted sum of a number of monotone terms by querying for the values of the target function at assignments of the learner's choice. When all coefficients are nonnegative, we exhibit an efficient learning algorithm requiring at most $(n-\lfloor \log s \rfloor+1)s$ queries, where n is the number of variables and s is the number of terms in the target formula. We prove a lower bound of $\Omega(ns/\log s)$ on the number of queries necessary for learning this class, so no algorithm can reduce the number of queries dramatically. The algorithm runs in time $O(ns^2)$ in the worst case. The same algorithm can be used to learn the 'inductive-read-k' subclass, a proper superclass of the 'read-k' subclass, with a number of queries not exceeding $\frac{1}{2}((n-\lfloor \log k \rfloor)(n-\lfloor \log k \rfloor+1)+2)k$, which improves upon the bound achievable by a naive learning algorithm by a factor of two. In addition, the above method can be extended to handle the nonmonotone case in some restricted sense: A similar algorithm can learn the *unate* linear combinations of terms with a comparable number of queries. In the general case, namely, when the coefficients vary over the reals (or any arbitrary field), we show that the number of queries required for exact learning of the k-term subclass is upper bounded by $q(n,\lfloor \log k \rfloor+1)$ and is lower bounded by $q(n,\lfloor \log k \rfloor)$, where $q(n,l)=\sum_{i=0}^{l}\binom{n}{i}$. These bounds are shown by generalizing Roth and Benedek's technique for analyzing the learning problem for k-sparse multivariate polynomials over GF(2) (Roth and Benedek, 1991) to those over an arbitrary field.

1. Introduction

We consider the learning problem for functions that are representable as a weighted sum of a number of *monotone terms*, namely a linear combination of products of Boolean variables. The learning model we study in this paper is exact learning (identification) via *function value queries*, namely queries that ask for the value of the

* Corresponding author. E-mail: atsu@sbl.cl.nec.co.jp and abe@sbl.cl.nec.co.jp.

SSDI 0304-3975(94)00166-9

target function at an assignment. The function value query coincides with the *membership query* when the target function is a Boolean function.

Our work was motivated by the question of whether DNF is learnable if the number of satisfied terms is given as a response to a membership query, in addition to the value of the DNF formula. Note that this corresponds to the case in which the target function is the arithmetic sum of the terms. As one can obtain more information this way, one suspects that the amount of computation and the number of queries necessary for learning DNF from this type of queries may be significantly reduced. The motivation for considering the weighted sum of monotone terms, which is more general than the sum of monotone terms, comes from the idea that it is natural for each term to have a different weight according to its importance. Note that linear combinations of monotone terms are very expressive because all real-valued functions on $\{0,1\}^n$ are representable in this form.

In this paper, we obtain a number of positive learnability results for this and related classes of functions. If all coefficients are restricted to be nonnegative, then the linear combinations of monotone terms can be learned exactly by an efficient learning algorithm, using a number of function value queries bounded above by $(n-\lfloor \log s \rfloor+1)s$, where n is the number of variables and s is the number of terms (with nonzero coefficients) in the target function. This bound is an improvement over an upper bound of $ns+1$ achievable by a simpler, 'naive' learning algorithm. We also obtain a lower bound of order $\Omega(ns/\log s)$ on the number of queries needed to learn this class, so no algorithm can reduce the number of queries dramatically. We also show that the number of queries made by the same learning algorithm when it is used to learn the 'inductive-read-k' subclass – a proper superclass of the 'read-k' subclass – is bounded above by $\frac{1}{2}((n-\lfloor \log k \rfloor)(n-\lfloor \log k \rfloor+1)+2)k$. It cuts down the worst-case number of queries made by the aforementioned naive algorithm for the same class, $n(n-\lfloor \log k \rfloor+1)k+1$, roughly by a half. This learning algorithm is also computationally efficient: Its worst-case running time is $O(ns^2)$, where s is the number of terms in the target.

The methodology used to obtain the above results can actually be extended to handle the nonmonotone case in some restricted sense: A similar algorithm can efficiently learn the *unate* (i.e., each variable can appear either negated or nonnegated but not both) linear combinations of terms, with the addition of *equivalence queries*. The learning algorithm makes at most $s+1$ equivalence queries and $(n+1)s$ function value queries, and runs in time $O(ns^2)$, where s is the number of terms in the target function.

In the general case, namely for the class of linear combinations of monotone terms with coefficients varying over the reals, the same upper and lower bounds on the number of required queries that are known to hold for learning multivariate polynomials over GF(2) hold: It is upper bounded by $q(n,\lfloor \log k \rfloor+1)$ and is lower bounded by $q(n,\lfloor \log k \rfloor)$, where $q(n,l)=\sum_{i=0}^{l}\binom{n}{i}$. These bounds are obtained by generalizing Roth and Benedek's technique they developed to analyze the learning

problem for multivariate polynomials over GF(2). All of the results for the 'general case' are actually shown to hold for coefficients belonging to an arbitrary field.

As we noted earlier, the problem of exactly learning a function defined as an operator applied on a set of (monotone) terms can be viewed as the problem of identifying the set of terms using that operator as the information source. This is similar in spirit to the PAC learning model with additional information proposed by Kahihara and Imai [7]. Together with previously known results due to various authors on the complexity of learning monotone DNF and multivariate polynomials over GF(2), the results in this paper imply that the summation $(+)$ is significantly more valuable than the logical OR $(\vee)$ or the exclusive OR $(\oplus)$, as a source of additional information. For example, for identifying a set of k monotone terms, the required number of queries for the three operators progress roughly as $O(n^k)$ for $\vee$, $O(n^{\log k})$ for $\oplus$, and $O(nk)$ for $+$. Note that the number of queries polynomial in both n and k is only for $+$. For identifying a read-k set of monotone terms, the number of required queries for $\vee$, $\oplus$ and $+$ seems to decrease roughly as 'exponential in n', $O(n^{\log k})$, and $O(n^2k)$. These comparisons are described in detail in Section 5.

2. Preliminaries

Let A_n denote the set of Boolean variables $x_1, x_2, \ldots, x_n$. A *literal* is either a variable x or its negation $\bar{x}$, and a *term* is a product of literals. A term is said to be *monotone* if it contains no negated variable. Since the product of variables belonging to the empty set is 1, 1 is a monotone term. We let MT denote the set of all monotone terms. For each term $t \in MT$, $v(t)$ denotes the set of variables in t. Let K be an arbitrary field. We let $\mathscr{F}(K)$ denote the set of *linear combinations of monotone terms* over K, namely: $\mathscr{F}(K) = \{\sum_{t \in MT} a_t \cdot t : a_t \in K\}$, where $\sum$ and $\cdot$ are the addition and multiplication of K and the value of $t \in MT$ is 0 or 1, the identity element of addition or multiplication of K, respectively. We abbreviate $\mathscr{F}(K)$ by $\mathscr{F}$, whenever it is clear from context. Throughout Section 3, we assume that $K = \mathbb{R}^+$ (in this case K is not a field). All of the results in Section 4 hold for an arbitrary field K, but most of the informal discussion assumes that $K = \mathbb{R}$. Also note that we abuse notation and let $\mathscr{F}$ denote, ambiguously, both the class of *functions* represented by linear combinations, and the class of *representations* themselves. We do the same for elements F of $\mathscr{F}$. For each $F \in \mathscr{F}$, $T(F)$ denotes the set of terms with nonzero coefficients in F, and for any $S \subseteq A_n$, $T_S(F)$ denotes the set of terms in $T(F)$ containing variables in S only. A function $F \in \mathscr{F}$ is said to be *k-term* if $|T(F)| \leqslant k$. A function $F \in \mathscr{F}$ is *read-k* if each variable is contained in at most k terms of $T(F)$. We say that a function $F \in \mathscr{F}$ is *inductive-read-k* if, for each variable, there are at most k terms that contain it as the variable with the greatest index, namely, for each $m \in \{1, 2, \ldots, n\}$, there are at most k terms of $T(F)$ which contain x_m and no variables in $\{x_{m+1}, x_{m+2}, \ldots, x_n\}$. We let *$k$-term-$\mathscr{F}$*, *read-$k$-$\mathscr{F}$* and *inductive-read-$k$-$\mathscr{F}$* denote the k-term, read-k and inductive-read-k subclasses of $\mathscr{F}$, respectively.

We also consider functions which do not belong to the class $\mathscr{F}$. A function F is a *unate linear combination of terms* if F is a linear combination of terms such that, for no variable x, both x and $\bar{x}$ are present. Note that, by definition, a linear combination of monotone terms is unate.

We let $Q(n,k)$ denote the class of subsets of A_n of cardinality at most k. We use the notation 1_S to denote the assignment in which all variables belonging to S are set to 1 and all others to 0. We also use 0_S which is similarly defined. We abbreviate 1_{A_n} and 0_{A_n} by 1 and 0, respectively. When c is an assignment, c_i denotes the value assigned to x_i by c, and $c_{\overline{x_i}}$ is the assignment such that $(c_{\overline{x_i}})_j = c_j$ for $j \neq i$ and $(c_{\overline{x_i}})_i \neq c_i$. Also, for any variable x, we sometimes let x^1 and x^0 stand for the literals x and $\bar{x}$, respectively, by convention. All logarithms in this paper are to the base 2.

3. The case with nonnegative coefficients

3.1. *An efficient learning algorithm*

In this section, we present our positive learnability results for linear combinations of monotone terms with nonnegative coefficients via function value queries.

In this case, there is a simple learning algorithm that does fairly well. The algorithm is based on a method for learning monotone DNF [1, 5]. In particular, it uses the same subroutine to find one monotone term with a nonzero coefficient as that used in [1, 5], and it works as follows. Let $S = A_n$ initially. If $F(1_{A_n}) = 0$ then $F \equiv 0$, so suppose $F(1_S) > 0$. This means that there is a term t with a nonzero coefficient such that $v(t) \subseteq S$. For every variable $x \in S$, ask for the value of $F(1_{S-\{x\}})$ and if $F(1_{S-\{x\}}) > 0$ then remove x from S. After this process is completed, we must have $F(1_S) > 0$ and $F(1_{S-\{x\}}) = 0$ for all $x \in S$. This means that F contains the term $\prod_{x \in S} x$ with coefficient $F(1_S)$. Provided that $F(1_{A_n}) > 0$, we can find a term t with a nonzero coefficient a by the process described above, using exactly n function value queries. We can then proceed to find another term with a nonzero coefficient by applying the same procedure to the function $F - at$. By repeating this procedure as many times as there are terms with a nonzero coefficient, we can identify F. The number of function value queries used in this algorithm is $ns + 1$, where s is the number of terms with a nonzero coefficient. But this algorithm does not fully use the information available to it, because within the case $F(1_{S-\{x\}}) > 0$, there are two subcases $F(1_{S-\{x\}}) = F(1_S)$ and $F(1_{S-\{x\}}) < F(1_S)$, and the algorithm does exactly the same thing in both cases. If $F(1_{S-\{x\}}) = F(1_S)$ then it means that every term t satisfying $v(t) \subseteq S$ fails to contain x, and if $F(1_{S-\{x\}}) < F(1_S)$ then it means that there are terms t_1, t_2 satisfying $v(t_1), v(t_2) \subseteq S$ such that $x \notin v(t_1)$ and $x \in v(t_2)$. Procedure Identify (Fig. 1) takes advantage of this information and recursively branches its search for terms into two parts in the latter case. In the first stage, it finds every term t_1 satisfying $v(t_1) \subseteq S - \{x\}$ and in the second, it finds every term t_2 satisfying $x \in v(t_2) \subseteq S$ making use of the information obtained in the first stage, i.e., the algorithm uses the value of $F - H'$ instead of $F - H$, where $H' - H$ is the subformula of

```
procedure Identify return(H)
H: a linear combination of monotone terms with nonnegative coefficients
begin
  a := F(1)
  if a=0 then return(0) else return(Subidentify(A_n, n, a, 0))
end
end procedure
procedure Subidentify(S, i, a, H) return(H_out)
S: a subset of A_n
i: an element of {1, 2, ..., n}
a: a positive real number
H, H_out: linear combinations of monotone terms with nonnegative coefficients
begin
  while i>0
    b := F(1_{S-{x_i}}) - H(1_{S-{x_i}})
    if b=a then S := S-{x_i}
    else if 0<b then exit while
    i := i-1
  end while
  if i=0 then return(H + a∏_{x∈S} x)
  else begin
    H' := Subidentify(S-{x_i}, i-1, b, H)
    return(Subidentify(S, i-1, a-b, H'))
  end else
end
end procedure
```

Fig. 1. Procedure Identify.

F found in the first stage. The number of function value queries used in procedure identify varies from $n-\lceil \log s \rceil + 2^{\lceil \log s \rceil}$ to $(n-\lceil \log s \rceil)s + 2^{\lceil \log s \rceil}$ depending on the exact form of F. Note that by recursively branching its search for terms, 'Identify' avoids repeatedly applying the subprocedure to find a single term as many times as there are terms, and consequently makes less queries than the naive algorithm.

Theorem 3.1. *If F is a linear combination of monotone terms with nonnegative coefficients, then procedure 'Identify' learns F exactly using at most $(n-\lfloor \log s \rfloor + 1)s$ function value queries in time $O(ns^2)$, where s is the number of terms with nonzero coefficients in F.*

Proof. Let S be a subset of A_n and H be a subformula of F, i.e., the value of every nonzero coefficient in H is the same as that of F. We define $T_S(F-H)$ to be the set of

terms in $F-H$ that contain variables in S only, namely:

$$T_S(F-H)=\{t: v(t)\subseteq S,\ t\in T(F-H)\}.$$

We first show that if Claim 3.1 stated below holds, then the output H of Identify must equal F. If $F=0$, procedure Subidentify is never called because $a=F(1)=0$, and $H=0$ is output. If $F\neq 0$, Subidentify is called with the input parameter quadruple $(A_n, n, a, 0)$, which does satisfy Relation 1, so it follows from Claim 3.1 that $0+\sum_{t\in T_{An}(F-0)} a_t t=\sum_{t\in T(F)} a_t t=F$ is output.

Claim 3.1. *Suppose that* $|T_S(F-H)|\geqslant 1$ *and a quadruple* (S, i, a, H) *satisfies Relation* 1. *Then, Subidentify*(S, i, a, H) *outputs* $H+\sum_{t\in T_S(F-H)} a_t t$.

Relation 1. *A quadruple* (S, i, a, H) *is said to satisfy Relation* 1 *if the following two conditions are satisfied.*

1. $F(1_S)-H(1_S)=a$,
2. $\forall t\in T_S(F-H), S\cap\{x_j: j>i\}=v(t)\cap\{x_j: j>i\}$.

Proof of claim 3.1. Note that after the while-loop in which S and i are updated, the quadruple (S, i, a, H) still satisfies Relation 1, because, for every $j>i$ (where i stands for its value after the while-loop), if variable x_j was removed from S in the while-loop, then x_j is not contained in any of the terms of $T_S(F-H)$ (when $b=a$), and if x_j was not removed in the while-loop, then x_j is contained in all the terms of $T_S(F-H)$ (when $b=0$).

First, we prove this claim for the case $|T_S(F-H)|=1$. Let t_0 be the only term contained in $T_S(F-H)$. In this case, $i=0$ holds after the while-loop, because $0<b<a$ is never satisfied in the while-loop. When $i=0$, Relation 1 is satisfied if and only if $a_{t_0}=a, v(t_0)=S$. So, $H+a\prod_{x\in S} x=H+a_{t_0}\prod_{x\in v(t_0)} x=H+\sum_{t\in T_S(F-H)} a_t t$ is output.

Now assume that Claim 3.1 holds when $|T_S(F-H)|<l (l\geqslant 2)$. We will show that it also holds when $|T_S(F-H)|=l$. Assume the input quadruple (S, i, a, H) satisfies Relation 1 and $|T_S(F-H)|=l$. Since $T_S(F-H)$ contains more than two terms, there exists a j such that both terms having and not having the variable x_j exist. For such j, $0<b=F(1_{S-\{x_j\}})-H(1_{S-\{x_j\}})<a$ holds. In this case, $i\neq 0$ holds after the while-loop, and hence the two recursive calls to Subidentify are made. The input quadruple $(S-\{x_i\},\ i-1, b, H)$ of the first recursive call satisfies Relation 1 because $b=F(1_{S-\{x_i\}})-H(1_{S-\{x_i\}})$ and x_i is not in $v(t)$ for all $t\in T_{S-\{x_i\}}(F-H)$. Since $|T_{S-\{x_i\}}(F-H)|<l$ holds also, by the inductive hypothesis, $H+\sum_{t\in T_{S-\{x_i\}}(F-H)} a_t t$ is output and set to H'. Therefore,

$$F(1_S)-H'(1_S)=F(1_S)-H(1_S)-\sum_{t\in T_{S-\{x_i\}}(F-H)} a_t t=a-b$$

holds. Note that x_i is contained in every term in $T_S(F-H')$, because $T_S(F-H')=T_S(F-H)-T_{S-\{x_i\}}(F-H)$. Thus, Relation 1 is satisfied by the input

quadruple $(S, i-1, a-b, H')$ of the second recursive call, and $|T_S(F-H')|<l$ holds, so by the inductive hypothesis, $H'+\sum_{t\in T_S(F-H')} a_t t = H+\sum_{t\in T_S(F-H)} a_t t$ is output. This completes the proof of Claim 3.1. □

Proof of Theorem 3.1 (*continued*). We next calculate the number of function value queries made by this algorithm. Procedure identify itself makes only one query (for the assignment 1). The number of queries made by subidentify equals the number of times b is calculated, which occurs once in each iteration of the while-loop, and thus essentially for each update of i. If we form a history of updates on i in subidentify by branching every time two recursive calls are made, and growing a single branch by one step every time i is updated, then we obtain a tree of depth n and width s. Hence, the number of queries, which is equal to the number of nodes of this tree, is at most ns. More precisely, since at the root the width is one and it takes at least $\lfloor \log s \rfloor$ steps for the width to reach s in the fastest (worst) case, the bound can be improved to $(n-\lceil \log s \rceil)s+2^{\lceil \log s \rceil} \leqslant (n-\lfloor \log s \rfloor+1)s$, which is tighter especially when s is exponentially large in n.

The running time of this algorithm is largely determined by the time it takes to compute the value b. To calculate b, $H(1_{S-\{x_i\}})$ has to be calculated, that is, for each term $t\in T(H)$, it must be determined whether t is satisfied by $1_{S-\{x_i\}}$ or not. This can be done by finding out whether t is satisfied by 1_S and whether $x_i\in v(t)$, which can be done in a constant number of steps by keeping the information of whether t is satisfied by 1_S and x_i's position in the sorted variable list of t from the previous iteration. Therefore, b can be calculated in $O(|T(H)|)$ time, and thus the algorithm runs in time $O(ns^2)$. □

3.2. *Learning the inductive-read-k subclass*

For learning inductive-read-k linear combinations of monotone terms with non-negative coefficients, the naive algorithm described at the beginning of the previous subsection needs $n(n-\lfloor \log k \rfloor+1)k+1$ function value queries in the worst case because an inductive read-k formula can have at most $(n-\lfloor \log k \rfloor+1)k$ terms. Algorithm identify improves upon this bound and reduces the coefficient on n^2 from k to $\frac{1}{2}k$. In the case of read-once, we can further reduce the required number of queries: the coefficient on n^2 can be made $\frac{1}{4}$ by modifying the algorithm so as not to query $F(1_{S-\{x_i\}})$ if a term $t\in T(F)$ containing the variable x_i has already been found.

Theorem 3.2. *If F is an inductive-read-k linear combination of monotone terms with non-negative coefficients, then algorithm 'Identify' exactly learns F using at most $\frac{1}{2}((n-\lfloor \log k \rfloor)(n-\lfloor \log k \rfloor+1)+2)k$ function value queries. In particular, when F is read-once, 'Identify' with the aforementioned modification learns F with at most $\frac{1}{4}n^2+n+1$ queries.*

Proof. Suppose identify is used to learn an inductive-read-k linear combination F. Suppose that we form a tree of recursive calls to subidentify, and partition this tree

into a number of subtrees, each of which is rooted by the first occurrence of the second call (of the two recursive calls to subidentify within subidentify). Then the terms found in each subtree has the same variable as the variable of the greatest index, and hence at most k terms are found in each subtree. Since the number of *effective* variables decreases by one as i increases by one, and when the number of effective variables j becomes $\lfloor \log k \rfloor$, there are at most $2^j \leqslant k$ terms found, it follows that the number of function value queries made by Identify is at most $1+\sum_{i=1}^{n-\lfloor \log k \rfloor - 1}(n-i-\lfloor \log k \rfloor+1)k+\sum_{i=1}^{\lfloor \log k \rfloor+1} 2^{i-1} \leqslant \frac{1}{2}((n-\lfloor \log k \rfloor)(n-\lfloor \log k \rfloor+1)+2)k$.

Let F be read-once and $l=|T(F)|$. Then it follows easily from a similar argument as for the inductive-read-k case that Identify uses at most $1+\sum_{i=0}^{l-1}(n-i)$ function value queries. If the algorithm is modified so as not to query $F(1_{S-\{x_i\}})$ when a term $t \in T(F)$ containing the variable x_i has already been found, then the number of function value queries decreases by at least $\sum_{i=1}^{l-2} i$. Hence, the modified algorithm needs at most $1+\sum_{i=0}^{l-1}(n-i)-\sum_{i=1}^{l-2} i \leqslant -l^2+(n+2)l$ function value queries, which is at most $\frac{1}{4}n^2+n+1$ for any value of l, $1 \leqslant l \leqslant n$. □

3.3. A lower bound

We next give a lower bound on the number of function value queries needed for exact learning of linear combinations of monotone terms with nonnegative coefficients.

Theorem 3.3. *Any exact learning algorithm for k-term linear combinations of monotone terms with nonnegative coefficients must make more than $(nk-1)/\log(k^2+1)=\Omega(nk/\log k)$ function value queries, in the worst case.*

Proof. Consider the following subset of k-term linear combinations of monotone terms with nonnegative coefficients:

$$\mathscr{G}=\{F \in \mathscr{F}: |T(F)|=k \text{ and } \forall t \in T(F),\ a_t \in \{1,2,\ldots,k\}\}.$$

The value of $F \in \mathscr{G}$ for an arbitrary assignment is one of the k^2+1 values, $0,1,\ldots,k^2$. Thus, for any new assignment, one of these values is assumed by at least $1/(k^2+1)$ of the functions in $\mathscr{G}$ that are consistent with the answers for the queries made so far. If an adversary selects such a value as the response to the current query, at least $1/(k^2+1)$ of the consistent functions remain. When any algorithm identifies the target function, there must not be two distinct functions that are consistent with the answers to the queries made up to that point. Therefore, the number of queries l necessary in this case must satisfy

$$\left(\frac{1}{k^2+1}\right)^l |\mathscr{G}| < 2. \tag{1}$$

Now, $|\mathscr{G}|$ is bounded from below by the product of the number of combinations of k terms over n variables $\binom{n}{k}$, and the number of coefficient assignments to a fixed set of

k terms, i.e., k^k. Since $\binom{2^n}{k} \geqslant (2^n/k)^k$ holds, we get $|\mathscr{G}| \geqslant 2^{nk}$. Therefore, by (1), we must have

$$l > \frac{nk-1}{\log(k^2+1)}. \quad \square$$

3.4. *Extension to learning unate linear combinations*

In this section, we diverge from the topic of learning linear combinations of *monotone* terms, and turn to the learning problem for *unate* linear combinations of terms, not necessarily monotone. Recall that a formula over the Boolean domain is said to be *unate*, if for no variable x, both x and $\bar{x}$ appear in it. It turns out that the unate linear combinations of terms with nonnegative coefficients are learnable by a similar method as that for the monotone case, but equivalence queries are required in this case. Consider, for example, the class of terms, $\mathscr{G} = \{x_1^{c_1} x_2^{c_2} \ldots x_n^{c_n} : c \in \{0,1\}^n\}$ where we use x^1 and x^0 to denote the literals x and $\bar{x}$, respectively (cf. Section 2). $\mathscr{G}$ certainly is a subclass of unate linear combinations of terms with nonnegative coefficients. Note that for every function in $\mathscr{G}$, there is exactly one assignment at which its value is not 0. Thus, if we use function value queries only, we need 2^n queries to identify a target function in $\mathscr{G} \cup \{0\}$. If we allow equivalence queries also, however, there is a simple learning algorithm for unate linear combinations of terms with nonnegative coefficients, which is similar to the naive algorithm of the previous section. We will describe this 'naive' algorithm first and then later present a more sophisticated one. Suppose we have an assignment c at which $F(c) > 0$. Then there must be a term t with a nonzero coefficient, which is the product of some subset of $\{x_1^{c_1}, x_2^{c_2}, \ldots, x_n^{c_n}\}$. Initialize $S = \emptyset$, and for $i = 1$ to n, repeat the following procedure:

1. ask for the value of $F(c_{\overline{x_i}})$,
2. if $F(c_{\overline{x_i}}) = 0$, then add i to S,
3. if $0 < F(c_{\overline{x_i}}) < F(c)$, then substitute $c_{\overline{x_i}}$ for c.

When this is completed, we will have found the term $\prod_{i \in S} x_i^{c_i}$ with coefficient $F(c)$. To find assignments at which F is positive, like c in the above argument, we need equivalence queries. Having found a subformula H of F, we can obtain an assignment c at which $F(c) - H(c) > 0$, by an equivalence query with hypothesis H, and a term in $F - H$ is found by carrying out the above procedure for $F - H$. By this method, the class of unate linear combinations of terms with nonnegative coefficients is exactly learnable using a number of queries bounded above by the bound given in the following theorem.

Theorem 3.4. *The class of unate linear combinations of terms with nonnegative coefficients is exactly learnable using at most $s+1$ equivalence queries and $(n+1)s$ function value queries in time $O(ns^2)$, where s is the number of terms with nonzero coefficients in the target function.*

```
procedure Unate-identify return(H)
H: a unate linear combination of terms with nonnegative coefficients
begin
  H := 0
  repeat
    c := EQUIV(H)
    if c = 'Yes' then return(H)
    H := Unate-subidentify(∅, n, F(c) − H(c), H, c)
  end repeat
end
end procedure
procedure Unate-subidentify(S, i, a, H, c) return(H_out)
S: a subset of {1, 2, ..., n}
i: an element of {1, 2, ..., n}
a: a positive real number
H, H_out: unate linear combinations of terms with nonnegative coefficients
c: an assignment to the variables x_1, ..., x_n
begin
  while i > 0
    b := F(c_{x̄_i}) − H(c_{x̄_i})
    if b = 0 then S := S ∪ {i}
    else if b < a then exit while
    i := i − 1
  end while
  if i = 0 then return(H + a ∏_{i∈S} x_i^{c_i})
  else begin
    H' := Unate-subidentify(S, i − 1, b, H, c_{x̄_i})
    return(Unate-subidentify(S ∪ {i}, i − 1, a − b, H', c))
  end else
end
end procedure
```

Fig. 2. Procedure Unate-identify.

Proof. It is easy to see that the 'naive' learning algorithm described above witnesses the theorem. □

In Fig. 2, we exhibit an improved version of the naive learning algorithm described above. This algorithm is analogous to 'Identify' from the previous subsection (hence the name 'Unate-identify'). Here is how this algorithm works. Unate-identify hypothesizes the current hypothesis H, which is guaranteed to be a subformula of the target

function F, and then obtains an assignment c at which the target function minus the current hypothesis $F-H$ assumes a positive value. This implies that there are some terms in $F-H$ that are satisfied by c. Unate-identify then calls 'Unate-subidentify', which finds all the terms in $F-H$ that are satisfied by c. Unate-identify repeats this process until all the terms are found and 'yes' is returned from the equivalence query.

Unate-subidentify is analogous to 'subidentify' from Section 3.3 in the way it recursively divides its search into subcases and finds all the terms that can be found starting from a given assignment c. It divides its search into two branches when $0<b<a$, since it means that there are, still to be found, both terms including $x_i^{c_i}$ and not including either x_i or $\overline{x_i}$. It is also possible to have $b>a$ unlike the monotone case, which means that there is a term that is not satisfied by c, but becomes satisfied by flipping the ith bit of c, namely it is satisfied by $c_{\overline{x_i}}$. Since the target function is *unate* this means that no term can include $x_i^{c_i}$, and hence the algorithm flips back the bit and continues its search. Note that it ignores the term that was newly turned on by the flip as this term was not satisfied by c. Since Unate-identify finds all the terms that are satisfied by each assignment given back by the equivalence query, it requires a significantly smaller number of queries than the naive algorithm in most cases, although in the worst case it can make as many queries.

4. The general case

In Section 3, we considered exact learnability of the subclass of $\mathscr{F}$ with nonnegative coefficients and showed that the number of function value queries needed is polynomial in the number of variables and the number of nonzero coefficient terms. This is impossible, however, when we allow real (and other more general) coefficients. For the class $\mathscr{F}$, i.e., when the coefficients are values in an arbitrary field, we show polynomial time exact learnability of two subclasses, k-term-$\mathscr{F}$ and inductive-read-k-$\mathscr{F}$, by generalizing the method Clausen et al. [4], Hellerstein and Warmuth [6] and Roth and Benedek [9] developed for multivariate polynomials over GF(2), and obtain comparable bounds on the number of queries needed to identify them.

4.1. Learning the k-term subclass

We first present the upper and lower bounds we obtain for the k-term linear combinations of monotone terms.

Theorem 4.1. *Let k-term-$\mathscr{F}$ be as defined in Section 2.*

1. *There exists an algorithm that exactly learns k-term-$\mathscr{F}$, making function value queries at assignments 0_S for all $S\in Q(n,\lfloor \log k \rfloor+1)$, running in time $O(kn|Q(n,\lfloor \log k \rfloor+1)|)$.*

2. *Any exact learning algorithm for k-term-$\mathscr{F}$ must make more than $|Q(n,\lfloor \log k \rfloor)|$ function value queries, in the worst case.*

Proof (*part* 1). We will prove a series of lemmas (Lemmas 4.1–4.3) from which Theorem 4.1, part 1 follows easily. Some preliminary definitions are in order.

Let K be an arbitrary field with identity elements 0, 1 of addition and multiplication, respectively. We generalize Roth and Benedek's theory developed for the vector space $\mathrm{GF}(2)^m$ over $\mathrm{GF}(2)$ to the vector space K^m over any field K.

We define a total order $\leqslant$ on the set 2^{A_n} by the 'inverse' lexical ordering, viewing the subsets S of A_n as bit vectors 0_S, namely:

$$S_1 \leqslant S_2 \Leftrightarrow \sum_{x_i \in S_1} 2^i \leqslant \sum_{x_i \in S_2} 2^i.$$

We next define a $|Q(n,l)| \times 2^n$ matrix $H_{n,l}$ composed only of 0 and 1, the additive and multiplicative identities of K. We use members of $Q(n,l)$ and 2^{A_n} to index the rows and columns of $H_{n,l}$, respectively. Both the rows and columns of $H_{n,l}$ are arranged in the order $\leqslant$ defined above. Now, the (S_1, S_2)-component of $H_{n,l}$ is defined as follows

$$H_{n,l}[S_1, S_2] = \begin{cases} 1 & \text{if the assignment } 0_{S_1} \text{ satisfies the term } \prod_{x \in S_2} x \\ 0 & \text{otherwise.} \end{cases}$$

For $F = \sum_{t \in MT} a_t \cdot t$ (where $a_t \in K$ and $\sum$ and $\cdot$ are the addition and multiplication operations of K), consider the 2^n dimensional column vector f whose S-component $f[S]$ is a_t such that $v(t) = S$. Then, the $|Q(n,l)|$ dimensional column vector g defined as $g = H_{n,l} f$ is the vector whose S-component $g[S]$ is $F(0_S)$, for any $S \in Q(n,l)$.

For $f \in K^m$, $|f|$ denotes the number of nonzero components in f. We define in general K_l^m as follows:

$$K_l^m = \{ f \in K^m : |f| \leqslant l \}.$$

The first two lemmas (Lemmas 4.1 and 4.2) are analogues of lemmas proved by Clausen et al. [4] for the multivariate polynomials over $\mathrm{GF}(q)$, and establish that $H_{n,l} f$ contains enough information to completely specify $f \in K_{2^l-1}^{2^n}$.

Clausen et al. proved their analogue of the following lemma as a corollary of a stronger result, and Hellerstein and Warmuth [6] presented a simpler proof for it (also for $\mathrm{GF}(2)$), which we adopt in the proof of our generalized version below.

Lemma 4.1. *Let n, l be nonnegative integers. If $f \in K_{2^{l+1}-1}^{2^n}$, then $f = 0$ if only if $H_{n,l} f = 0$.*

Proof. The 'only if' part is trivial, so we prove the 'if' part only. We prove this by induction on n and l. When $l = 0$, $f \in K_1^{2^n}$ and $H_{n,0} = (1, \ldots, 1)$, since $Q(n,0) = \{\emptyset\}$ and $H[\emptyset, S] = 1$ for any $S \in 2^{A_n}$. If $f \neq 0$, then there exists exactly one $S \in 2^{A_n}$ such that $f[S] \neq 0$. In this case, $H_{n,0} f = f[S] \neq 0$. The proof is trivial when $n = 0$, because $H_{0,l} = (1)$.

Let $n \geqslant 1$ and $l \geqslant 1$. Assume the statement of the lemma holds for the pairs $(n-1, l)$ and $(n-1, l-1)$. Suppose $H_{n,l} f = 0$. $H_{n,l}$ is decomposable [8, 9] as

$$\begin{pmatrix} H_{n-1,l} & H_{n-1,l} \\ H_{n-1,l-1} & 0 \end{pmatrix}$$

and f is decomposable as $\binom{f_0}{f_1}$, $f_0, f_1 \in K_{2^{l+1}-1}^{2^{n-1}}$. Then both $H_{n-1,l}(f_0 + f_1) = 0$ and $H_{n-1,l-1} f_0 = 0$ hold. Since $f_0 + f_1 \in K_{2^{l+1}-1}^{2^{n-1}}$, $f_0 + f_1 = 0$ holds by the induction hypothesis on $(n-1, l)$. $f_0 = -f_1$ implies $|f_0| = |f_1| \leqslant 2^l - 1$. Hence $f_0 \in K_{2^l-1}^{2^{n-1}}$, and thus $f_0 = 0$ holds by the induction hypothesis on $(n-1, l-1)$. It therefore follows that $f = 0$. □

From Lemma 4.1 follows the next lemma, which is also analogous to a lemma proved by Clausen et al. [4] for the multivariate polynomials over GF(q).

Lemma 4.2. *For all nonnegative integers n, l, the mapping $\phi_{H_{n,l}}$ from $K_{2^l-1}^{2^n}$ to $K^{|Q(n,l)|}$, defined by $\phi_{H_{n,l}}(f) = H_{n,l} f$, is one-to-one.*

Proof. Suppose $H_{n,l} f = H_{n,l} g$. Then $H_{n,l}(f-g) = 0$ holds. Since $f - g \in K_{2^{l+1}-1}^{2^n}$, $f - g = 0$ by Lemma 4.1. Thus $f = g$ holds. □

'G-Interpol', shown in Fig. 3, essentially parallels the algorithm called 'Interpol' developed by Roth and Benedek [9]. It is a procedure which solves simultaneous linear equations represented by the matrix $H_{n,l}$, with the restriction that the solution be in $K_{2^l-1}^{2^n}$. This is a very efficient algorithm which runs in time polynomial in n even though the number of components of $H_{n,l}$ is $2^n|Q(n,l)|$, if all vectors f, f_+, f_0, f_1 are represented as lists of indices and values of their nonzero components. 'G-Interpol' here is almost the same as the original algorithm, except that $f_1 := f_+ - f_0$, $f_0 := f_+ - f_1, g_2 - g_0$(on lines 15, 19, 17, respectively, in Fig. 3) replace $f_1 := f_+ \oplus f_0$, $f_0 := f_+ \oplus f_1, g_2 \oplus g_0$ in the original algorithm. Note that this leaves the algorithm unchanged when it is applied to $K^m = \mathrm{GF}(2)^m$. (In GF(2), $\oplus$ happens to equal both $+$ and $-$ operations of the field, but $\oplus$ in Interpol needs to be generalized as $-$ in G-Interpol.) Another difference is that our version of the algorithm computes not only the indices of nonzero components but also their values.

Lemma 4.3 (Generalization of Lemma 3.1 and Theorem 3.1 in [9]). *Upon input of arbitrary nonnegative integers n, l and an arbitrary vector $g \in K^{|Q(n,l)|}$, the behavior of procedure 'G-Interpol' satisfies all of the following:*

1. *If there exists a vector $f \in K_{2^l-1}^{2^n}$* satisfying $g = H_{n,l} f$, *it outputs such f (it is unique) and STATUS = "success."*
2. *Otherwise, it outputs STATUS = "failure."*
3. *It runs in time*[1] $O(2^l n |Q(n,l)|)$.

[1] As mentioned earlier, in order to obtain this time complexity upper bound, we need to represent all vectors f, f_+, f_0, f_1 as lists of indices and values of their nonzero components.

```
procedure G-Interpol(n, l, g) return(f, STATUS)
n, l: nonnegative integers
g: an element of K^{|Q(n,l)|}
f: an element of K^{2^n}
STATUS: "success" or "failure"
begin
  STATUS := "success"
  if l = 0 then
    if g = 0 then f := 0 else STATUS := "failure"
  else if n = 0 then f := g
  else begin
    (f_+, STATUS) := G-Interpol(n-1, l, g_+)
    if STATUS = "success" then
      (f_0, STATUS) := G-Interpol(n-1, l-1, g_0)
      if STATUS = "success" then f_1 := f_+ - f_0
      if STATUS = "failure" or |f_0| + |f_1| ≥ 2^l then
        (f_1, STATUS) := G-Interpol(n-1, l-1, g_2 - g_0)
        if STATUS = "success" then
          f_0 := f_+ - f_1
          if |f_0| + |f_1| ≥ 2^l then STATUS := "failure"
        end if
      end if
      if STATUS = "success" then f := (f_0^f / f_1^f)
    end if
  end else
  return(f, STATUS)
end
end procedure
[NOTE]
g_+: the subvector of g consisting of components with indices smaller than {x_n}
g_0: the subvector of g consisting of components with indices no smaller than {x_n}
g_2: the subvector of g_+ with indices belonging to Q(n-1, l-1).
```

Fig. 3. Procedure G-Interpol.

Proof. The proof of this lemma given Lemma 4.2 is similar to the proof of Lemma 3.1 and Theorem 3.1 in [9]. □

Proof of Theorem 4.1 (*Part* 1 *continued given Lemmas* 4.1, 4.2, *and* 4.3). Applying Lemma 4.3 with $\lfloor \log k \rfloor + 1$ in place of l yields the bounds given in the theorem statement. □

Proof of Theorem 4.1 (*part* 2). This proof is a straigtforward generalization of the corresponding result on multivariate polynomials over GF(2) in [9]. Consider the following set of k-term linear combinations of monotone terms: $\mathscr{G} = \{\prod_{x\in S}(1-x)\prod_{y\in A_n - S} y\colon S\in Q(n,\lfloor \log k \rfloor)\}$. An arbitrary member $\prod_{x\in S}(1-x)\prod_{y\in A_n-S} y$ of $\mathscr{G}$ takes the value 1 at 0_S and 0 at all other assignments. Thus receiving the value 0 as the value of the target function at assignment 0_S rules out *no* members of $\mathscr{G}\cup\{0\}$ if $|S|>\lfloor \log k \rfloor$, and rules out *exactly one* member of $\mathscr{G}\cup\{0\}$ if $|S|\leqslant\lfloor \log k \rfloor$. Hence, the worst-case number of queries required to remove all but one of the members of $\mathscr{G}\cup\{0\}$ is at least $|\mathscr{G}\cup\{0\}|-1=|\mathscr{G}|=|Q(n,\lfloor \log k \rfloor)|$. □

4.2. Learning the inductive-read-k subclass

In this section, we present an efficient learning algorithm for the 'inductive read-k' linear combinations of monotone terms. Hellerstein and Warmuth [6] proposed and analyzed a learning algorithm for read-k multivariate polynomials over GF(2). The basic structure of 'Read-k-interpol', shown in Fig. 4, is based on their algorithm, but is improved by using 'G-Interpol' as a subroutine and generalized for a general vector space K^m over K, i.e., it also computes the *values* of nonzero components.

Theorem 4.2. *There exists an algorithm that exactly learns*[2] *inductive-read-k-*$\mathscr{F}$ *(as defined in Section 2) making function value queries at assignments* 0_S *for all* $S\in Q(n,\lfloor \log k \rfloor+2)$, *running in time* $O(kn|Q(n,\lfloor \log k \rfloor+2)|)$.

Proof. The theorem follows immediately from the following lemma (Lemma 4.4). In the lemma statement below, we let ind-k-K^{2^n} denote the set of those vectors representing inductive read-k linear combinations, namely:

$$\text{ind-}k\text{-}K^{2^n} = \{f\in K^{2^n}\colon f \text{ has at most } k \text{ nonzero components that are indexed by elements of } 2^{A_m}-2^{A_{m-1}} \text{ for each } m=1,2,\ldots,n\}.$$

Lemma 4.4. *Upon input of arbitrary nonnegative integers* n, l *and an arbitrary vector* $g\in K^{|Q(n,l+1)|}$, *the behavior of procedure 'Read-k-interpol' satisfies all of the following conditions*:

1. *If there exists a vector* $f\in ind\text{-}(2^l-1)-K^{2^n}$ *satisfying* $g=H_{n,l+1}f$, *it outputs such* f *(it is unique) and* $STATUS=$"*success.*"
2. *Otherwise, it outputs* $STATUS=$"*failure.*"
3. *It runs in time*[3] $O(2^l n|Q(n,l+1)|)$.

Proof. The proof of this lemma is similar to the proof of corollary in [6]. □

[2] Note that inductive-read-k-$\mathscr{F}$ properly contains read-k-$\mathscr{F}$.

[3] The same assumption as in Lemma 4.3 is necessary.

```
procedure Read-k-interpol(n, l, g) return(f, STATUS)
n, l: nonnegative integers
g: an element of K^{|Q(n,l+1)|}
f: an element of K^{2^n}
STATUS: "success" or "failure"
begin
  STATUS := "success"
  if n = 0 then f := g
  else begin
    (f_1, STATUS) := G-Interpol(n-1, l, g_2 - g_0)
    if STATUS = "success" then
      (f_0, STATUS) := Read-k-interpol(n-1, l, g_+ - H_{n-1,l+1} f_1)
      if STATUS = "success" then f = (f_1^0)
    end if
  end else
  return(f, STATUS)
end
end procedure
[NOTE]
g_+: the subvector of g consisting of components with indices smaller than {x_n}
g_0: the subvector of g consisting of components with indices no smaller than {x_n}
g_2: the subvector of g_+ with indices belonging to Q(n-1, l).
```

Fig. 4. Procedure Read-k-interpol.

Proof of Theorem 4.2 (*continued*). Finally, applying Lemma 4.4 with $l = \lfloor \log k \rfloor + 1$ gives the bound given in the theorem. □

5. Remarks

In this paper, we studied exact learnability of functions that can be defined as a weighted sum of a set of monotone terms. If instead of a weighted sum, an operator such as $\vee$ or $\oplus$ is applied on the terms, then the corresponding function class is DNF or the class of multivariate polynomials over GF(2). Exact learnability of these classes by queries has been studied by many researchers, and as described in Section 1, the starting point of our research was to extend the family of learnable classes by allowing more powerful queries.[4] From this view point, we compare a part of our results with

[4] Schapire and Sellie [10] have independently considered the same learning problem with the addition of *equivalence queries*, and have shown that the entire class is learnable with polynomially many queries in this case.

Table 1
Comparison of value of three function classes

Target set T		$\bigvee_{t\in T} t$	$\bigoplus_{t\in T} t$	$\sum_{t\in T} t$
k-term	U	$kn+n^{k-1}$ [5]	$n^{\lfloor \log k \rfloor+1}$ [9, Lemma 2.1]	$(n-\lfloor \log k \rfloor+1)k$ Theorem 3.1
	L	2^{k-1} [1, Theorem 2]	$(\frac{n}{\lfloor \log k \rfloor})^{\lfloor \log k \rfloor}$ [9]	$\frac{nk-1}{\log(k+1)}$ Theorem 3.3
Read-once	U	$\frac{1}{4}n^2+2n$ ([2, Theorem 5])	$\frac{1}{4}n^2+2n+1$	$\frac{1}{4}n^2+n+1$ Theorem 3.2
	L	?	?	?
Read-k ($k\geqslant 2$)	U	?	$n^{\lfloor \log k \rfloor+2}$ [6, Lemma 3]	$\frac{1}{2}((n-\lfloor \log k \rfloor)(n-\lfloor \log k \rfloor+1)+2)k$ Theorem 3.2
	L	$2^{\lfloor n/2 \rfloor}$ [1 Theorem 2]	?	?

T: a set of monotone terms, U: upper bound, L: lower bound, ?: We are not aware of any nontrivial bound. (Some bounds hold only for $n>1$ or $k>1$.)

other results on the learning problems for DNF and multivariate polynomials over GF(2).

Table 1 summarizes bounds on the number of function value queries required to identify a set of monotone terms when the operators applied on them are $\vee$, $\oplus$ and $+$. First, we explain past results concerning the $\vee$ and $\oplus$ columns and then compare the three columns for each row.

Angluin [1] and Gu and Maruoka [5] independently gave a learning algorithm for monotone DNF using at most nk membership queries and $k+1$ equivalence queries, where n and k are the numbers of variables and terms in the target formula, respectively. Gu and Maruoka [5] also showed that k-term monotone DNF can be learned with at most $nk+n^{k-1}$ ($k\geqslant 1$) membership queries and no equivalence queries. Angluin has also shown a result that implies lower bounds for the number of membership queries needed when no other queries are used: $2^{\lfloor n/2 \rfloor}$ for (read-twice) monotone DNF, 2^{k-1} for k-term monotone DNF provided $k-1\leqslant n/2$. In this paper, we have evaluated upper bounds for the number of membership queries needed for read-once monotone DNF and multivariate polynomials over GF(2) including coefficients ($\approx\frac{1}{4}n^2$) as indicated in Table 1, which improves somewhat upon the $O(n^2)$ bound due to Angluin et al. [2] for read-once monotone formulas.

As for multivariate polynomials[5] over GF(2), Roth and Benedek [9], and Hellerstein and Warmuth [6] independently showed exact learnability of the k-term subclass using $q(n,\lfloor \log k \rfloor+1)$ membership queries, where $q(n,l)=\sum_{i=0}^{l}\binom{n}{i}$. Roth and Benedek

[5] Ben-Or and Tiwari [3] showed that k-term subclass of multivariate polynomials over the reals is exactly learnable using at most $2k$ function value queries.

have also shown that the number of membership queries needed to learn the same class is at least $q(n,\lfloor \log k \rfloor)$. Hellerstein and Warmuth [6] proved exact learnability of read-k subclass with at most $q(n,\lfloor \log k \rfloor+2)$ membership queries.

The upper and lower bounds shown in the last column in Table 1 are a part of our results in this paper. Let us now compare the three columns in each row in Table 1. For read-once formulas, the upper bounds for the three function classes obtained so far are polynomial in n and have no significant difference. For k-term formulas, the upper bounds for all three function classes are also polynomial in n, but the degree of n progresses as $k-1$ for $\vee$, $\lfloor \log k \rfloor+1$ for $\oplus$ and 1 for $+$. The upper bounds for read-k $\oplus_{t\in T} k$ and $\sum_{t\in T} t$ are polynomial in n although the degrees of n are improved as $\lfloor \log k \rfloor+2$ for $\oplus$ and 2 for $+$, but no algorithm to learn read-k $\bigvee_{t\in T} t$ using a number of membership queries polynomial in n exists. Notice that the upper bound for $\sum_{t\in T} t$ is polynomial in both n and k, where k is the number of terms, but no such algorithms to learn $\bigvee_{t\in T} t$ and $\oplus_{t\in T} t$ exist. From consideration above, we can see that as information sources for identifying a set of monotone terms, $\sum_{t\in T} t$ is the most valuable, and then comes $\oplus_{t\in T} t$ and $\bigvee_{t\in T} t$ comes last.

Acknowledgment

We would like to thank Prof. Manfred K. Warmuth of U.C. Santa Cruz and Prof. Lisa Hellerstein of Northwestern University for sharing their work on learning GF(2) polynomials with us. We also wish to thank anonymous referees for offering some helpful comments.

References

[1] D. Angluin, Queries and concept learning, *Mach. Learning* **2** (1988) 319–342.
[2] D. Angluin, L. Hellerstein and M. Karpinski, Learning read-once formulas with queries, *J. ACM* **40**(1) (1993) 185–210.
[3] M. Ben-Or and P. Tiwari, A deterministic algorithm for sparse multivariate polynomial interpolation, in: *Proc. 20th Ann. ACM Symp. on Theory of Computing* (1988) 301–309.
[4] M. Clausen, A. Dress, J. Grabmeier and M. Karpinski, On zero-testing and interpolation of k-spase multivariate polynomials over finite fields, *Theoret. Comput. Sci.* **84** (1991) 151–164.
[5] Q.P. Gu and A. Maruoka, Learning boolean functions, Tech. Report of IEICE COMP87-82, 1988.
[6] L. Hellerstein and M. Warmuth, Interpolating GF[2] polynomials, unpublished manuscript.
[7] K. Kakihara and H. Imai, Notes on the PAC learning of geometric concepts with additional Information, in: *Proc. 3rd Workshop on ALT* (1992) 252–259.
[8] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes* (North-Holland, Amsterdam, 1977).
[9] R.M. Roth and G.M. Benedek, Interpolation and approximation of sparce multivariate polynomials over GF[2], *SIAM J. Comput.* **20**(2) (1991) 291–314.
[10] R. Schapire and L. Sellie, Learning sparse multivariate polynomials over a field with queries and counterexamples, in: *Proc. 6th Ann. Workshop on Comput. Learning Theory* (ACM, New York, 1993) 17–26.