OPTIMAL ATTRIBUTE-EFFICIENT LEARNING OF DISJUNCTION, PARITY, AND THRESHOLD FUNCTIONS

Ryuhei Uehara, Kensei Tsuchida, and Ingo Wegener

Addresses

Ryuhei Uehara, Center of Information Science, Tokyo Woman's Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167, Japan, email: uehara@twcu.ac.jp

Kensei Tsuchida, Faculty of Engineering, Toyo University, Kujirai, Kawagoe-shi, Saitama 350, Japan, email: kensei@krc.eng.toyo.ac.jp

Ingo Wegener, FB Informatik, LS II, Univ. Dortmund, 44221 Dortmund, Germany, email: wegener@ls2.informatik.uni-dortmund.de

Key words: decision trees, complexity of Boolean functions, attribute-efficient learning, derandomization.

Abstract

Decision trees are a very general computation model. Here the problem is to identify a Boolean function f out of a given set of Boolean functions F by asking for the value of f at adaptively chosen inputs. For classes F consisting of functions which may be obtained from one function g on n inputs by replacing arbitrary $n-k$ inputs by given constants this problem is known as attribute-efficient learning with k essential attributes. Results on general classes of functions are known. More precise and often optimal results are presented for the cases where g is one of the functions disjunction, parity or threshold.

1 INTRODUCTION

Decision trees are the adequate computation model, if one likes to identify an unknown object from a given universe by asking queries with a finite number of possible answers. The queries may be asked adaptively, i. e., which question is asked can depend on previous answers. The aim is to minimize the worst case number of queries which is the depth of the decision tree.

The problem in this general setting has already been discussed by Picard (1965). A lot of different problems may be treated in this general framework (see Ahlswede and Wegener (1987)). Boolean decision trees are the most important subclass of decision trees. A known Boolean function f has to be evaluated at an unknown input a. We may query for the bits a_i of a. The complexity measure is the number of queries. All queries have the same cost. We only remark that the minimal depth of Boolean decision trees equals the minimal depth of branching programs or binary decision diagrams (see Wegener (1987)).

Here another class of problems is considered. A class F of Boolean functions on n variables is given and we want to find out which function $f \in F$ is chosen by an adversary. For this purpose we may choose adaptively inputs $a \in \{0,1\}^n$ and the adversary has to answer the query a with the correct value of $f(a)$. The information theoretic lower bound is $\lceil \log |F| \rceil$, since we ask binary queries. This is optimal, e. g., for the class of all 2^{2^n} Boolean functions. Then we have to query all 2^n inputs. In the following we consider much smaller classes F. The class of functions representable by read-once formulas has been investigated by Angluin, Hellerstein, and Karpinski (1993).

The following situation is of particular interest. The class F is based on one Boolean function g. For some sets $S \subseteq \{1,\ldots,n\}$ (typically all sets or all sets of cardinality k) there exists one function $g_S \in F$ which is a subfunction of g where each x_i, $i \notin S$, is replaced by some given constant a_i. The identification of the unknown function $f \in F$ is equivalent to the identification of the set of essential attributes S. Angluin (1988) gives an overview on query learning. The situation described above is known as attribute-efficient learning. Already Littlestone (1988) and recently Bshouty and Hellerstein (1996) have designed quite efficient algorithms for attribute-efficient learning. Their algorithms work for quite general classes of functions and are efficient in particular for small k. Because of their generality they do not meet the lower bound for special classes of functions.

We like to obtain more precise results for special classes of functions. The classes OR and OR(k) are based on the disjunction of n variables. The not essential variables are replaced by the constant 0. The class OR contains all OR_S, $S \subseteq \{1,\ldots,n\}$, i. e., the disjunction of all x_i, $i \in S$, and OR(k) all OR_S where $|S| = k$. In the same way we obtain the classes PAR and PAR(k) for the parity function which decides whether the number of ones in the input is odd or even. The threshold function is the basis of discrete neural nets. The threshold function on n variables and threshold t decides whether the input contains at least t ones. Again we replace the not essential variables by 0. Then we consider the classes THR, THR(k), and $\mathrm{THR}_t(k)$, i. e., we distinguish whether we consider all $S \subseteq \{1, \dots, n\}$ or all $S \subseteq \{1, \dots, n\}$ of cardinality k and we distinguish whether the threshold t is unknown or known.

Our choice of classes of functions is motivated in the following way. Our results fit into the theory of attribute-efficient learning. The functions disjunction, parity, and threshold are the most important basic gates in circuit theory and, therefore, the most fundamental functions. If we know that some wires into a gate are destroyed and we like to learn which wires are destroyed by testing the gates on chosen inputs, we obtain the considered problem. This application motivates also the investigation of large k, e. g., $k = n - O(1)$. This situation leads to the attribute-efficient learning with respect to the *not* essential attributes.

In the Sections 3, 4, and 5 we consider the three different classes of functions. In Section 2 we collect the information theoretical lower bounds.

Remark: A preliminary version appears as extended abstract in the proceedings of the EuroCOLT conference.

2 SIMPLE LOWER BOUNDS

Since the queries have two possible answers, $\lceil \log |F| \rceil$ queries are necessary for the class F.

Theorem 1: The following lower bounds hold:

i) n for OR and PAR.

ii) $\left\lceil \log \binom{n}{k} \right\rceil$ for OR(k), PAR(k), and THR$_t$(k), if $k \geq t > 0$. This bound is larger than $k \log(n/k)$ and $(n-k) \log(n/(n-k))$.

iii) $\left\lceil \log \left(\binom{n}{k} \cdot k + 2 \right) \right\rceil$ for THR(k).

iv) $n - 1 + \lceil \log(n+1) \rceil$ for THR.

Proof: The classes OR and PAR contain 2^n functions and the classes OR(k), PAR(k) and THR$_t$(k) contain $\binom{n}{k}$ functions. The estimations of $\log \binom{n}{k}$ are standard calculations. For THR(k) we have the possible threshold values $t \in \{1, \dots, k\}$ each leading to $\binom{n}{k}$ functions. For different t we obtain different functions. Moreover, we obtain for $t = 0$ the constant 1 and for $t > k$ the constant 0. For the class THR we have

$$\sum_{1 \leq k \leq n} k \binom{n}{k} + 2$$

different functions, since we cannot distinguish whether a constant function "depends" on k or k' variables. The number of different functions equals $n2^{n-1}+2$ and $\lceil \log(n2^{n-1} + 2) \rceil = \left\lceil \log 2^{n-1} \left(n + 2^{-(n-2)} \right) \right\rceil \geq n - 1 + \lceil \log(n+1) \rceil$. □

Later we derive for some cases better lower bounds by adversary arguments.

3 DISJUNCTIONS

By de Morgan's rules it easily follows that all results for disjunctions also hold for conjunctions, if the non essential variables for conjunctions are replaced by ones.

A query is an input $a \in \{0,1\}^n$. In the following we identify queries a with the query set $A = \{i \mid a_i = 1\}$.

The class of functions OR causes no problem. We meet the lower bound n with the simple nonadaptive strategy asking all query sets of size 1 also called singletons.

Proposition 2: The class $\mathrm{OR}(k)$ can be learned with at most $2k\left(\lceil \log(n/2k) \rceil + 1\right)$ queries.

Proof: We start with $2k$ query sets which build a partition of $\{1, \ldots, n\}$ and whose size is bounded by $\lceil n/(2k) \rceil$. A query set leading to the answer 1 is called a winner. By definition of the problem there are at most k winners which again are split into two query sets of almost the same size. Hence, $\lceil \log(n/2k) \rceil + 1$ phases with at most $2k$ queries each are sufficient. The k winners of the last phase are the essential elements (variables). □

This result leads to two new problems. The upper bound is asymptotically optimal, if k is not too large. But it is by the factor 2 larger than the lower bound of Theorem 1. Can we get rid of this factor? For large k the lower bound is much smaller than the upper bound. What is the right value?

We answer these questions with the following two theorems.

Theorem 3: The class $\mathrm{OR}(k)$ can be learned with at most $k\lceil \log(n/k) \rceil + 2k - 2$ queries.

Proof: We partition the set of elements to k sets of size at most $\lceil n/k \rceil$. We ask these k sets as queries. If a set S is a winner, we can find by binary search with at most $\lceil \log\lceil n/k \rceil \rceil = \lceil \log(n/k) \rceil$ queries an essential element x. Then we test whether $S - \{x\}$ is still a winner and continue in this way. For each of the k essential elements the binary search can be done with at most $\lceil \log(n/k) \rceil$ queries. Moreover, we ask k queries in the beginning. For each essential element x we have found we ask an additional question (the question $S - \{x\}$ above). We can save this question for the last two essential elements. After having found the last essential element we are done. If we have found the last but one essential element, we know whether there is another set which is a winner. If and only if not, the corresponding set $S - \{x\}$ is a winner. □

If $k = o(n)$, the leading term in Theorem 3 has the same constant factor as the lower bound of Theorem 1, namely 1.

Theorem 4: If $n/2 \le k \le n-1$, $n-1$ queries are necessary and sufficient for $\mathrm{OR}(k)$.

Proof: The claim is proved with an adversary argument. First we consider the class $\mathrm{OR}(n-1)$. Query sets of size larger than 1 are useless, since the answer 1 is known in advance. Hence, only singletons are tested and the adversary may answer the first $n-1$ queries by 1.

We restrict ourselves w. l. o. g. to strategies which do not include identified elements (elements which are known to be essential or not essential) into further query sets. The adversary strategy is the following. If $k = n-1$, we follow the strategy described above. Otherwise the adversary does the following. As long as the query sets are disjoint the adversary answers 0 for the query sets of size 1 and answers 1 for larger query sets. The answers 1 are allowed, since $k \geq n/2$. If the adversary has declared $n-k-1$ elements as not essential, we may erase these queries and are left with the problem OR(k) on $k+1$ elements for which we can use the adversary strategy described at the beginning of this proof. If for the first time an element is contained for the second time in a query set, the adversary declares this element as essential. By the assumption that already identified elements are not contained in later query sets, the first query A containing i contains also another element j. This element is declared as not essential. In this way we may eliminate 2 queries and are left with the adversary strategy for the problem OR($k-1$) on $n-2$ elements. For this problem we prove with the same strategy that $n-3$ queries are necessary. $\square$

4 PARITY FUNCTIONS

The class PAR can be learned as the class OR with n queries of the singletons.

Parity (or mod 2) is like disjunction a binary, associative and commutative operation with 0 as neutral element. But $(\{0,1\}, \oplus)$ is an additive group while the equation $1 \vee x = 0$ has no solution. Therefore, we obtain here results different from the previous section. The class OR(k) is efficiently learnable for small k and needs the maximum of $n-1$ queries if $k \geq n/2$. The group properties of the parity operation imply that for given n it is the same whether we like to learn PAR(k) or PAR($n-k$).

Proposition 5: For given n, the complexity of learning PAR(k) is equal to the complexity of learning PAR($n-k$).

Proof: Let g_1 be the parity function on k of the n variables and g_2 be the parity function on the $n-k$ other variables. This defines a one-to-one mapping between the classes PAR(k) and PAR($n-k$). For a query set S let $a_1(S)$ and $a_2(S)$ be the answers given for g_1 and g_2 resp. Then $a_1(S) \oplus a_2(S) \equiv |S| \bmod 2$. Hence, we can compute $a_1(S)$ from $a_2(S)$ and vice versa. A strategy for PAR(k) which identifies g_1 identifies with the corresponding answers g_2, if it is used for PAR($n-k$). $\square$

Because of Proposition 5 we assume w. l. o. g. that $k \leq n/2$. For the OR-classes, one essential variable can be identified with binary search using $\lceil \log n \rceil$ queries. For the PAR-classes the same holds, if the number of essential elements is odd. If we partition the set of elements into two parts, one has an odd number of essential elements and the other one an even number. With one query we can find out which is the "odd part" and can continue with this set. After having found one essential element, we have an even number of unknown essential elements and are in the difficult situation. We cannot make use of the already found essential element. Otherwise we may find the same essential element for a second time. If we find in the situation of an even number of unknown essential elements a set S with an odd number of unknown essential elements, then the complement of S with respect to the unidentified elements also contains an odd number of essential elements. We can apply the binary search technique to both sets. We get good upper bounds (see Theorem 1) for $k \in \{1, 2, 3\}$.

Proposition 6: i) PAR(1) can be learned with $\lceil \log n \rceil$ queries.

ii) PAR(2) can be learned with $3\lceil \log n \rceil - 2$ queries.

iii) PAR(3) can be learned with $4\lceil \log n \rceil - 3$ queries.

Proof: The first result follows from the binary search technique. For the second claim we use a so-called separating system $S_0, \ldots, S_{\lceil \log n \rceil - 1}$. That means, for $i \neq j$ there is a set S_m where $i \in A_m$ and $j \notin A_m$ or vice versa. For this purpose let A_m consist of the numbers j whose m-th bit in the binary representation equals 1. First we query the sets of the separating system. If i and j are the essential elements, the corresponding set A_m leads to the answer 1. Then we may start binary searches on A_m and its complement, sets of at most $\lceil n/2 \rceil$ elements. The third result is obvious, since we may find the first essential element with binary search. Then we can apply the second result (even for a set of $n-1$ elements). We save one query, since the first query can be chosen on a set belonging to the separating system in the second round. □

How can we find for larger k a set with an odd number of essential objects? This question is different from the more general approach of Bshouty and Hellerstein (1996). They considered p. e. c.-classes of functions which are closed under projections and embeddings. OR(k) and PAR(k) are p. e. c.-classes but many other classes of functions have the property p. e. c. In the more general situation one looks for so-called (n, k, r, α)-splitters. Actually, Bshouty and Hellerstein (1996) do not use the notion splitter.

Definition 7: An (n, k, r, α)-splitter of size s is a sequence $P_1, \ldots, P_s$ of partitions of $\{1, \ldots, n\}$ into r sets such that for each set S of cardinality k at least a fraction of $\alpha > 0$ partitions put the elements of S into r different sets.

Lemma 8: An (n, k, r, α)-splitter of size s implies that PAR(k) can be learned with $rs + k\lceil \log n \rceil$ queries.

Proof: With rs queries of the sets in $P_1, \ldots, P_s$ we find one partition where k subsets give the answer 1. With binary search on these sets we find the k essential elements. □

Since the notion of splitters is quite new, the results on splitters are scattered through the literature. Moreover, it has to be distinguished whether splitters only exist or can be constructed efficiently. Friedman (1984) constructs splitters for constant k leading to $O(\log n)$ learning algorithms for PAR(k) and constant k. This construction was used for the construction of Boolean formulas. Ragde and Widgerson (1991) (based on Mehlhorn (1982)) and Håstad, Wegener, Wurm, and Yi (1994) also used splitters for the construction of Boolean circuits. Naor, Schulman, and Srinivasan (1995) explicitly investigate splitters under this name. The deterministic construction of splitters in Bshouty and Hellerstein (1996) is a rediscovery of the construction used by Håstad, Wegener, Wurm, and Yi (1994).

These splitters lead by Lemma 8 to the existence of learning algorithms for PAR(k) with $O(k^3 \log n)$ queries and the efficient construction of learning algorithms with $O(k^4 \log n)$ queries (Hofmeister (1996)). Using special properties of the parity function we improve the upper bound. Since decision trees are a nonuniform computation model we concentrate our discussion on the minimal number of necessary queries and not on the efficient construction of the learning algorithms.

For the class of parity functions we may find the essential elements sequentially. We do not have to partition the essential elements in classes of size 1. For the identification of one essential element it is sufficient to find a set with an odd number of essential elements.

First we consider randomized learning algorithms with zero-error. Either we can guarantee that the algorithm always gives the right answer and that the expected number of queries is small or we guarantee an upper bound on the number of queries and a small upper bound on the probability that the algorithm answers "don't know". (For an introduction to randomized algorithms see Motwani and Raghavan (1995)). We are not the first to investigate randomized decision trees, see e. g. Saks and Widgerson (1986) and Heiman and Widgerson (1991).

Obviously, one half of the queries leads to the answer 1. Asking randomized queries we get in expected time 2 a query set with answer 1. Then, we find two essential elements with binary search. This leads to an error-free randomized algorithm with an expected number of $k\,(\lceil \log n \rceil + 1)$ queries. In order to meet the lower bound we should decrease the size of the query sets.

Theorem 9: For PAR(k) there is a randomized error-free learning algorithm with an expected number of $k \log(n/k) + O(k)$ queries. If the number of queries is bounded by $O(k \log(n/k))$, the probability of not having identified the unknown function can be bounded by a function tending with respect to $k \log(n/k)$ exponentially fast to 0.

Proof: The algorithm works as follows. If we have already found l essential elements (in the beginning $l = 0$), we work on the other $n - l$ elements and query random sets of size $(n-l)/(k-l)$ (more precisely $\lceil (n-l)/(k-l) \rceil$). The probability that such a set contains an odd number of essential elements can be bounded in the following way. If $k-l$ is small, this probability tends to $1/2$ which can be proved by elementary calculations. Otherwise, the probability distribution of the number of essential elements in the random query set tends to the Poisson distribution with parameter $\lambda = 1$. Hence, the probability that the query set contains exactly one essential element tends to $e^{-1} \geq 0.36$ (e the Eulerian constant $2.718\ldots$). Hence, the probability of success is for large n bounded below by $1/3$.

For a successful query we perform the binary search only on the small query set and not on the large complement. For the binary searches we are done in any case with

$$\sum_{1\le m\le k}\left\lceil \log\frac{n-k+m}{m}\right\rceil \le \sum_{1\le m\le k}\log\frac{n}{m}+O(k)=k\log n-\log(k!)+O(k)$$

$$=k\log n-(k\log k-O(k))+O(k)=k\log(n/k)+O(k)$$

queries. The expected number of queries to find the next set with an odd number of essential elements is bounded by 3 leading to an additional expected number of $3k$ queries.

For the second claim we get smaller upper bounds than for general error-free randomized algorithms. Only a small part of our algorithm is randomized. Furthermore, we have independent trials each with a probability of success at least $1/3$. We need k successes and may perform $O(k\log(n/k))$ trials. Now the probability of less then k successes can be estimated with Chernoff's bounds. □

Now we look for deterministic learning algorithms. The main idea is to "recycle" query sets, i. e. to try to use them more than once. A random query set S contains approximately half of the essential elements. The set is good, if it contains an odd number of essential elements. If the number of essential elements in S is even, we try other sets. If we find the first essential element i contained in S, we know without any further query that $S-\{i\}$ contains an odd number of essential elements. These ideas are made precise in the next proposition which will be improved later. Moreover, we derandomize the algorithm.

Proposition 10: The class PAR(k) can be learned with $\left\lceil\log\binom{n}{k}\right\rceil+k\lceil\log n\rceil+2$ queries.

Proof: If $k=\Omega(n)$, the result is obvious, since $n-1$ queries of singletons suffice. If $k\le n/3$,

$$\binom{n}{k-1}\le\frac{1}{2}\binom{n}{k}$$

and

$$\binom{n}{k}+\binom{n}{k-1}+\binom{n}{k-2}+\cdots+\binom{n}{1}\le 2\binom{n}{k}.$$

We choose $m=\left\lceil\log\binom{n}{k}\right\rceil+2$ random sets $R_1,\dots,R_m$. Let $S\subseteq\{1,\dots,n\}$ be one set where $|S|\in\{1,\dots,k\}$. The probability that $S\cap R_j$ has odd cardinality equals $1/2$.

The probability that all random sets are bad for S, i. e. $S\cap R_1,\ \dots,\ S\cap R_m$ all have even cardinality, equals $\frac{1}{4}\binom{n}{k}^{-1}$. Therefore, the probability that for at least one of the less than $2\binom{n}{k}$ sets S all random sets are bad is bounded above by $1/2$. Hence, there exist m query sets $T_1,\dots,T_m$ such that for each $S\subseteq\{1,\dots,n\}$ with $1\le|S|\le k$ there exists a set T_j such that $|S\cap T_j|$ is odd.

The sets $T_1,\dots,T_m$ are used as query sets. Let S be the unknown set of essential elements. Then $\left|S\cap T_{i(1)}\right|$ is odd for some $i(1)$ and we find an essential element $j(1)$ with binary search on $T_{i(1)}$. Let $S_1 := S - \{j(1)\}$. Then $\left|S_1 \cap T_{i(2)}\right|$ is odd for some $i(2)$ and we find an essential element $j(2)$ with binary search on $T_{i(2)} - \{j(1)\}$ and so on. Here we have used the fact that we have implicitly knowledge about the query set $T_i - \{j(1)\}$ if have asked the query T_i and know that $j(1)$ is essential.

Altogether we have asked m queries $T_1, \ldots, T_m$ and need $k\lceil \log n \rceil$ further queries for the binary searches. □

The term $\left\lceil \log \binom{n}{k} \right\rceil$ is equal to the information theoretic lower bound. In order to obtain an asymptotically optimal algorithm the term $k \log n$ should be replaced by $O\left(k \log(n/k)\right)$. If $k = O\left(n^{1-\varepsilon}\right)$, nothing has to be done. Hence, we have to consider functions like $k = n/\log^2 n$ or $k = n/\log\log n$.

Theorem 11: The class PAR(k) can be learned with $O\left(k \log(n/k)\right)$ queries.

Proof: Because of Proposition 5 and the simple strategy using $n-1$ queries we can assume that $k = n/h(n)$ for some function $h(n)$ tending to ∞. The first idea is to replace the random sets in the proof of Proposition 10 by random sets of size $n/k = h(n)$. For sets S with $k' = o(k)$ elements the intersection of S and such a random set is with large probability empty. Therefore, we work with random sets of size $(n/k)^2 = h(n)^2$. The cost for each binary search is increased by a factor 2 only. For sets S where $|S| \geq n/h(n)^2$ the probability of success for a random set is at least $1/3$. Hence, with $O\left(\log \binom{n}{k}\right)$ sets we can identify all but $n/h(n)^2$ essential elements.

In the second phase we work with sets of size $h(n)^4$ and can identify all but $n/h(n)^4$ essential elements. This process is continued until all essential elements can be identified.

The number of query sets besides the queries in the binary searches is bounded by

$$\begin{aligned}
& O\left(\log \binom{n}{n/h(n)} + \log \binom{n}{n/h(n)^2} + \log \binom{n}{n/h(n)^4} + \cdots\right) \\
= \ & O\left(\frac{n}{h(n)} \log h(n) + \frac{n}{h(n)^2} 2 \log h(n) + \frac{n}{h(n)^4} 4 \log h(n) + \cdots\right) \\
= \ & O\left(k \log(n/k)\right).
\end{aligned}$$

We still have to estimate the number of queries during the binary searches. In the first phase we find not more than $n/h(n)$ essential elements each with a cost of $2\log h(n)$. In the second phase we find not more than $n/h(n)^2$ essential elements each with a cost of $4 \log h(n)$ and so on. Hence, the same estimation as before shows that $O\left(k \log(n/k)\right)$ queries suffice. □

Again we have found asymptotically optimal algorithms. These algorithms cannot be constructed efficiently. In applications one may use the randomized algorithms.

5 THRESHOLD FUNCTIONS

The simple strategy of testing all singletons is optimal for the classes OR and PAR. For the class THR we obtain an optimal algorithm, if $n+1$ is not a power of 2. Otherwise we miss the lower bound by 1.

Theorem 12: The class THR can be learned with $n-1+\lceil\log(n+2)\rceil$ queries.

Proof: In the first phase we only use query sets of type $\{1,\dots,m\}$. Since threshold functions are monotone, we can find with binary search and $\lceil\log(n+2)\rceil$ queries the minimal $s\in\{0,\dots,n\}$ such that the answer to the query $\{1,\dots,s\}$ is 1 or we find out that such an s does not exist. In the last case we are done, since the function we look for is the constant 0. If $s=0$, the function is the constant 1. If $s\in\{1,\dots,n\}$, we know that s is an essential element. The query $\{1,\dots,s\}-\{j\}$ decides for $j\in\{1,\dots,s-1\}$ whether j is essential and the query $\{1,\dots,s-1,j\}$ does the same for $j\in\{s+1,\dots,n\}$. These are $n-1$ additional queries. □

Theorem 13: Let $0\le t\le k\le n$. The class $THR_t(k)$ can be learned with at most $(k-1)\log\frac{n-1}{k-1}+3k-3+\lceil\log(n+2)\rceil$ queries.

Proof: The first phase is the same as in the proof of Theorem 12. Afterwards we know the parameters n, k, and t. Moreover, if we are not done, we know an essential element s such that the query $\{1,\dots,s\}$ leads to the answer 1 and the query $\{1,\dots,s-1\}$ leads to the answer 0. Hence, we know that $\{1,\dots,s-1\}$ contains exactly $t-1$ essential elements and $\{s+1,\dots,n\}$ contains the remaining $k-t$ essential elements.

We use an algorithm for OR$(k-t)$ on the set $\{s+1,\dots,n\}$ and add to each query the set $\{1,\dots,s-1\}$. This algorithm identifies the $k-t$ essential elements in $\{s+1,\dots,n\}$. Then we use an algorithm for AND$(t-1)$ on the set $\{1,\dots,s-1\}$ and add to each query the essential element s. This algorithm identifies the $t-1$ essential elements in $\{1,\dots,s-1\}$.

In Section 3 we have seen that the complexity of learning AND-classes is the same as for the corresponding OR-classes. By Theorem 3 the number of queries of the second and third phase is bounded by

$$\begin{aligned}(k-t)\left\lceil\log\frac{n-s}{k-t}\right\rceil &+ 2(k-t)+(t-1)\left\lceil\log\frac{s-1}{t-1}\right\rceil+2(t-1)\\ &\le (k-t)\log\frac{n-s}{k-t}+(t-1)\log\frac{s-1}{t-1}+3k-3\\ &\le (k-1)\log\frac{n-1}{k-1}+3k-3.\end{aligned}$$

The last inequality is for $a=\frac{s-1}{n-1}$ and $b=\frac{t-1}{k-1}$ equivalent to

$$-b\log b-(1-b)\log(1-b)\le -b\log a-(1-b)\log(1-a).$$

This inequality holds, since $0\le a,\ b\le 1$, see e.g. Ahlswede and Wegener (1987), Lemma 5.3, p.20. Such inequalities are used, e.g., for the proof of the noiseless coding theorem, the left hand side is the entropy for the distribution $(b,1-b)$. □

Theorem 14: The class THR(k) can be learned with at most $2(k-1)\log\frac{n-1}{k-1}+6k-6+\lceil\log(n+2)\rceil$ queries.

Proof: The first phase of the algorithms in the proofs of Theorem 12 and Theorem 13 does not need the knowledge of t. Hence, we can use this phase also here.

Afterwards we can solve the problem by the application of algorithms for OR (and equivalently AND). But now the number of essential elements is not known, since it depends on the unknown parameter t. The algorithms in the proofs of Proposition 2 and Theorem 3 rely on the fact that the number of essential elements is known. But with at most $4k$ additional queries we can make the algorithm in the proof of Proposition 2 independent of the knowledge of k. We start with the query $\{1,\ldots,n\}$. During the whole algorithm all winners (queries with answer 1) are split into two sets of almost the same size and these sets are used as query sets. Without knowing k the algorithm does not produce more than k winners in each phase. The additional queries are those during the phases $0,\ldots,\lceil\log k\rceil$ where the query sets have approximately the size $n/2^0,\ldots,n/2^{\lceil\log k\rceil}$. The number of these queries is bounded by $2^{\lceil\log k\rceil+1}\leq 4k$. Hence, the number of queries for THR(k) is bounded by

$$\lceil\log(n+2)\rceil+2(k-t)\lceil\log\frac{n-s}{2(k-t)}\rceil+6(k-t)+2(t-1)\lceil\log\frac{s-1}{2(t-1)}\rceil+6(t-1).$$

Now we can use the same estimations as in the proof of Theorem 13. □

We know from the Sections 2 and 3 that these upper bounds are asymptotically optimal for $k\leq n/2$. The optimal number of queries does not rely essentially on the threshold value t or even on the knowledge of t. Does this hold also for $k>n/2$?

First we see that we get OR, if $t=1$. We know that for this case we need $n-1$ queries, if $k\geq n/2$. Moreover, the cases $t=1$ and $t=k$ are equivalent. This can be generalized.

Proposition 15: For given n, the complexity of learning $THR_t(k)$ is equal to the complexity of learning $THR_{k-t+1}(k)$.

Proof: Let S be the set of essential elements for the problem $THR_t(k)$. A query Q leads to the answer 1 if and only if $|Q\cap S|\geq t$. If S is the set of essential elements for the problem $THR_{k-t+1}(k)$, the query $Q':=\{1,\ldots,n\}-Q$ leads to the answer 1 if and only if $|Q'\cap S|\geq k-t+1$ which is equivalent to $|Q\cap S|\leq t-1$. Hence, we can simulate the query Q for $THR_t(k)$ by the query Q' for $THR_{k-t+1}(k)$ and vice versa. □

We are still left with a variety of parameter choices. The case $k=n-1$ leads to an interesting result. Afterwards we investigate the majority function, namely the case $k=n/2$.

Theorem 16: Let $t\leq n/2$. To learn the class $THR_t(n-1)$

- $\max\{\lceil\log n\rceil,\lceil(n-1)/t\rceil\}$ queries are necessary and

- $\lceil n/t \rceil + \lceil \log t \rceil - 1$ queries are sufficient.

By Proposition 15 we obtain similar results for the case $t > n/2$.

Proof: The lower bound $\lceil \log n \rceil$ is the simple information theoretical lower bound. For the lower bound $\lceil (n-1)/t \rceil$ a simple adversary argument works. A query set of size $t+1$ or larger is useless, since we know in advance that the answer is 1. A query set of size $t-1$ or smaller is useless, since we know in advance that the answer is 0. Hence, all query sets have size t. An adversary may answer 1 for the first $\lceil (n-1)/t \rceil - 1$ questions. Then there are still at least 2 candidates for the not essential element and another query is necessary.

For the upper bound we construct $\lceil n/t \rceil$ query sets of size t such that each element is contained in at least one set. We look for a query set leading to the answer 0, since this set contains the only not essential element. At least one of the queries is good for our purposes. Hence, it is sufficient to ask $\lceil n/t \rceil - 1$ queries. Since $t \le n/2$, we now can use binary search on this set, if we add always as many already identified elements such that the query size becomes t. □

This result is interesting, since we now know of an example where large and small threshold values, in particular the cases AND and OR, are difficult while $t = n/2$ is easy. For $THR_{n/2}(n-1)$ we have shown that $\lceil \log n \rceil$ queries are sufficient.

Moreover, we can prove that the threshold value $n/2$ admits efficient learning algorithms for all numbers of essential elements. We leave it to the reader to generalize this result to other threshold values.

Theorem 17: The class $THR_{n/2}(n-k)$ can be learned with $O(k \log(n/k))$ queries.

Proof: If $k > n/2$, nothing has to be learned and 0 queries are sufficient. If $k \le n/2$ and $k = \Theta(n)$, the upper bound follows from Theorem 12. Hence, we may assume that $k \le n/8$. Moreover, we assume that k divides n. Otherwise, we may add less than k elements known to be not essential. Additional not essential elements have no influence on the answer to queries and can therefore be added. This does not hold for additional essential elements.

We partition $\{1, \ldots, n\}$ into blocks $B_i, 1 \le i \le k$, where $B_i = \{(i-1)n/k + 1, \ldots, in/k\}$. We like to search for the k not essential elements using one of the strategies for OR(k) from Proposition 2 or Theorem 3. Each query set S will be a subset of some B_i. If S has size j, we need a set S_j which is disjoint from S (or even B_i) and which contains exactly $n/2 - j$ essential elements. The query $S \cup S_j$ then decides whether S contains at least one not essential element. Hence, we may use the OR-strategies, if the so-called dummy sets S_j are available. Since $|S| \le n/k$, we only need dummy sets with $n/2 - j$ essential elements where $1 \le j \le n/k$.

As in the proof of Theorem 12 we determine the minimal s such that $S^* = \{1, \ldots, s\}$ contains $n/2$ essential elements. We know in advance that $n/2 \le s \le n/2 + k$. Hence,

$\lceil \log(k+1) \rceil$ queries are sufficient. By asking the queries $S^* - \{1\}, \ldots, S^* - \{k + n/k\}$ we find at least n/k essential elements $e_1, \ldots, e_{n/k}$. The set $S^* - \{e_1, \ldots, e_j\}$ contains exactly $n/2 - j$ essential elements, $1 \le j \le n/k$. With $\lceil \log(k+1) \rceil + k + n/k$ queries we have found dummy sets which are, since $k \le n/8$, for large n disjoint from the last third of the blocks B_i. We may repeat the whole process twice where we shift the elements by $n/3$ and $2n/3$ resp. positions in order to find dummy sets for all blocks.

The number of queries for the computation of dummy sets is bounded by $3\,(\lceil \log(k+1) \rceil + k + n/k)$ and afterwards we may use the OR(k)-strategy from Proposition 2 or Theorem 3.

This bound is good enough if and only if $n/k = O(k \log(n/k))$. This condition is equivalent to $k = \Omega(n^{1/2}/\log^{1/2} n)$. For smaller values of k we need a new strategy for the computation of dummy sets.

As usual we first determine the smallest set $S^* = \{1, \ldots, s\}$ with exactly $n/2$ essential elements. For some l with $s < l \le 2n/3$ we partition the elements $l, \ldots, n$ to $k+1$ subsets $R_1, \ldots, R_{n+1}$ of equal size $r \le n/k$. We try to find the not essential elements in $R_1, \ldots, R_{k+1}$ by using an OR-strategy for at most k essential elements. This can be done efficiently by testing only subsets of the R-sets. For the queries $R_1, \ldots, R_{k+1}$ we need a dummy set D with exactly $n/2 - r$ essential elements. We consider the following candidates $D_0 = S^* - \{1, \ldots, r\}, \ldots, D_k = S^* - \{1, \ldots, r+k\}$. The set D_0 contains at least $n/2 - r$ essential elements and the set D_k at most $n/2 - r$ essential elements.

What happens if we query the sets $R_1 \cup D_j, \ldots, R_{k+1} \cup D_j$? If the answers are all 0, D_j is too small. Since the number of not essential elements is k, at least one set R_i consists of r essential elements only. Since also $R_i \cup D_j$ leads to the answer 0, D_j is too small. Otherwise, D_j may be too large. Hence, for our purposes it is sufficient to find the largest j and, therefore , the smallest D_j such that at least one of the queries $R_1 \cup D_j, \ldots, R_{k+1} \cup D_j$ leads to the answer 1. This can be done with binary search with $(k+1)\lceil \log(k+1) \rceil$ queries. Then D_j contains exactly $n/2 - r$ essential elements. Moreover, the set R_i, such that $R_i \cup D_j$ leads to the answer 1, consists of r essential elements only. Hence, larger dummy sets can be obtained easily by taking D_j and the appropriate number of elements out of R_i. Again this procedure has to be repeated at most three times.

Altogether we need besides the queries for the corresponding OR(k)-strategies only $O(k \log k)$ additional queries. This is small enough if $\log k = O(\log(n/k))$ and, therefore, if $k = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$.

We have proved the theorem, since at least one of the conditions $k = \Omega(n^{1/2}/\log^{1/2} n)$ and $k = O(n^{1-\varepsilon})$ is always fulfilled. □

References

Ahlswede, R. and Wegener, I. (1987). *Search Problems.* Wiley.

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

Angluin, D., Hellerstein, L., and Karpinski, M. (1993). *Learning read-once formulas with queries.* Journal of the ACM 40, 185–210.

Bshouty, N. H. and Hellerstein, L. (1996). *Attribute-efficient learning in query and mistake-bound models.* Proc. of the 9th Conf. on Computational Learning Theory COLT '96, 235–243.

Friedman, J. (1984). *Constructing $O(n \log n)$ size monotone forumlae for the k-th elementary symmetric polynomial of n Boolean variables.* Proc. 25th Symp. on Foundations of Computer Science, 506–515.

Håstad, J., Wegener, I., Wurm, N., and Yi, S. (1994). *Optimal depth, very small size circuits for symmetric functions in AC^0.* Information and Computation 108, 200–211.

Heiman, R. and Wigderson, A. (1991). *Randomized vs. deterministic decision tree complexity.* Computational Complexity 1, 311-329.

Hofmeister, T. (1996). *Personal communication.*

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

Mehlhorn, K. (1982). *On the program size of perfect and universal hash functions.* Proc. 23rd Symp. on Foundations of Computer Science, 170–175.

Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms.* Cambridge University Press.

Naor, M., Schulman, L., and Srinivasan, A. (1995). *Splitters and near-optimal decomposition.* Proc. 36th Symp. on Foundations of Computer Science, 182–191.

Picard, C. (1965). *Théorie des Questionnaires.* Gauthier-Villars, Paris.

Ragde, P. and Widgerson, A. (1991). *Linear-size constant-depth polylog-threshold circuits.* Information Processing Letters 39, 143–146.

Saks, M. and Widgerson, A. (1986). *Probabilistic Boolean decision trees and the complexity of evaluating game trees.* Proc. of 27th Symp. on Foundations of Computer Science, 29–38.

Wegener, I. (1987). *The Complexity of Boolean Functions.* Wiley-Teubner.