Chapter 5

Optimal Attribute-Efficient Learning of Disjunction, Parity, and Threshold Functions

5.1 Basic results

Since the queries have two possible answers, $\lceil \log |F| \rceil$ queries are necessary for the class F. For the classes OR and PAR this leads to the lower bound n which is optimal, since these classes can be learned by asking $E_1, E_2, \ldots, E_n$. Theorem 5.1 contains some other information theoretic lower bounds.

Theorem 5.1 The following lower bounds hold:

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

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

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

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

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

different functions. The number of different functions equals $n2^{n-1} + 2$ and $\lceil \log(n2^{n-1} + 2) \rceil = \left\lceil \log 2^{n-1} \left(n + 2^{-(n-2)} \right) \right\rceil = n - 1 + \lceil \log(n+1) \rceil$. ∎

5.2 Disjunctions

We present two learning algorithms for the class $\mathsf{OR}(k)$. The second one needs less queries but the first one does not use the knowledge of k. In Section 5.4 we investigate the learning of

$\mathsf{THR}(k)$. Since the threshold value t is unknown, we obtain the subproblem of learning $\mathsf{OR}(l)$ and $\mathsf{AND}(m)$ for unknown l and m. Then we apply the algorithm presented in the proof of Theorem 5.2.

Theorem 5.2 There exists a learning algorithm which learns $\mathsf{OR}(k)$ with $2k\,(\log \frac{n}{k} + 2)$ queries without knowing k (if $k = 0$, the algorithm needs one query).

Proof. We start with the query $\{1, \ldots, n\}$. Each winner (a query with answer 1) is 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 number of queries in the i-th phase can be bounded by $\min\{2^{i-1}, 2k\}$ and after $\lceil \log n \rceil + 1$ phases all query sets have size 1. Hence, the total number of queries is bounded by

$$\begin{aligned} 2^0 + \cdots + 2^{\lceil \log k \rceil} + (\lceil \log n \rceil - \lceil \log k \rceil)2k &\leq 2 \cdot 2^{\lceil \log k \rceil} - 2k\lceil \log k \rceil + 2k(\log n + 1) \\ &\leq 2k - 2k \log k + 2k(\log n + 1) \\ &= 2k \left(\log \frac{n}{k} + 2 \right). \end{aligned}$$

∎

Theorem 5.3 The class $\mathsf{OR}(k)$ can be learned with at most $\min\{n - 1, k\lceil \log \frac{n}{k} \rceil + 2k - 2\}$ queries.

Proof. The upper bound $n-1$ follows from the strategy to ask $E_1, ..., E_{n-1}$. For the other upper bound we partition the set of n elements to k sets of size at most $\lceil \frac{n}{k} \rceil$. We ask these k sets as queries. We assume that a set S is a winner. Then, we can find an essential element x by binary search with at most $\lceil \log \lceil \frac{n}{k} \rceil \rceil = \lceil \log \frac{n}{k} \rceil$ queries. 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 $\lceil \log \frac{n}{k} \rceil$ queries. Moreover, we ask k queries in the beginning. For each essential element x we have found we ask an additional query (the query $S - \{x\}$ above). We can save this additional query 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. Thus, we can determine whether the corresponding set $S - \{x\}$ is a winner without making a query. ∎

The second upper bound beats the trivial upper bound $n - 1$ only if $k = n/4$ or $k \leq n/5$. If $k = o(n)$, the upper bound in Theorem 5.3 is $k \log \frac{n}{k} + o(k \log \frac{n}{k})$ while the lower bound is larger than $k \log \frac{n}{k}$. Hence, our algorithm is almost optimal. Moreover, the upper bound of $\mathsf{OR}(k)$ given by the algorithm in Theorem 5.3 is asymptotically optimal if $0 \leq k < \frac{n}{2}$. However, the lower bound given in Theorem 5.1 is much smaller than the upper bound for large k. We prove a better lower bound, and show that the upper bound of $\mathsf{OR}(k)$ given by the algorithm in Theorem 5.3 is optimal if $\frac{n}{2} \leq k \leq n - 1$.

Theorem 5.4 If $\frac{n}{2} \leq k \leq n - 1$, $n - 1$ queries are necessary for $\mathsf{OR}(k)$.

Proof. The claim is proved with an adversary argument. An adversary answers the queries of the learner consistently and tries to force the learner to ask many queries. It is our aim to describe an adversary which can force the learner to ask $n - 1$ queries. In order to simplify the analysis the adversary gives sometimes more information to the learner. If the learner nevertheless has to ask $n - 1$ queries, we are done.

First we consider the class $\mathsf{OR}(n-1)$. Query sets of size larger than 1 are useless, since the answer 1 is known in advance. Hence, only E_is are used as queries and the adversary may answer the first $n-2$ queries by 1. Then the learner still has to ask a query.

We restrict ourselves w. l. o. g. to strategies which do not include identified elements (elements which are known to be essential or nonessential) 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 and at most $n-k-1$ query sets of size 1 are asked, the adversary answers 0 for the query sets of size 1 and answers 1 for larger query sets. The answers 1 are allowed, since $k \geq \frac{n}{2}$. There are two cases to consider. In the first case the learner has asked $n-k-1$ query sets of size 1. Then the learner has obtained the knowledge of $n-k-1$ nonessential elements by $n-k-1$ queries. The answers to these queries do not include more information. Hence, the learner is faced with the problem $\mathsf{OR}(k)$ on $k+1$ variables. As we have shown, the learner needs k additional queries for this purpose and we are done.

In the second case the learner asks a query A which is not disjoint from a previous query B. By our assumptions $|B| \geq 2$. The adversary answers the query by 1. Furthermore, the adversary gives the following information for free. He chooses an element $i \in A \cap B$ and an element $j \in B - \{i\}$. Then he tells the learner that i is essential and j is nonessential. This is consistent with the previous answers. Now the answers to the queries A and B do not contain further information. The learner has obtained the knowledge of one essential element and one nonessential element by 2 queries. Hence, the learner is faced with the problem $\mathsf{OR}(k-1)$ on $n-2$ variables. For this problem we prove with the same strategy that $n-3$ queries are necessary. ∎

5.3 Parity functions

For small k a small number of queries is sufficient to learn $\mathsf{OR}(k)$ while the maximum number of $n-1$ queries is necessary to learn $\mathsf{OR}(n-k)$. For parity functions we prove that the complexity of learning $\mathsf{PAR}(k)$ is equal to the complexity of learning $\mathsf{PAR}(n-k)$. What is the reason for this difference?

Parity (or mod 2 denoted by $\oplus$) 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.

Proposition 5.5 For given n, the complexity of learning $\mathsf{PAR}(k)$ is equal to the complexity of learning $\mathsf{PAR}(n-k)$.

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 $\mathsf{PAR}(k)$ and $\mathsf{PAR}(n-k)$. 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 $a_1(S) \oplus a_2(S) \equiv |S| \mod 2$. Hence, we can compute $a_1(S)$ from $a_2(S)$ and vice versa. A strategy for $\mathsf{PAR}(k)$ which identifies g_1 identifies with the corresponding answers g_2, if it is used for $\mathsf{PAR}(n-k)$. ∎

Because of Proposition 5.5 we assume without loss of generality that $k \leq \frac{n}{2}$. 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 determine whether a subset contains essential elements when the answer to the query is zero. If we find 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. Then we can apply the binary search technique to both sets. We get good upper bounds (see Theorem 5.1) for $k \in \{1, 2, 3\}$.

Proposition 5.6 (i) $\mathsf{PAR}(1)$ can be learned with $\lceil \log n \rceil$ queries.

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

(iii) $\mathsf{PAR}(3)$ 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 $\lceil \frac{n}{2} \rceil$ elements. The third result is obvious; we find the first essential element with binary search, and apply the second result (even for a set of $n - 1$ elements). We save one query, since the first query can be chosen as 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? For this purpose we may use (n, k, r, α)-splitters defined as follows (see Naor, Schulman, and Srinivasan [NSS95]):

Definition 5.7 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 5.8 An (n, k, r, α)-splitter of size s implies that $\mathsf{PAR}(k)$ 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. ∎

We review the applications of splitters in learning theory. Friedman constructs splitters for constant k leading to $O(\log n)$ learning algorithms for $\mathsf{PAR}(k)$ and constant k [Fri84]. Ragde and Widgerson [RW91] (based on [Meh82]) and Håstad, Wegener, Wurm, and Yi [HWWY94] use splitters for the construction of Boolean circuits. The deterministic construction of splitters by Bshouty and Hellerstein [BH96] is a rediscovery of the construction in [HWWY94]. These splitters lead by Lemma 5.8 to the existence of learning algorithms for $\mathsf{PAR}(k)$ with $O(k^3 \log n)$ queries and the efficient construction of learning algorithms with $O(k^4 \log n)$ queries [Hof96].

Using special properties of the parity function we improve this upper bound. 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". The reader not familiar with randomized algorithms may look at [MR95] for an introduction. The use of randomized algorithms in the context of query complexity has been proved in [SW86] and [HW91] for randomized decision trees.

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

Proof. The randomized algorithm repeats the following steps until all essential elements have be found: Without loss of generality, we assume that the algorithm has k' unidentified essential elements in a set of size n' with $0 < k' < n'$. First, the algorithm chooses at random and independently r elements, where $r = \left\lceil \frac{n'}{k'} \right\rceil$. The choices are made with replacement, i.e., the algorithm may choose an element more than once. Next the algorithm collects the chosen elements into a query set (of size at most r), and makes a query. Then, the query set contains exactly one essential element with probability at least

$$\binom{r}{1} \frac{k'}{n'} \left(1 - \frac{k'}{n'}\right)^{r-1} = \left\lceil \frac{n'}{k'} \right\rceil \frac{k'}{n'} \left(1 - \frac{k'}{n'}\right)^{\left\lceil \frac{n'}{k'} \right\rceil - 1} \geq \left(1 - \frac{k'}{n'}\right)^{\frac{n'}{k'}} > \frac{1}{e},$$

where e is the Eulerian constant $2.718\cdots$. Hence, the probability of success to find a set containing an odd number of essential elements is bounded below by $\frac{1}{3}$. 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 at most

$$\begin{aligned} \sum_{1 \leq m \leq k} \left\lceil \log \frac{n-k+m}{m} \right\rceil &\leq \sum_{1 \leq m \leq k} \log \frac{n}{m} + O(k) = k \log n - \log(k!) + O(k) \\ &= k \log n - (k \log k - O(k)) + O(k) = k \log \frac{n}{k} + O(k) \end{aligned}$$

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 $\frac{1}{3}$. We need k successes and may perform $O(k \log \frac{n}{k})$ 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 and lead to a derandomization of the algorithm.

Proposition 5.10 The class $\mathsf{PAR}(k)$ can be learned with $\lceil \log \binom{n}{k} \rceil + k \lceil \log n \rceil + 2$ queries.

Proof. If $k = \Omega(n)$, the result is obvious, since $n-1$ queries of E_is suffice. If $k \leq \frac{n}{3}$,

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

and

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

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

The probability that all random sets are bad for S, i. e. $S \cap R_1$, ..., $S \cap R_m$ 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 $\frac{1}{2}$. Hence, there exist m query sets $T_1, \ldots, T_m$ such that for each $S \subseteq \{1, \ldots, n\}$ with $1 \leq |S| \leq k$ there exists a set T_j such that $|S \cap T_j|$ is odd.

The sets $T_1, \ldots, T_m$ are used as query sets. Let S be the unknown set of essential elements. Then $|S \cap T_{i(1)}|$ 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 $|S_1 \cap T_{i(2)}|$ 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 we 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 $\lceil \log \binom{n}{k} \rceil$ 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 $O\left(k \log \frac{n}{k}\right)$. If $k = O\left(n^{1-\varepsilon}\right)$, nothing has to be done. Hence, we have to consider functions like $k = \frac{n}{\log^2 n}$ or $k = \frac{n}{\log \log n}$.

Theorem 5.11 The class $\mathsf{PAR}(k)$ can be learned with $O\left(k \log \frac{n}{k}\right)$ queries.

Proof. If $k \leq n/2$ and $k = \Omega(n)$, the proposed upper bound is $O(n)$ and we are done by using the trivial strategy asking $E_1, \ldots, E_{n-1}$. Hence, we can assume that $k = \frac{n}{h(n)}$ for some function $h(n)$ tending to ∞. The first idea is to replace the random sets in the proof of Proposition 5.10 by random sets of size $\frac{n}{k} = h(n)$. For sets S with $k' = o(k)$ elements the intersection of S and such a random set is empty with large probability. Therefore, we work with random sets of size $\left(\frac{n}{k}\right)^2 = h(n)^2$. The cost for each binary search is increased by a factor 2 only. For sets S where $|S| \geq \frac{n}{h(n)^2}$ the probability of success for a random set is at least $\frac{1}{3}$. Hence, with $O\left(\log \binom{n}{k}\right)$ sets we can identify all but $\frac{n}{h(n)^2}$ essential elements.

In the second phase we work with sets of size $h(n)^4$ and can identify all but $\frac{n}{h(n)^4}$ 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

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

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

The upper bound of Theorem 5.11 is asymptotically optimal for $k \leq n/2$, since it meets the lower bound of Theorem 5.1. Applying Proposition 5.5 we have proved that the complexity of learning the concept classes $\mathsf{PAR}(k)$ is only by a constant factor larger than the simple information theoretic lower bound.

Remark. If one is interested in *applying* learning algorithms, one has to notice that in Proposition 5.10 and Theorem 5.11 only the existence of learning algorithms with the proposed bounds has been proved. For most applications the simple randomized error-free learning algorithm described in the proof of Theorem 5.9 is good enough. The algorithms proving all the other bounds in this paper are explicitly described and can be applied without further considerations.

5.4 Threshold functions

Theorem 5.12 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 $\{1,\ldots,m\}$. Since threshold functions are monotone, we can find with binary search and $\lceil\log(n+2)\rceil$ queries the minimal $s \in \{0,\ldots,n\}$ such that the answer to the query $\{1,\ldots,s\}$ 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 $s \in \{1,\ldots,n\}$, we know that s is an essential element. The query $\{1,\ldots,s\}-\{j\}$ decides for $j \in \{1,\ldots,s-1\}$ whether j is essential and the query $\{1,\ldots,s-1,j\}$ does the same for $j \in \{s+1,\ldots,n\}$. These are $n-1$ additional queries. ∎

Theorem 5.13 Let $0 \leq t \leq k \leq n$. The class $\mathsf{THR}_t(k)$ can be learned with at most $(k-1)\log\frac{n-1}{k-1} + 3k - 3 + \lceil\log(n-k+1)\rceil$ queries.

Proof. The first phase is the same as in the proof of Theorem 5.12. 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 $\{1,\ldots,s\}$ leads to the answer 1 and the query $\{1,\ldots,s-1\}$ leads to the answer 0. Hence, we know that $\{1,\ldots,s-1\}$ contains exactly $t-1$ essential elements and $\{s+1,\ldots,n\}$ contains the remaining $k-t$ essential elements. Here, we know that $t \leq s \leq n-k+t$. Thus, the step finding s requires at most $\lceil\log(n-k+1)\rceil$ queries.

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

In Section 5.1 we have seen that the complexity of learning AND-classes is the same as for the corresponding OR-classes. By Theorem 5.3 the number of queries of the second and third phases is bounded by

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

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

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

This inequality holds, since $0 \leq a,\ b \leq 1$, see, e.g., [AW87, Lemma 5.3, p.20]. ∎

Theorem 5.14 The class $\mathsf{THR}(k)$ can be learned with at most $2(k-1)\log \frac{n-1}{k-1} + 4k - 4 + \lceil \log(n+2) \rceil$ queries.

Proof. The first phase of the algorithm in the proof of Theorem 5.12 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. Hence, we use the algorithm of Theorem 5.2. Then, the number of queries for $\mathsf{THR}(k)$ is bounded by

$$\lceil \log(n+2) \rceil + 2(k-t)\log \frac{n-s}{k-t} + 4(k-t) + 2(t-1)\log \frac{s-1}{t-1} + 4(t-1).$$

Now we can use the same estimations as in the proof of Theorem 5.13. ∎

We have the following corollary by simple calculations together with the lower bounds in Section 5.1:

Corollary 5.15 The upper bound of THR given in the algorithm in Theorem 5.12 is optimal if $n+1$ is not a power of 2. Otherwise, we miss the lower bound by 1. The upper bound of $\mathsf{THR}_t(k)$ given in the algorithm in Theorem 5.13 is asymptotically optimal within a constant factor if $k \leq \frac{n}{2}$. The upper bound of $\mathsf{THR}(k)$ given in the algorithm in Theorem 5.14 is asymptotically optimal within a constant factor if $k \leq \frac{n}{2}$.

For the classes $\mathsf{THR}_t(k)$ and $\mathsf{THR}(k)$, their upper bounds are asymptotically optimal for $k \leq \frac{n}{2}$. 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 $k > \frac{n}{2}$? We only answer this question in special cases. We first observe that we get $\mathsf{OR}(k)$ from $\mathsf{THR}_t(k)$ if $t = 1$. We know that for this case we need $n-1$ queries, if $k \geq \frac{n}{2}$. Moreover, the cases $t = 1$ and $t = k$ are equivalent. This observation can be generalized.

Proposition 5.16 For given n, the complexity of learning $\mathsf{THR}_t(k)$ is equal to the complexity of learning $\mathsf{THR}_{k-t+1}(k)$.

Proof. Let S be the set of essential elements for the problem $\mathsf{THR}_t(k)$. 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 $\mathsf{THR}_{k-t+1}(k)$, 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 $\mathsf{THR}_t(k)$ by the query Q' for $\mathsf{THR}_{k-t+1}(k)$ and vice versa. ∎

We are still left with a variety of parameter choices. Because of Proposition 5.16 we may restrict ourselves to the case $t \leq n/2$. If one uses threshold gates in Boolean circuits, the gates become more powerful if t is close to $n/2$. In particular, for $t = 1$ we get the "simple" OR-function and for $t = n/2$ the "complex" MAJORITY-function. Hence, one might believe that threshold functions with small t are easier to learn than threshold functions with t close to $n/2$. But the following theorems prove that this is not true. For the case $k = n-1$ it is very hard to learn the OR-function ($n-1$ queries are necessary) while it is easy to learn the MAJORITY-function ($\lceil \log n \rceil$ queries are sufficient). More precisely the learning complexity decreases, if $t \leq n/2$, with t. In Theorem 5.18 we present an asymptotically optimal learning algorithm (for the lower bound see Theorem 5.1 (i)) to learn the MAJORITY-function. We leave it to the reader to generalize this result to other threshold values.

Theorem 5.17 Let $t \leq \frac{n}{2}$. To learn the class $\mathsf{THR}_t(n-1)$

- $\max\{\lceil \log n \rceil, \left\lceil \frac{n-1}{t} \right\rceil\}$ queries are necessary and
- $\lceil \frac{n}{t} \rceil + \lceil \log t \rceil - 1$ queries are sufficient.

By Proposition 5.16 we obtain similar results for the case $t > \frac{n}{2}$.

Proof. The lower bound $\lceil \log n \rceil$ is the simple information theoretical lower bound. For the lower bound $\left\lceil \frac{n-1}{t} \right\rceil$ 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 $\left\lceil \frac{n-1}{t} \right\rceil - 1$ queries. Then there are still at least 2 candidates for the nonessential element and another query is necessary.

For the upper bound we construct $\lceil \frac{n}{t} \rceil$ 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 nonessential element. At least one of the queries is good for our purposes. Hence, it is sufficient to ask $\lceil \frac{n}{t} \rceil - 1$ queries. Since $t \leq \frac{n}{2}$, 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. ∎

Theorem 5.18 The class $\mathsf{THR}^n_{\frac{n}{2}}(n-k)$ can be learned with $O\left(k \log \frac{n}{k}\right)$ queries.

Proof. If $k > \frac{n}{2}$, nothing has to be done and 0 queries are sufficient. If $k \leq \frac{n}{2}$ and $k = \Theta(n)$, the upper bound follows from Theorem 5.12. Hence, we may assume that $k \leq \frac{n}{8}$. Moreover, we assume that k divides n. Otherwise, we may add less than k elements known to be nonessential. Additional nonessential elements have no influence on the answer to queries and can therefore be added. This does not hold for additional essential elements.

We present two algorithms. The first one meets the upper bound $O(k \log \frac{n}{k})$, if $k = \Omega\left(\sqrt{\frac{n}{\log n}}\right)$, and the second one, if $k = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$. Since each k fulfills at least one of the conditions, the two algorithms are sufficient to prove the theorem.

The number of nonessential elements is smaller than the number of essential elements. The main idea is to search for the k nonessential elements. The set $\{1, \ldots, n\}$ of elements is partitioned to $B_1 = \{1, \ldots, \lceil \frac{n}{3} \rceil\}$, $B_2 = \{\lceil \frac{n}{3} \rceil + 1, \ldots, \lceil \frac{2n}{3} \rceil\}$, and $B_3 = \{\lceil \frac{2n}{3} \rceil + 1, \ldots, n\}$. We only

describe how to find the at most k nonessential elements in B_3. The same approach can be used for B_1 and B_2. In order to apply OR-strategies to search for nonessential elements we construct so-called *dummy* sets D_j where $D_1 \supseteq \cdots \supseteq D_m$ for some m and D_j contains exactly $\frac{n}{2} - j$ essential elements. Let S contain $j \leq m$ elements and let S be disjoint from D_1. The query $S \cup D_j$ is answered 0 iff S contains at least one nonessential element. This is equivalent to an OR-query S with respect to nonessential elements. We have to be careful, since the OR-strategy only can work with queries whose size is bounded by m.

Both algorithms determine (as in the proof of Theorem 5.12) the minimal s such that $S^* = \{1, \ldots, s\}$ contains $\frac{n}{2}$ essential elements. Since $\frac{n}{2} \leq s \leq \frac{n}{2} + k$, $\lceil \log(k+1) \rceil$ queries are sufficient for this purpose.

The first algorithm then asks the queries $S^* - \{i\}$, $1 \leq i \leq k + \frac{n}{k}$. The query $S^* - \{i\}$ is answered by 0 iff the element i is essential. Since the number of nonessential elements equals k, we determine at least $\frac{n}{k}$ essential elements $e_1, \ldots, e_{\frac{n}{k}} \in \{1, \ldots, k + \frac{n}{k}\}$. The set $D_j := S^* - \{e_1, \ldots, e_j\}$ contains $\frac{n}{2} - j$ essential elements, $D_j \subseteq D_{j-1}$ and $D_j \subseteq S^* \subseteq B_1 \cup B_2$, since $k \leq \frac{n}{8}$. Now we determine the nonessential elements of B_3 with an OR-strategy starting with a partition of B_3 into $\lceil \frac{k}{3} \rceil$ sets of size bounded by $\frac{n}{k}$. Each query which is a winner, is partitioned to two queries of almost the same size. The number of queries to determine the nonessential elements in B_3 is bounded by $\lceil \log(k+1) \rceil + k + \frac{n}{k}$ for the construction of the dummy sets and $O(k \log \frac{n}{k})$ for the OR-strategy. Since $k = \Omega\left(\sqrt{\frac{n}{\log n}}\right)$, $\frac{n}{k} = O(k \log \frac{n}{k})$ and we are done in this case.

The second algorithm chooses some l where $s < l \leq \lceil \frac{2n}{3} \rceil + 1$ such that the elements $l, \ldots, n$ can be partitioned to $k+1$ sets $R_1, \ldots, R_{k+1}$ of the same size $r \leq \frac{n}{k}$. Since the number of nonessential elements is k, there exists one set R_i consisting of r essential elements. Later the algorithm will determine such an i. We try to find a set $D^* \subseteq S^*$ with exactly $\frac{n}{2} - r$ essential elements. With D^* and R_i it is easy to fulfill our task. Let $R_i = \{e_1, \ldots, e_r\}$, then $D_j := D^* \cup \{e_1, \ldots e_{r-j}\}$, $1 \leq j \leq r$, can be used as dummy sets. The elements of R_i all are essential and we can determine the nonessential elements in the union of all R_h, $h \neq i$, with these dummy sets and an OR-strategy asking queries whose size is bounded by r. This again can be done with $O(k \log \frac{n}{k})$ queries. We still have to describe how we can find the dummy sets.

We consider $C_0 = S^* - \{1, \ldots, r\}, \ldots, C_k = S^* - \{1, \ldots, r+k\}$ as candidates for D^*. We know that C_0 contains at least $\frac{n}{2} - r$ essential elements and that C_k contains at most $\frac{n}{2} - r$ essential elements. Hence, one of the candidates can serve as D^*. Let us assume that we ask the queries $R_1 \cup C_j, \ldots, R_{k+1} \cup C_j$. If all queries are answered by 0, even the union of R_i (with r essential elements) and C_j does not contain $\frac{n}{2}$ essential elements. Hence, C_j contains less than $\frac{n}{2} - r$ essential elements. If at least one query is answered by 1, then C_j contains at least $\frac{n}{2} - r$ essential elements. Hence, with binary search on $j \in \{0, \ldots, k\}$ and altogether $O(k \log k)$ queries we find the largest j such that one of the queries $R_1 \cup C_j, \ldots R_{k+1} \cup C_j$ is answered by 1. This implies that C_j contains at least $\frac{n}{2} - r$ essential elements while C_{j+1} contains less than $\frac{n}{2} - r$ essential elements. Since $C_j \supseteq C_{j+1}$ and $|C_j - C_{j+1}| = 1$, C_j contains exactly $\frac{n}{2} - r$ essential elements and we choose $D^* = C_j$. Moreover, let R_i be one of the sets such that the query $R_i \cup D^*$ has been answered by 1, then all r elements of R_i are essential. Hence, we have found D^* and R_i. The number of additional queries is $O(k \log k) = O(k \log \frac{n}{k})$, since $k = O(n^{1-\varepsilon})$. ∎