Query-efficient computation in property testing and learning theory

García Soriano, D.

Citation for published version (APA):
Chapter 7

Learning parities

The content of this chapter is based on the paper


7.1 Overview of learning theory

Just as we studied the problem of isomorphism to parities under the property testing viewpoint in Chapters 2 and 3, we consider now the similar problem from the learning perspective (in the noiseless setting). We assume some familiarity with computational learning theory; refer to the book of Kearns and Vazirani [KV94] for an easily readable introduction to the field. Nevertheless, for the benefit of the reader we give here a condensed and somewhat informal account.

7.1.1 Mistake-bounded learning and equivalence queries

In learning one is interested in a certain concept class \(C\) of objects to which the learning algorithm applies. Here we will always take \(C\) to be a subset of boolean functions. The mistake-bound model of learning is an online model introduced by Littlestone in [Lit88]. In this setting learning proceeds in rounds. A target function \(f \in C\) is selected arbitrarily by the “teacher” and is unknown to the “learner”. In each round the teacher provides an unlabelled example \(x \in \{0,1\}^*\), and the learner must predict the value \(f(x)\). Then the learner is told the correct value of \(f(x)\), according to which it can update its current hypothesis, i.e., the process whereby the future predictions will be made. The mistake bound of the learner, with respect to a target function \(f\), is the worst-case number of mistakes that it makes over all (arbitrary, possibly infinite) sequences of examples. The mistake bound on a concept class \(C\) is the maximum of the mistake bounds taken over all possible target functions \(f \in C\).
The discussion so far ignores the running time of the learner. In order to
quantify the behaviour of learning algorithms with respect to the input size, we
need to consider a sequence of concept classes \( C_n, n \in \mathbb{N} \), one for each input size
\( n \) (the value of \( n \) being known by the learner). In this chapter we will always
assume \( C_n \) is a collection of boolean functions on \( \{0, 1\}^n \).

7.1.1. Definition. We say that the class \( C = \bigcup_{n \in \mathbb{N}} C_n \) is learnable with mistake
bound \( M = M(n) \) and running time per round \( t = t(n) \) if there is a learner \( L \) that
on each example \( x \in \{0, 1\}^n \) runs in time \( t(n) \) and outputs its guess for \( f(x) \), and
has mistake bound at most \( M(n) \) on \( C_n \).

We say that \( C \) is efficiently learnable with mistake bound \( M \) if in addition
\( t(n) = \text{poly}(n) \).

It is easily seen that, up to one additional query, this model coincides with
that of Angluin \cite{Angluin88} on learning with equivalence queries.

Note that the learner may never know if its current hypothesis is a good
approximation to \( f \) in some sense (unless \( M \) mistakes have already been made).
For instance, suppose that the teacher gives the same example over and over again.
For this particular sequence of examples, any non-trivial learner will make one
mistake at most, but usually this will not allow us to learn anything about how \( f \)
behaves on other inputs.

7.1.2 PAC learning

The Probably Approximately Correct (PAC) model was introduced in the seminal
paper of Valiant \cite{Valiant84} and has since attracted a lot of attention from researchers
in learning theory. In spite of being the first model defined it is a bit more
complicated to describe. Here we additionally need to refer to a hypothesis class \( H \)
of boolean functions; it represents the possible hypothesizes the learning algorithm
may output. We assume that \( H \supseteq C \), but in general \( H \) does not need to coincide
with \( C \), and the choice of \( H \) has a significant impact on the learning complexity.
(When \( H = C \), the learner is called proper.)

The PAC model is a model for “batch” learning: the learner trains on some
fixed dataset, and with this information it constructs a hypothesis that is hopefully
“good”. The dataset consists of labelled samples of some concept \( f \in C_n \), by which
we mean a pair \( (x, f(x)) \), where \( x \in \{0, 1\}^n \). The input \( x \) in these samples come
from some unknown distribution \( D \) over \( \{0, 1\}^n \). When the learner \( L \) has collected
enough data, it outputs a representation of a hypothesis \( h \in H \) that must, with
high probability, be close to \( f \). (The representation of \( h \) must be polynomial-time
evaluatable: given \( x \), the value of \( h(x) \) must be computable in polynomial time.)

\footnote{Here the learner does not receive a particular input \( x \) to try to guess \( f(x) \); instead, it is
allowed to ask “equivalence queries” whereby it tells the teacher a representation of its guess
\( h \) for the entire function \( f \). The teacher then tells the learner whether the guess was correct
(\( h = f \)), and if not the teacher provides a counterexample, meaning an input \( x \) with \( f(x) \neq h(x) \).}
A confidence parameter $\delta \in (0, 1)$ controls the error probability of the algorithm, whereas an error parameter $\varepsilon$ controls the error of the hypothesis $h$. Since we trained the algorithm with examples from $D$, we can only hope for $h$ to be a good approximation to $f$ for random inputs drawn from $D$. In other words, we want to have $\text{dist}_D(f, h) \leq \varepsilon$ with probability at least $1 - \delta$, where

$$\text{dist}_D(f, h) = \Pr_{x \sim D}[f(x) \neq h(x)].$$

7.1.2. Definition. We say that the class $C$ is PAC-learnable using $H$ if there exists an algorithm $L$ that is provided with two real numbers $\delta, \varepsilon$ and has the following property:

For every concept $f \in C$, every input distribution $D$, and all $0 < \varepsilon, \delta < 1/2$, $L$ outputs a hypothesis concept $h \in H$ that, with probability at least $1 - \delta$, satisfies $\text{dist}_D(f, h) \leq \varepsilon$.

The probability is taken over the random samples and the internal random coin flips of $A$.

In order to define efficient PAC learning, we fix a “reasonable” representation scheme for the elements of $H$ and let $\text{size}(h)$ denote the length of the shortest description of $h \in H$ under this scheme. Note that $\text{size}(f)$ is a lower bound on the running time of a learner when presented with target $f$, because at the end of the day the learner must output a representation of its hypothesis.

7.1.3. Definition. We say that the class $C = \bigcup_{n \in \mathbb{N}} C_n$ is efficiently PAC-learnable using $H$ if there exists a learner $L$ for $C$ that, when learning a target function $f \in C_n$, runs in time $\text{poly}(n, 1/\varepsilon, 1/\delta, \text{size}(f))$; and moreover the hypothesis $h \in H$ output by $L$ can be evaluated on any given input in time polynomial in $n$ and $\text{size}(h)$.

Note that the learning must work under any distribution $D$. Sometimes this requirement is weakened so the tester only needs to work for a specific $D$.

If we disregard the efficiency requirements, there is a tight characterization of the complexity of PAC learning in terms of the so-called Vapnik-Chervonenkis dimension of the class.

7.1.4. Definition. Let $C$ be a concept class of boolean functions on some domain $A$. For a sequence of inputs $S = (x_1, \ldots, x_m) \in A^m$, define

$$\Pi_C(S) = \{(f(x_1), \ldots, f(x_m)) \mid f \in C\}$$

Then $S$ is said to be shattered by $C$ if $\Pi_C(S) = \{0, 1\}^m$, i.e., any assignment of answers to the elements of $S$ is consistent with some $f \in C$. 
(If we agree on an ordering of the input domain $A$, we can speak of sets of inputs instead of sequences, as is usually done.)

7.1.5. **Definition.** The VC dimension of $C$ is the cardinality of the largest set of inputs shattered by $C$, or $\infty$ if there are arbitrarily large shattered sets.

7.1.6. **Theorem (see [KV94, Theorems 3.3 and 3.4]).** Let $C$ be a concept class of VC dimension $d$. Then $C$ can be PAC-learned with sample complexity

$$O \left( \frac{d}{\varepsilon} \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon} \log \frac{1}{\delta} \right)$$

On the other hand, any PAC learner for $C$ must use $\Omega(d/\varepsilon)$ examples in the worst case.

There are standard conversion techniques that can be used to transform any mistake-bounded algorithm into a PAC learning algorithm. These transformations preserve the running time of the mistake-bounded algorithm, and the sample size required by the PAC algorithm is equal to the mistake bound, up to constant factors that depend on its approximation and confidence parameters.

7.1.7. **Theorem (Angluin [Ang88]; Littlestone [Lit89]).** Any algorithm $A$ that learns $C$ in the mistake-bound model with mistake bound $M$ and maximum running time per round $t$ can be converted into an algorithm $A'$ that learns $C$ in the PAC model using a sample set of size $O(\frac{1}{\varepsilon}M + \frac{1}{\varepsilon} \log \frac{1}{\delta})$ and running time $O(\frac{1}{\varepsilon}Mt \log \frac{M}{\delta})$, where $\varepsilon$ and $\delta$ are the approximation and confidence parameters of $A'$.

Let us briefly mention how the PAC learner constructed operates. First, it takes $\Theta(\frac{M}{\varepsilon} + \frac{1}{\varepsilon} \log \frac{1}{\delta})$ random samples, and runs the mistake-bounded learner on them. Without loss of generality, the learner only changes hypotheses after a mistake, so at most $M$ hypotheses are produced. The PAC learner then draws a new set of $O(\frac{1}{\varepsilon} \log \frac{M}{\delta})$ samples, uses it to test the validity of each hypothesis, and selects the best. We omit the proof.

These conversion techniques imply that positive results for mistake-bound learning, in particular those given in this chapter, directly yield corresponding positive results for PAC learning. We mention here that no such conversion is known in the opposite direction. In fact, Blum [Blu94] proved that under widely held assumptions (namely, the existence of one-way functions) the mistake-bound model is strictly harder than the PAC model.
7.1.3 Attribute-efficient learning

The study of attribute-efficient learning was initiated in the same paper introducing the mistake-bound model \[ \text{Lit88} \]. A learning algorithm \( L \) for \( C \) in the mistake-bound model is attribute-efficient if all the foregoing conditions are met and the mistake bound of \( A \) on any \( f \in C \) is bounded by a polynomial in \( \text{size}(f) \), independent of \( n \). Similarly, an algorithm \( A \) for learning \( C \) in the PAC model is attribute-efficient if the sample size required for \( A \) to learn the target function \( f \in C \) is polynomial in \( \text{size}(f), 1/\delta, 1/\varepsilon \).

7.2 General observations

One of the long-standing open questions in both the mistake-bound and the PAC learning models is whether parities can be learned attribute-efficiently in polynomial time \[ \text{Blu96} \]. With each vector \( \tilde{f} \in \{0,1\}^n \) we associate a parity function \( f: \{0,1\}^n \to \{0,1\} \) defined by \( f(x) = \langle \tilde{f}, x \rangle = \sum_{i=1}^n \tilde{f}_i x_i \mod 2 \) for all \( x \in \{0,1\}^n \). As any two parities are \( \frac{1}{2} \)-far apart, learning a parity function with \( \varepsilon < \frac{1}{2} \) can be thought of as identifying the corresponding \( n \)-bit vector \( \tilde{f} \). With a slight abuse of notation, from now on we will denote by \( f \) both the parity function \( f: \{0,1\}^n \to \{0,1\} \) and its corresponding vector \( \tilde{f} \in \{0,1\}^n \); in particular we may for instance write \( |f| \) for the Hamming weight of \( \tilde{f} \). The concept class \( \text{PAR}_{\leq k} \) is defined as the class of all parity functions of Hamming weight at most \( k \). We will also refrain from explicitly mentioning \( n \) throughout the chapter so we will write \( \text{PAR}_{\leq k} \) instead. The description length of any function \( f \in \text{PAR}_{\leq k} \) is \( O(k \log n) \), and thus ideally we would like to have \( \text{poly}(n) \)-time algorithms that learn \( \text{PAR}_{\leq k} \) with a mistake bound (respectively sample size) of \( \text{poly}(k \log n) \). This would correspond to attribute-efficient learning as defined above.

It is well known that, in exponential time, \( \text{PAR}_{\leq k} \) can be learned attribute-efficiently in the mistake-bound model (and hence in the PAC model too). A simple algorithm with mistake bound at most \( k \log n \) is the halving algorithm. It maintains a set \( \mathcal{H} \subseteq \text{PAR}_{\leq k} \) of candidate parity functions, and given an example \( x \), it predicts \( \text{Maj} \{ h(x) \mid h \in \mathcal{H} \} \). Whenever a mistake is made, all “wrong” candidates (of which there are at least \( |\mathcal{H}|/2 \)) are removed from \( \mathcal{H} \). If initially the set \( \mathcal{H} \) was set to be \( \text{PAR}_{\leq k} \), then after at most \( \log |\text{PAR}_{\leq k}| \leq k \log n \) mistakes the function \( f \) is learned. The running time of the halving algorithm is dominated by the time needed to compute the predicate \( \text{Maj} \{ h(x) \mid h \in \mathcal{H} \} \). A naive computation of this predicate requires \( |\text{PAR}_{\leq k}| \geq \binom{n}{k} \) steps, and in fact the running time of all known algorithms that try to compute or approximate this predicate is super-polynomial for any \( k = \omega(1) \).

The question of learning juntas efficiently in the PAC setting under the uniform distribution was raised by Mossel, O’Donnell and Servedio \[ \text{MOS04} \]. They showed that \( k \)-juntas are PAC-learnable under the uniform distribution in time roughly
$n^{\omega+1} \cdot \text{poly}(2^k, n)$, which is approximately $n^{0.704}$ (here $\omega < 2.373$ is the exponent in the time bound for matrix multiplication \cite{CW90, Sto10, Wil11}). As $\text{PAR}_k$ is a subset of the class of all juntas, this applies to parities too and yields, in this particular case, an improvement over the halving algorithm in terms of running time.

On the other hand, if we pay no attention to efficiency issues it is possible to show that the halving algorithm is optimal for parities with regard to the sample complexity/mistake bound (though it is known not to be optimal in general; see \cite{Liu88} for counterexamples). The reason is that any subset of $\text{PAR}_n$ forms a list-decodable code (for more detailed accounts on what this means see, e.g., the surveys of Trevisan \cite{Tre04} and Sudan \cite{Sud01}).

### 7.2.1. Theorem.

Let $C \subseteq \text{PAR}_n$ and $0 < \varepsilon < 1/2$, $\delta < 1/6$. Then any PAC learner for $C$ must draw $\Omega(\log (|C| \cdot \delta(1/2 - \varepsilon)))$ samples, even under the uniform distribution.

In particular, the VC dimension of $\text{PAR}_k$ is $\Theta(\log \binom{n}{k})$, and any mistake-bounded learner for $\text{PAR}_k$ must make $\Omega(\log \binom{n}{k})$ mistakes in the worst case.

**Proof.** For the first part, assume there is a PAC learner for $C$ using $q(\delta)$ samples. With probability $1 - \delta$, its hypothesis $h$ is $\varepsilon$-close to $C$ (although it need not be a linear function). Let $\alpha = 1/2 - \varepsilon$. There is a well known bound of $1/(4\alpha^2)$ on the number of Hadamard codewords (elements of $\text{PAR}_n$) at distance $\leq \varepsilon$ from any given $h$; it can be easily proven by analyzing the Fourier coefficients of $h$ (see \cite{Wol08}). By Occam’s razor (Theorem 2.2 in \cite{KV94}), we can reduce the number of candidates from $O(1/\alpha^2)$ to one with $O(\log(1/(\delta\alpha^2)))$ additional samples and error probability $\delta$. Finally we can test for equality with the single remaining candidate with confidence $\delta$ with $O(\log(1/\delta))$ samples. All in all we get an algorithm that draws $q(\delta) + O(\log(1/(\alpha\delta)))$ samples and with confidence $1 - 3\delta > 1/2$ manages to exactly identify $f \in C$. Yet a straightforward information-theoretic argument shows that the latter task needs at least $\log |C| - 1$ samples, hence the lower bound.

For the “in particular” part, note that $\text{VCdim}(C) \leq \log |C|$ always holds, and the inequality $\text{VCdim}(C) \geq \Omega(\log |\text{PAR}_k|)$ follows from the first part and the upper bound of Theorem 7.1.6 after setting constant values for $\delta, \varepsilon$. Finally recall that PAC learning lower bounds imply mistake-bound learning lower bounds by virtue of Theorem 7.1.7.

On the other hand, with a mistake bound of $n$ (respectively, a sample set of size $O(n)$), parities can be learned straightforwardly in polynomial time: one can check, for each new example, whether it is a linear combination of the previous ones; if it is, we output the appropriate linear combination of the previous answers. At most $n$ mistakes will be made since this is the size of a maximum linearly independent subset of $\{0, 1\}^n$. We will call this the trivial algorithm (see also \cite{Blu96}). Just which tradeoffs are attainable between a learner’s running time and its mistake bound is one of the driving questions we investigate in this chapter.
7.3 Results and related work

Despite the simplicity of these algorithms, no other methods for learning parities in the mistake-bound model were known prior to this work. In particular, it was unknown whether \( \omega(1) \)-parities could be learned in polynomial time with \( o(n) \) mistakes. Our main result (stated next) is the first step in this direction.

7.3.1. Theorem (Buhrman et al. [BGM10, Main result]).

Let \( k, t : \mathbb{N} \to \mathbb{N}^+ \) be two functions satisfying \( k(n) \leq t(n) \leq n^2 \). For every \( n \in \mathbb{N} \) (and the corresponding integers \( k = k(n) \) and \( t = t(n) \)) there is a deterministic algorithm that learns \( \text{PAR} \leq k \) in the mistake-bound model, with mistake bound \( k\left\lceil \frac{n}{t} \right\rceil + \lceil \log \left( \frac{n}{k} \right) \rceil \) and running time per round \( O\left( \left( \frac{n}{k} \right) (kn/t)^2 \right) \).

Let us examine a few interesting values for the parameters in Theorem 7.3.1. For example, putting \( k = \log n / \log \log n \) and \( t = \log n \) yields mistake bound of \( O(n/\log \log n) \) and running time per round \( O(n^{2+o(1)}) \). More generally, it is interesting to find out when \( \text{PAR} \leq k \) can be efficiently learned with \( o(n) \) mistakes. On the other hand, we saw in Section 7.2 (from arguments using the VC dimension) that for \( k = \Omega(n) \) it is impossible to learn \( \text{PAR} \leq k \) with sublinear mistake bound, even disregarding computational efficiency. So we only need to consider the case \( k = o(n) \). Recall that the running time of the halving algorithm is at least \( \binom{n}{k} \), which is super-polynomial for any super-constant \( k \), and is \( 2^{\Omega(k \log n)} \) for any positive \( k = n^{1-\Omega(1)} \). In the following we show that, with appropriate parameters, our main theorem can be used to outperform the halving algorithm. Specifically,

- for any \( k = o(\log n) \), \( \text{PAR} \leq k \) can be learned with \( o(n) \) mistakes in polynomial time;
- for any \( k = o(n) \), \( \text{PAR} \leq k \) can be learned with \( o(n) \) mistakes in time \( \tilde{O}(2^{O(k \log n)}) \).

The two items above are formalized next.

7.3.2. Corollary (Case \( k = O(\log n) \)).

For any \( \omega(1) \leq k = O(\log n) \) and \( c \in \mathbb{N} \), define \( t = t(n) = \lceil \frac{kn^c}{k} \rceil \) (for large enough \( n \)). Then \( \text{PAR} \leq k \) can be learned deterministically with mistake bound \( O(n^{1-c/k}) \) and running time per round \( O(n^{c+2-2c/k}) \). Consequently (see Theorem 7.1.7), \( \text{PAR} \leq k \) can be learned deterministically in the PAC model with \( O(n^{1-c/k}) \) samples and running time \( \tilde{O}(n^{3+c-3c/k}) \).

In particular, if \( k = o(\log n) \) then the mistake bound (sample size) is \( o(n) \).

\(^2\)We assume that the functions \( k(n), t(n) \) are computable in \( O(n^2) \) time.
7.3.3. Corollary (Case \(k = o(n)\)). For any \(\omega(1) \leq k = o(n)\) let \(t = t(n) = o(n)\) be an arbitrary function with growth rate \(\omega(k)\). Then \(\text{PAR}_{\leq k}\) can be learned deterministically with mistake bound \(O(kn/t + k \log \frac{1}{k}) = o(n)\), and total running time \(2^{O(k \log \frac{1}{k} + \log n)}\).

Consequently (see Theorem 7.1.7), \(\text{PAR}_{\leq k}\) can be learned deterministically in the PAC model with \(O(kn/t + k \log \frac{1}{k}) = o(n)\) samples and running time \(2^{O(k \log \frac{1}{k} + \log n)}\).

For example, if \(t = k \log k\) then the running time is \(2^{O(k \log \log k + \log n)}\).

In addition to the corollaries above, observe that Theorem 7.3.1 with \(t = \frac{n}{\log(n/k)}\) gives the same mistake bound as the halving algorithm with slightly better running time. Similarly, we can obtain the features of the trivial algorithm by setting \(k = t = n\).

7.3.1 Learning parities in the PAC model

In the PAC model, Klivans and Servedio \cite{KS06} were the first to show non-trivial algorithms for learning parities with sample sets of sublinear size. (They attribute the second item of the following theorem to Spielman, although it seems to have previously appeared in the literature, e.g., \cite{HB01}.)

7.3.4. Theorem (Klivans & Servedio \cite{KS06}).

1. \(\text{PAR}_{\leq k}\) can be learned in the PAC model with \(O(n^{1-1/k} \log n)\) samples in time \(O(n^4)\).

2. \(\text{PAR}_{\leq k}\) can be learned in the PAC model with \(O(k \log n)\) samples in time \(\tilde{O}(n^{[k/2]})\).

Since our main theorem holds in the harder mistake-bound model, using the standard conversion techniques (Theorem 7.1.7) it also implies results similar to those in Theorem 7.3.4 even with improved parameters. In particular, from Corollary 7.3.2 (with \(c = 1\)) and Corollary 7.3.3 we get the following.

7.3.5. Theorem (Buhrman et al. \cite{BGM10}).

1. \(\text{PAR}_{\leq k}\) can be learned in the PAC model with \(O(n^{1-1/k})\) samples in time \(\tilde{O}(n^{3-3/k} \log n)\).

2. \(\text{PAR}_{\leq k}\) can be learned in the PAC model with \(o(n)\) samples in time \(\approx 2^{O(k \log n)}\).

In the first item, the number of samples required by our algorithm is improved by a factor of \(\log n\), and the running time is improved by a factor of \(n^{3/k}\). As for Item 2, our algorithm requires more than \(O(k \log n)\) samples (although still...
7.4. Proof of the main theorem

The algorithm from Theorem 7.3.1 is based on an idea that was recently used by Alon, Panigrahy and Yekhanin, who gave elegant deterministic approximation algorithms for the Nearest Codeword and Remote Point problems (see [APY09] for details). First we outline the main idea in this algorithm, and then provide its formal description together with the proof.

7.4.1 Informal description of the algorithm

Recall that, in the halving algorithm, a set $\mathcal{H}$ of candidate parity functions is maintained, and given an example $x$, the prediction of the learner is

$$\text{Maj}\ \{h(x) \mid h \in \mathcal{H}\}.$$  

The problem with this method is that for any $k = o(1)$, the initial set $\mathcal{H} = \text{PAR}_{\leq k}$ is of super-polynomial size, and we have no efficient algorithm to compute the majority vote.

In order to overcome this problem, we use a special set of affine spaces that enables a compact representation of (a superset of) the candidate parity functions, while at the same time enabling efficient approximation of their majority vote,
for any example $x$. Specifically, our learning algorithm begins by obtaining a set of affine spaces $N_1, N_2, \cdots \subset \{0, 1\}^n$, at least one of them containing the target parity function $f$. In every step of the learning process, these sets of affine spaces are updated according to the response given by the teacher. The way these updates are performed guarantees:

- the running time is polynomial in $n$ and linear in the number of affine spaces $N_i$;
- after every mistake, some sets $N_i$ get shrunk, so that the quantity $\sum_i |N_i|$ is at least halved (this is ensured by approximating the majority vote);
- the target function $f$ is never removed from any $N_i$.

Since $\sum_i |N_i| \geq |\bigcup_i N_i|$, after at most a logarithmic (in $\sum_i |N_i|$) number of mistakes the target function $f$ is the only element left in $\bigcup_i N_i$, and hence $f$ is learned.

### 7.4.2 Formal description and proof

Fix $n$ and $t = t(n)$. Define $S \subseteq 2^{\{1, \ldots, t\}}$ as $S = \{s \subseteq [t] \mid |s| = k\}$, hence $|S| = \binom{t}{k}$. Let $\pi = C_1, \ldots, C_t$ be an equipartition of $\{e_1, e_2, \ldots, e_n\}$ (the standard basis of $\{0, 1\}^n$) into $t$ parts. For every $s \in S$ we define the linear subspace (over $\mathbb{F}_2^n$) $M_s = \text{span}(U_s)$, where $U_s$ is a set of unit vectors defined as

$$U_s \triangleq \bigcup_{i \in s} C_i.$$ 

That is, $M_s$ consists of all binary vectors whose nonzero entries belong to the parts that are indexed by the elements of $s$. Notice that for every $s \in S$, $M_s$ is a span of at most $k \lceil \frac{n}{t} \rceil$ vectors, and hence

$$|M_s| \leq 2^{k \lceil \frac{n}{t} \rceil}.$$ 

Let $\ell = k \lceil \frac{n}{t} \rceil$. For every affine space $N \subseteq \{0, 1\}^\ell$, $x \in \{0, 1\}^\ell$ and $z \in \{0, 1\}$, we define the affine space $N(x, z) \triangleq \{y \in N \mid \langle y, x \rangle = z \mod 2\}$. Given $x \in \{0, 1\}^\ell$, $z \in \{0, 1\}$ and a representation for $N$ as a system $\text{Lin}^N$ of independent linear equations in triangular form, the corresponding representation of $N(x, z)$ (and the cardinality $|N(x, z)|$) can be computed in time $O(\ell^2)$. This is done by adding $x' = x \sqcup z \in \{0, 1\}^{\ell+1}$ as a row to $\text{Lin}^N$ and performing only one step of the Gaussian elimination procedure to bring the matrix back into triangular form. Notice that this procedure has three possible outcomes:

- (i) $x'$ is inconsistent with $\text{Lin}^N$, and hence $|N(x, z)| = 0$;
- (ii) $x'$ is a linear combination of equations in $\text{Lin}^N$, and hence $|N(x, z)| = |N|$;
(iii) $x'$ is linearly independent of $Lin^N$, and hence $|N(x, z)| = |N|/2$.

### 7.4.1. Proposition.

1. Every $f \in \{0, 1\}^n$ with $|f| \leq k$ is contained in $\bigcup_{s \in S} M_s$;
2. $|\bigcup_{s \in S} M_s| \leq \binom{t}{k} 2^{|n|/t}$.
3. Let $\{N_s \mid s \in S\}$ be a family of affine subspaces of $\{0, 1\}^n$. For any $x \in \{0, 1\}^n$ there exists $z \in \{0, 1\}$ for which $\sum_{s \in S} |N_s(x, z)| \geq \frac{1}{2} \sum_{s \in S} |N_s|$.

**Proof.**

1. This follows from the fact that every set of $k$ unit vectors is contained in the union of some $d \leq k$ subsets $C_{i_1}, \ldots, C_{i_d}$ in the partition $\pi$. Let $s \subseteq [t]$, $|s| = k$ be a set that contains $i_1, \ldots, i_d$. Then $f \in M_s$.
2. $|\bigcup_{s \in S} M_s| \leq \sum_{s \in S} |M_s| \leq \binom{t}{k} 2^{|n|/t}$.
3. This is a consequence of the equality $|N_s(x, 0)| + |N_s(x, 1)| = |N_s|$.

The learner proceeds as follows:

**Initialization:**

Obtain a system of equations describing each of the linear spaces $M_s$ as defined above; and then initialize the affine spaces $N_s = M_s$ for all $s \in S$.

**On example** $x \in \{0, 1\}^n$:

Compute $n_0 = \sum_{s \in S} |N_s(x, 0)|$ and $n_1 = \sum_{s \in S} |N_s(x, 1)|$. If $n_0 \geq n_1$ output 0, else output 1.

**On answer** $l = \langle f, x \rangle$:

Update $N_s := N_s(x, l)$ for each $s \in S$.

It might be helpful for the reader to think that each $M_s$ runs an independent instance of the trivial algorithm of Section 7.3. Each instance assumes that all relevant parity bits of $f$ belong to the corresponding $M_s$, and $N_s$ is the set of candidates (parities consistent with the answers to previous examples) left under this assumption. Some of these candidate sets will vanish as new values of $f$ are learned, but at least one of them contains $f$ and hence will always remain non-empty. The prediction made by the algorithm can be viewed as a weighted majority of all “surviving” instances, where the weight of an instance is proportional to the number of candidates left for it. Thus, whenever a new sample is, when restricted to a set $M_s$, linearly independent of the prior ones,
we “penalize” the $s$th instance by halving its weight; while if the new example is inconsistent we remove the $s$th instance from consideration.

**Proof of Theorem 7.3.1.** First notice that the invariant $f \in \bigcup_{s \in S} N_s$ holds at any stage of the learning algorithm. Initially it holds by Item 1 of Proposition 7.4.1 and every time the algorithm shrinks the sets $N_s$, only elements that are not equal to $f$ are removed.

Since all the subspaces $N_s$ contain vectors of Hamming weight at most $\ell = k \lceil \frac{n}{t} \rceil$, we can treat them as affine subspaces of $\{0,1\}^{\ell}$ by truncating all their irrelevant coordinates. In addition, for any $N_s$, an example $x \in \{0,1\}^n$ can be truncated to the corresponding $\ell$-bit vector by removing all the irrelevant coordinates (with respect to $N_s$). Making this observation, the $O\left(\left( t k \right) \left( \frac{n}{t} \right)^2 \right)$ bound on the running time (per round) of the algorithm now follows from the remarks above on Gaussian elimination and the fact that $|S| \leq \left( t k \right)$.

Finally, we have to show that the number of mistakes that the learner makes is bounded by $k \lceil \frac{n}{t} \rceil + \lceil \log \left( \frac{n}{k} \right) \rceil$. Notice that by the definition of the output value, and by Item 3 of Proposition 7.4.1, every time the learner makes a mistake the quantity $\sum_{s \in S} |N_s|$ reduces by a factor of at least 2. Since at every step $0 < \left| \bigcup_{s \in S} N_s \right| \leq \sum_{s \in S} |N_s|$, and since initially we started with $\sum_{s \in S} |N_s| = \sum_{s \in S} |M_s| \leq \left( t k \right) 2^{k \lceil n/t \rceil}$ (see Item 2 of Proposition 7.4.1), after at most

$$\log \left( \sum_{s \in S} |M_s| \right) \leq k \left[ \frac{n}{t} \right] + \left| \log \left( \frac{t}{k} \right) \right|$$

mistakes the size of $\bigcup_{s \in S} N_s$ will decrease to 1, which by the invariant above will imply that $\bigcup_{s \in S} N_s = \{f\}$, and the learner will no longer make any errors. \hfill \Box

### 7.4.3 Optimality of the system of affine spaces

To recap, we have constructed a set $\mathcal{A}$ of $m = \binom{n}{k}$ affine spaces of dimension $d = \frac{kn}{t} \leq \frac{3n}{m^{1/k}}$ that together “cover” all vectors of weight $\leq k$, in that every such vector belongs to one of the elements of $\mathcal{A}$. The mistake bound we get is $\log \sum_{A \in \mathcal{A}} |A| = \log m + d \leq \log m + 3n/m^{1/k}$. One may ask whether this value can be improved upon by finding a better system of affine spaces. It turns out that such a possibility can be ruled out:

**7.4.2. Proposition.** Let $1 \leq k \leq n/100$ and suppose $\mathcal{A}$ is a collection of $m \leq \left( \frac{n}{100k} \right)^k$ affine spaces over $\{0,1\}^n$ such that every $x \in \{0,1\}^n$ with $|x| \leq k$ belongs to some $A \in \mathcal{A}$. Then

$$\log \left( \sum_{A \in \mathcal{A}} |A| \right) \geq \log m + \frac{n}{3m^{1/k}}.$$
7.4. Proof of the main theorem

7.4.3. Lemma. If $V \in \{0,1\}^n$ is an affine subspace of dimension $d$, then the number of vectors in $V$ of weight at most $k$ is upper bounded by

$$\binom{d}{\leq k} = \binom{d}{0} + \binom{d}{1} + \cdots + \binom{d}{k}. $$

Proof. As $V$ is a $d$-dimensional affine subspace, there is a set $D \subseteq [n], |D| = d$ such that the projection $\pi_D(x)$ of each element $x \in V$ onto the coordinates of $D$ uniquely determines $x$. Consider the set $V_k \triangleq \{x \in V \mid |x| \leq k\}$ and write $\pi_D(V_k) \triangleq \{\pi_D(x) \mid x \in V_k\}$; we have just seen that $|\pi_D(V_k)| = |V_k|$. But no element of $\pi_D(V_k)$ has weight greater than $k$ by construction, so $|\pi_D(V_k)| \leq \sum_{i=0}^{k} \binom{d}{i}$. □

Proof of Proposition 7.4.2. Write $A = \{A_1, \ldots, A_m\}$ and $d_i \triangleq \dim A_i$, so $|A_i| = 2^{d_i}$. In order to cover all vectors of weight at most $k$, simple counting (along with the preceding lemma) tells us that the following inequality must hold:

$$\sum_{i=1}^{m} \binom{d_i}{\leq k} \geq \binom{n}{\leq k}. $$

Suppose there are $r$ subspaces of dimension larger than $2k$ and let $d_1, \ldots, d_r > 2k \geq d_{r+1}, \ldots, d_m$. For the small subspaces we can compute $\sum_{i \leq 2k} \binom{d_i}{\leq k} \leq m4^k$.

Hence

$$\sum_{i=1}^{r} \binom{d_i}{\leq k} \geq \binom{n}{\leq k} - m4^k \geq \binom{n}{\leq k} - \left(\frac{n}{25k}\right)^k$$

by our bound on $m$. Together with the well-known inequality $(a/b)^b \leq \binom{a}{b} \leq (ae/b)^b$, this implies that

$$\sum_{i=1}^{r} d_i^k \geq \left(\frac{n}{e}\right)^k - \left(\frac{n}{25e}\right)^k \geq \left(\frac{n}{3}\right)^k.$$

The function $f : [e^{k-1}, \infty] \to \mathbb{R}^+$ defined by $f(x) = (\log x)^k$ is concave if $k \geq 1$, therefore Jensen’s inequality [CT91] applied to the sequence $\{2^{d_i}\}_{i \in [r]}$ yields

$$f\left(\frac{1}{r} \sum_{i=1}^{r} x_i\right) \geq \frac{1}{r} \sum_{i=1}^{r} f(x_i) \geq \frac{1}{r} \left(\frac{n}{3}\right)^k,$$

from which one obtains (by taking $k$th roots)

$$\log\left(\sum_{d_i > 2k} 2^{d_i}\right) \geq \log r + \frac{n}{3r^{1/k}} \geq \log m + \frac{n}{3m^{1/k}},$$

where the last step made use of the fact that the function $g(r) = \log r + n/(3r^{1/k})$ is decreasing in a range containing $[1, m]$. □
7.5 Analysis of the $\tilde{O}(n^{k/2})$ algorithm

As mentioned previously, the possibility to improve the running time of the halving algorithm to roughly $O(n^{k/2})$ has been noted by several authors [KS06, HB01]. In this section we explain how to extend the ideas to derive a mistake-bounded algorithm.

**Proof of Theorem 7.3.6.** Let $A = \text{PAR}_{\leq [k/2]} \times \text{PAR}_{\leq [k/2]}$. Then $|A| = O(n^{k+1})$. We can associate each element $(p, q) \in A$ with the “parity pair” $p \oplus q$; each parity $r \in \text{PAR}_{\leq k}$ will then correspond to several pairs in $A$, namely those such that $r(x) = p(x) \oplus q(x)$ for all $x \in \{0, 1\}^n$. Thus, we can view $A$ as a multiset of $\leq (k + 1)$-parities (as well as a set of parity pairs). The *answer* of a parity pair $(p, q)$ on $x$ is defined as $p(x) \oplus q(x)$.

We will show that, given any input $x$, we can compute the majority vote of the answers of all parity pairs in $A$ that agree with all previous examples in $O(n^{k/2})$ time, effectively simulating the halving algorithm over the multiset $A$. This implies that the number of mistakes will be bounded by $\log |A| = O(k \log n)$.

In order to compute this majority, it is enough to know how many parity pairs in $A$ are consistent with all the examples seen so far and would output 0 for the new example (and how many of them would output 1). Assume we have been given the examples $x = x_1, x_2, \ldots, x_{m-1} \in \{0, 1\}^{n \times (m-1)}$ with answers $y = y_1, y_2, \ldots, y_{m-1} \in \{0, 1\}^{m-1}$, together with the new example $x_m \in \{0, 1\}^n$, and we are required to output our prediction for $f(x_m)$, where $f$ is the unknown parity function. Let $a \triangleq y_1 y_2 \ldots y_{m-1} 0$ be the $m$-bit vector that contains the answers to all previous $m - 1$ examples and whose last entry is 0 (representing that we are trying to count how many consistent parity pairs would answer 0 for $x_m$). Each parity $p \in \text{PAR}_{\leq [k/2]}$ will give an answer for examples $x_1, \ldots, x_{m-1}, x_m$; let $p(x) \triangleq p(x_1)p(x_2)\ldots p(x_m) \in \{0, 1\}^m$ be their concatenation. Consider the multisets

$$V \triangleq \{p(x) \mid p \in \text{PAR}_{\leq [k/2]}\}$$

and

$$W_a \triangleq \{p(x) \oplus a \mid p \in \text{PAR}_{\leq [k/2]}\}$$

(where $\oplus$ denotes bitwise addition mod 2). Sort the multiset $V \cup W_a$ in, say, lexicographical order, keeping track of whether each vector comes from $V$ or from $W_a$. For each range of (consecutive) equal elements in the sorted sequence $V \cup W_a$ (equal to some vector $c \in \{0, 1\}^m$), count how many of them are from $V$ and how many are from $W_a$; call these numbers $r$ and $s$ respectively. What this means is that there are $\lceil k/2 \rceil$-parities $p_1, p_2, \ldots, p_r, q_1, \ldots, q_s$ such that

$$c = p_1(x) = p_2(x) = \cdots = p_r(x) = q_1(x) \oplus a = q_2(x) \oplus a = \cdots = q_s(x) \oplus a,$$

and $p_1, \ldots, p_r$ are distinct (as are $q_1, \ldots, q_s$).
Thus, there are exactly \( rs \) pairs of parities in \( \text{PAR}_{\leq \lfloor k/2 \rfloor} \) such that \( p(x) \oplus q(x) = a \) and \( p(x) = c \). For each range of equal elements in the sorted sequence \( V \cup W_a \), we will find a possible value of \( c \). Summing \( rs \) over all these ranges we obtain the number of pairs \( (p, q) \in A \) such that \( p(x) \oplus q(x) = a \). We can compute this in linear time by making one pass over the sorted sequence. We can similarly compute the number of parity pairs consistent with previous examples that output 1, and then predict the bit that agrees with the majority of consistent parity pairs.

For the implementation, note that we can go through all \( \text{PAR}_{\leq \lfloor k/2 \rfloor} \) parities and compute their answers on \( x_m \) in \( O \left( \binom{n}{\lfloor k/2 \rfloor} \right) \) time (a naive implementation might give an additional factor of \( n \), but this factor can be avoided with some care, for example by backtracking). Note also that before any example has been given, the sequence \( V \cup W_a \) can be regarded as a multiset of empty vectors, and is thus sorted; and given a new example, if we keep the multiset \( V \cup W_a \) corresponding to our answer from the previous round, we can update the sequence in \( O(|V|) \) time by performing one step of radix sort, since it is already sorted with respect to the first \( m - 1 \) bits and we only need to sort it with respect to the newly computed bit (the answer of the parity to \( x_m \)), which we can consider the most significant one.

Hence, the total running time per round is \( O(|V|) = O \left( \binom{n}{\lfloor k/2 \rfloor} \right) \).

\[ \square \]

### 7.6 Summary

We developed new deterministic algorithms for learning parities in both the mistake-bound and the \( \text{PAC} \) models of learning. For the mistake-bound model we showed the first efficient algorithm that learns \( k \) parities for non-constant \( k \) while making a sublinear number of mistakes.

The mistake bound of our algorithm is still far from the value achieved by the halving algorithm. It remains a major open problem to determine whether parities can be learned attribute-efficiently in polynomial time. The halving algorithm has no known efficient implementation, but if \( \mathsf{P} = \mathsf{NP} \) it can be converted into one that runs in polynomial time, and has approximately the same mistake bound. (This follows from the result of Stockmeyer [Sto83] that if one is provided with access to an \( \mathsf{NP} \) oracle, then it is possible to use a randomized polynomial-time algorithm to approximate, to within a constant factor, the number of solutions to an \( \mathsf{NP} \) predicate.) Two possible avenues of research remain open: either construct an efficient algorithm with improved mistake bound (ideally approaching the bounds of the halving algorithm), or show that the existence of such an algorithm is unlikely by relating it to hardness assumptions from classical complexity theory.