Journal of Machine Learning Research 8 (2007) 1431-1460 Submitted 5/06; Revised 10/06; Published 7/07

Attribute-Efficient and Non-adaptive Learning of Parities and DNF Expressions*

Vitaly Feldman† VITALY@EECS.HARVARD.EDU
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138

Editor: Peter Auer

Abstract

We consider the problems of attribute-efficient PAC learning of two well-studied concept classes: parity functions and DNF expressions over $\{0,1\}^n$. We show that attribute-efficient learning of parities with respect to the uniform distribution is equivalent to decoding high-rate random linear codes from low number of errors, a long-standing open problem in coding theory. This is the first evidence that attribute-efficient learning of a natural PAC learnable concept class can be computationally hard.

An algorithm is said to use membership queries (MQs) *non-adaptively* if the points at which the algorithm asks MQs do not depend on the target concept. Using a simple non-adaptive parity learning algorithm and a modification of Levin's algorithm for locating a weakly-correlated parity due to Bshouty et al. (1999), we give the first non-adaptive and attribute-efficient algorithm for learning DNF with respect to the uniform distribution. Our algorithm runs in time $\tilde{O}(ns^4/\varepsilon)$ and uses $\tilde{O}(s^4 \cdot \log^2 n/\varepsilon)$ non-adaptive MQs, where s is the number of terms in the shortest DNF representation of the target concept. The algorithm improves on the best previous algorithm for learning DNF (of Bshouty et al., 1999) and can also be easily modified to tolerate random persistent classification noise in MQs.

Keywords: attribute-efficient, non-adaptive, membership query, DNF, parity function, random linear code

1. Introduction

The problems of PAC learning parity functions and DNF expressions are among the most fundamental and well-studied problems in machine learning theory. Along with running time efficiency, an important consideration in the design of learning algorithms is their *attribute-efficiency*. A class $\mathcal{C}$ of Boolean functions is said to be *attribute-efficiently learnable* if there is an efficient algorithm which can learn any function $f \in \mathcal{C}$ using a number of examples which is polynomial in the "size" (description length) of the function f to be learned, rather than in n, the number of attributes in the domain over which learning takes place. Attribute-efficiency arises naturally from a ubiquitous practical scenario in which the total number of potentially influential attributes is much larger than the number of relevant attributes (i.e., the attributes on which the concept actually depends), whereas examples are either scarce or expensive to get.

∗. Parts of this work are published in the Proceedings of 18th Annual Conference on Learning Theory, 2005.
†. Supported by grants from the National Science Foundation NSF-CCF-9877049, NSF-CCF-0432037, and NSF-CCF-0427129.

Learning of DNF expressions and attribute-efficient learning of parities from random examples with respect to the uniform distribution are both long-standing challenges in learning theory. The lack of substantial progress on these questions has resulted in attempts to solve them in stronger learning models. The most well-studied such model is one in which a *membership query oracle* is given to the learner in addition to the example oracle. The learning algorithm may query this oracle for a value of the target function at any point of its choice. Jackson (1997) gave the first algorithm that learns DNF from membership queries (MQs) under the uniform distribution and later Bshouty, Jackson, and Tamon (1999) gave a more efficient and attribute-efficient algorithm for learning DNF in the same setting. The first algorithm for attribute-efficient learning of parities using MQs is due to Blum et al. (1995), and their result was later refined by Uehara et al. (1997).

A restricted model of membership queries, which addresses some of the disadvantages of the MQ model, is the model in which MQs are asked non-adaptively. An algorithm is said to use MQs *non-adaptively* if the queries of the algorithm do not depend on the target concept (in our context we will often call it non-adaptive for brevity). In other words, the learning algorithm can be split into two stages. In the first stage, given the learning parameters, the algorithm generates a set S of queries for the membership oracle. In the second stage, given the answers to the queries in S, the algorithm produces a hypothesis (without further access to the oracle). An immediate advantage of this model (over the usual MQ model) is the fact that the queries to the membership oracle can be parallelized. This, for example, is crucial in DNA sequencing and other biological applications where tests are very time-consuming but can be parallelized (Farach et al., 1997; Damaschke, 1998, and references therein). Another advantage of a non-adaptive learner is that the same set of points can be used to learn numerous concepts. This is conjectured to happen in the human brain where a single example can be used to learn several different concepts and hence systems that aim to reproduce the learning abilities of the human brain need to possess this property (Valiant, 1994, 2000, 2006).

As it is detailed later, attribute-efficiency is easy to achieve using a simple technique that relies on adaptive MQs but there is no known general method to convert a learning algorithm to an attribute-efficient one using MQs non-adaptively. It is important to note that in the two practical applications mentioned above, attribute-efficiency is also a major concern. It is therefore natural to ask: which classes can be PAC learned attribute-efficiently by non-adaptive MQs? We refer to this model of learning as *ae.naMQ learning*. This question was first explicitly addressed by Damaschke (1998) who proved that any function of r variables is ae.naMQ learnable when it is represented by the truth table of the function (requiring $r \log n + 2^r$ bits). Later Hofmeister (1999) gave the first ae.naMQ algorithm for learning parities and Guijarro et al. (1999a) gave an algorithm for learning functions of at most $\log n$ variables in the decision tree representation. But the question remains open for numerous other representations used in learning theory.

1.1 Previous Results

Blum et al. (1995) were the first to ask whether parities are learnable attribute-efficiently (in the related *on-line mistake-bound* model). They also presented the first algorithm to learn parity functions attribute-efficiently using MQs. Their algorithm is based on the following approach: first all the relevant attributes are identified and then a simple (not attribute-efficient) algorithm restricted to the relevant variables is used to learn the concept. Since then other algorithms were proposed for attribute-efficient identification of relevant variables (Bshouty and Hellerstein, 1998; Guijarro et al., 1999b). All the algorithms are based on a binary search for a relevant variable given a positive

and a negative example. Binary search and the fact that queries in the second stage depend on the variables identified in the first stage only allows for the construction of adaptive algorithms via this approach. Uehara et al. (1997) gave several algorithms for attribute-efficient learning of parities that again used adaptiveness in an essential way.

Hofmeister gave the first ae.naMQ algorithm for learning parities based on BCH error-correcting codes. When learning the class of parities on at most k variables his algorithm has running time of $O(kn)$ and uses $O(k \log n)$ non-adaptive MQs. While the complexity of this algorithm is asymptotically optimal it is based on the relatively complex Berlekamp-Massey algorithm for creating and decoding BCH codes (Massey, 1969).

Little previous work has been published on attribute-efficient learning of parities from random examples only. Indeed, the first non-trivial result in this direction has only recently been given by Klivans and Servedio (2004). They prove that parity functions on at most k variables are learnable in polynomial time using $O(n^{1-\frac{1}{k}} \log n)$ examples.

1.1.1 LEARNING DNF

Efficient learning of unrestricted DNF formulae under the uniform distribution begins with a famous result by Jackson (1997). The algorithm, while polynomial-time, is somewhat impractical due to the $\tilde{O}(ns^{10}/\varepsilon^{12})$ bound on running time (where s is the number of terms in the target DNF). By substantially improving the key components of Jackson's algorithm, the works of Freund (1992), Bshouty et al. (1999), and Klivans and Servedio (2003) resulted in an algorithm that learns DNF in time $\tilde{O}(ns^6/\varepsilon^2)$ and uses $\tilde{O}(ns^4/\varepsilon^2)$ MQs.[1] This algorithm is non-adaptive, but is also not attribute-efficient. Using the algorithm for identification of relevant variables by Bshouty and Hellerstein mentioned above, Bshouty et al. (1999) gave an attribute-efficient version of their algorithm running in time $\tilde{O}(rs^6/\varepsilon^2 + n/\varepsilon)$ and using $\tilde{O}(rs^4 \log n/\varepsilon^2)$ adaptive MQs, where r is the number of relevant variables.

Bshouty et al. (2003) give an algorithm for learning DNF expressions from examples generated by a random walk on the Boolean hypercube. This model is more passive than non-adaptive MQs but their algorithm is not attribute-efficient as it is an adaptation of the non-attribute-efficient algorithm of Bshouty and Feldman (2002). In fact, it is information-theoretically impossible to learn anything non-trivial attribute-efficiently in this model.

1.2 Our Results

We give a simple and fast randomized algorithm for ae.naMQ learning of parities (Theorem 9) and provide a transformation that converts a non-adaptive parity learning algorithm into an algorithm for finding significant Fourier coefficients of a function while preserving attribute-efficiency and non-adaptiveness (Theorem 13). Using these components we give the first ae.naMQ algorithm for learning DNF expressions with respect to the uniform distribution (Theorem 24). It runs in time $\tilde{O}(ns^4/\varepsilon)$ and uses $\tilde{O}(s^4 \log^2 n/\varepsilon)$ MQs. The algorithm improves on the $\tilde{O}(ns^6/\varepsilon^2)$-time and $\tilde{O}(ns^4/\varepsilon^2)$-query algorithm of Bshouty et al. (1999). In Theorem 28 we also show a simple and general modification that allows the above algorithm to efficiently handle random persistent classification noise in MQs (see Section 2.1 for the formal definition of the noise model). Earlier algorithms for learning DNFs that handled persistent classification noise were based on Jackson's DNF learning algorithm and therefore are substantially less efficient (Jackson et al., 1997; Bshouty and Feldman, 2002).

1. Bshouty et al. claimed sample complexity $\tilde{O}(ns^2/\varepsilon^2)$ but this was in error as explained in Remark 19.

Alongside our ae.naMQ algorithm for learning of parities we establish the equivalence between attribute-efficient learning of parities from random uniform examples and decoding high-rate random linear codes from a low number of errors, a long-standing open problem in coding theory widely believed to be intractable (Theorems 6 and 8). Thus we may consider this equivalence as evidence of the hardness of attribute-efficient learning of parities from random examples only. Previously hardness of attribute-efficient learning results were only known for specially designed concept classes (Decatur et al., 1999; Servedio, 2000).

The connection between attribute-efficient learning of parities by membership queries and linear codes was earlier observed by Hofmeister (1999). His result allows to derive attribute-efficient parity learning algorithms from efficiently decodable linear codes with appropriate parameters. Our result can be seen as an adaptation of this connection to random and uniform examples. The restriction to the uniform distribution allows us to prove the connection in the other direction, giving the above-mentioned negative result for attribute-efficient learning of parities from random examples only.

1.3 Organization

In the next section we describe the models and tools that will be used in this work. In Section 3, we give the required background on binary linear codes and prove the equivalence between attribute-efficient learning of parities from random uniform examples and decoding high-rate random linear codes from a low number of errors. In Section 4, we show a simple algorithm for ae.naMQ learning of parities. Section 5 gives a way to convert a non-adaptive parity learning algorithm into an algorithm for finding significant Fourier coefficients of a function while preserving attribute-efficiency and non-adaptiveness, yielding an ae.naMQ algorithm for weakly learning DNF expressions. Then in Section 6 we describe our ae.naMQ algorithm for learning DNF expressions and in Section 7 we show how this algorithm can be modified to handle random persistent classification noise.

2. Preliminaries

For vectors $x, y \in \{0,1\}^n$ we denote by $x \oplus y$ the vector obtained by bitwise XOR of x and y; by $[k]$ the set $\{1,2,\ldots,k\}$; by e_i a vector with 1 in i-th position and zeros in the rest; by x_i the i-th element of vector x. Dot product $x \cdot y$ of vectors $x, y \in \{0,1\}^n$ denotes $\sum_i x_i y_i \pmod 2$ or simply vector product xy^T over $\mathbf{GF}(2)$ (with vectors being row vectors by default). By $\mathtt{wt}(x)$ we denote the Hamming weight of x and we define $\mathtt{dist}(x,y) = \mathtt{wt}(x \oplus y)$.

To analyze the accuracy and confidence of estimates produced by random sampling we will use the following standard inequalities.

Lemma 1 (Chernoff) *Let $X_1, \ldots, X_m$ be a sequence of m independent Bernoulli trials, each with probability of success $\mathbf{E}[X_i] = p$ and let $S = \sum_{i=1}^m X_i$. Then for $0 \le \gamma \le 1$,*

$$\mathbf{Pr}[S > (1+\gamma)pm] \le e^{-mp\gamma^2/3}$$

and

$$\mathbf{Pr}[S < (1-\gamma)pm] \le e^{-mp\gamma^2/2} .$$

Lemma 2 (Bienaymé-Chebyshev) *Let $X_1, \ldots, X_m$ be pairwise independent random variables all with mean μ and variance σ^2. Then for any $\lambda \geq 0$,*

$$\mathbf{Pr}\left[\left|\frac{1}{m}\sum_{i=1}^{m} X_i - \mu\right| \geq \lambda\right] \leq \frac{\sigma^2}{m\lambda^2} .$$

For a function $t(\cdots)$ we say a function $q(\cdots)$ (of the same parameters as t) is $\tilde{O}(t(\cdots))$ when there exist constants α and β such that $q(\cdots) \leq \alpha t(\cdots)\log^{\beta}(t(\cdots))$.

2.1 PAC Learning

We study learning of Boolean functions on the Boolean hypercube $\{0,1\}^n$. Our Boolean functions take values $+1$ (true) and -1 (false). Our main interest are the classes of parity functions and DNF expressions. A parity function $\chi_a(x)$ for a vector $a \in \{0,1\}^n$ is defined as $\chi_a(x) = (-1)^{a \cdot x}$. We refer to the vector associated with a parity function as its *index* and the Hamming weight of the vector as the *length* of the parity function. We denote the concept class of parity functions $\{\chi_a \mid a \in \{0,1\}^n\}$ by PAR and the class of all the parities of length at most k by $\mathrm{PAR}(k)$. We represent a parity function by listing all the variables on which it depends. This representation for a parity of length k requires $\theta(k \log n)$ bits.

For the standard DNF representation and any Boolean function f we denote by DNF-size(f) the number of terms in a DNF representation of f with the minimal number of terms. In context of learning DNF this parameter is always denoted s. The uniform distribution over $\{0,1\}^n$ is denoted $\mathcal{U}$.

Our learning model is Valiant's well-known PAC model (Valiant, 1984) for learning Boolean functions over $\{0,1\}^n$. In this model, for a concept c and distribution $\mathcal{D}$ over X, an *example oracle* $\mathrm{EX}_{\mathcal{D}}(c)$ is an oracle that upon request returns an example $\langle x, c(x)\rangle$ where x is chosen randomly with respect to $\mathcal{D}$, independently of any previous examples. For $\varepsilon \geq 0$ we say that function g ε-approximates a function f with respect to distribution $\mathcal{D}$ if $\mathbf{Pr}_{\mathcal{D}}[f(x) = g(x)] \geq 1 - \varepsilon$. For a concept class $\mathcal{C}$, we say that an algorithm $\mathcal{A}$ *efficiently* learns $\mathcal{C}$, if for every $\varepsilon > 0$, n, $c \in \mathcal{C}$, and distribution $\mathcal{D}$ over $\{0,1\}^n$, $\mathcal{A}(n, \varepsilon, s)$ (where s is the size of c in the representation associated with $\mathcal{C}$) outputs, with probability at least $1/2$, and in time polynomial in $n, 1/\varepsilon$, and s a hypothesis h that ε-approximates c. When a learning algorithm is guaranteed to learn only with respect to a specific distribution we specify the distribution explicitly. We say that an algorithm *weakly* learns $\mathcal{C}$ if it produces a hypothesis h that $(\frac{1}{2} - \frac{1}{p(n,s)})$-approximates (or *weakly approximates*) c for some polynomial p.

Note that in this definition of learning we do not use the confidence parameter δ that requires a learning algorithm to succeed with probability at least $1 - \delta$. Instead we assume that it equals $1/2$. In order to obtain an algorithm with success probability $1 - \delta$ one can always use a standard confidence boosting procedure (cf. the textbook by Kearns and Vazirani, 1994). The boosting procedure consists of repeating the original algorithm $k = \log(1/\delta) + 1$ times with slightly increased accuracy (e.g., $\varepsilon/2$), each time on new examples and independent coin flips. The hypotheses obtained from these runs are then tested on an independent sample of size $O(\varepsilon^{-1}\log(1/\delta))$ and the best one is chosen.

A membership query oracle $\mathrm{MEM}(c)$ is the oracle that, given any point $x \in \{0,1\}^n$, returns the value $c(x)$. When learning with respect to $\mathcal{U}$, $\mathrm{EX}_{\mathcal{U}}(c)$ can be trivially simulated using $\mathrm{MEM}(c)$ and therefore $\mathrm{EX}_{\mathcal{U}}(c)$ is not used at all.

An algorithm $\mathcal{A}$ is said to be *attribute-efficient* if the number of examples (both random and received from the MQ oracle) it uses is polynomial in the size of the representation of the concept and $1/\varepsilon$. We say that a variable x_i is *relevant* for a function f if there exists $y \in \{0,1\}^n$ such that $f(y) \neq f(y \oplus e_i)$. The number of relevant variables of the target concept is denoted by parameter r. Attribute-efficiency does not allow the number of examples to depend polynomially on n. Instead the number of examples used can depend polynomially on r and $\log n$ since for most representations (including the ones considered in this work) the size of the representation of f is lower bounded by both $\log n$ and r.

2.1.1 NOISE MODELS

We consider two standard models of noise in learning. The first one is the well-studied random classification noise model introduced by Angluin and Laird (1988). In this model for any $\eta \leq 1/2$ called the *noise rate* the regular example oracle $\mathrm{EX}_{\mathcal{D}}(c)$ is replaced with the faulty oracle $\mathrm{EX}^{\eta}_{\mathcal{D}}(c)$. On each call, $\mathrm{EX}^{\eta}_{\mathcal{D}}(c)$, draws x according to $\mathcal{D}$, and returns $\langle x, c(x)\rangle$ with probability η and $\langle x, \neg c(x)\rangle$ with probability $1-\eta$. When η approaches $1/2$ the result of the corrupted query approaches the result of the random coin flip, and therefore the running time of algorithms in this model is allowed to polynomially depend on $\frac{1}{1-2\eta}$.

This model of noise is not suitable for corrupting labels returned by $\mathrm{MEM}(c)$ since a learning algorithm can, with high probability, find the correct label at point x by asking the label of x polynomial (in $\frac{1}{1-2\eta}$) number of times and then returning the label that appeared in the majority of answers. An appropriate modification of the noise model is the introduction of *random persistent classification noise* by Goldman, Kearns, and Schapire (1993). In this model, as before, the answer to a query at each point x is flipped with probability $1-\eta$. However, if the membership oracle was already queried about the value of f at some specific point x or x was already generated as a random example, the returned label has the same value as in the first occurrence (i.e., in such a case the noise persists and is not purely random). If the learner does not ask for the label of a point more than once then this noise can be treated as the usual independent random classification noise.

2.1.2 FOURIER TRANSFORM

The Fourier transform is a technique for learning with respect to the uniform distribution (primarily) based on the fact that the set of all parity functions $\{\chi_a(x)\}_{a\in\{0,1\}^n}$ forms an orthonormal basis of the linear space of real-valued function over $\{0,1\}^n$. This fact implies that any real-valued function f over $\{0,1\}^n$ can be uniquely represented as a linear combination of parities, that is $f(x) = \sum_{a\in\{0,1\}^n} \hat{f}(a)\chi_a(x)$. The coefficient $\hat{f}(a)$ is called Fourier coefficient of f on a and equals $\mathbf{E}_{\mathcal{U}}[f(x)\chi_a(x)]$; a is called the *index* and $\mathtt{wt}(a)$ the *degree* of $\hat{f}(a)$. Given the values of f on all the points of the hypercube $\{0,1\}^n$ one can compute the values of all the Fourier coefficients $\{\hat{f}(a)\}_{a\in\{0,1\}^n}$ using the Fast Fourier Transform (FFT) algorithm in time $O(n2^n)$ (Cooley and Tukey, 1965). The same algorithm FFT also converts the set of all Fourier coefficients $\{\hat{f}(a)\}_{a\in\{0,1\}^n}$ into the values of the function f on all the points of the hypercube. This transformation is called inverse Fourier transform. For further details on the technique we refer the reader to the survey by Mansour (1994).

2.1.3 RANDOMIZED FUNCTIONS

Besides deterministic functions on $\{0,1\}^n$ we will also deal with functions whose value on a point x is a real-valued random variable $\Psi(x)$ independent of $\Psi(y)$ for any $y \neq x$ and of any previous evaluations of $\Psi(x)$. To extend learning and Fourier definitions to this case we include the probability over the random variable Ψ in estimations of probability, expectation and variance. For example, we say that a randomized function Ψ ε-approximates f with respect to $\mathcal{D}$ if $\mathbf{Pr}_{\mathcal{D},\Psi}[f(x) = \Psi(x)] \geq 1-\varepsilon$. Similarly, $\widehat{\Psi}(a) = \mathbf{E}_{\mathcal{U},\Psi}[\Psi(x)\chi_a(x)]$.

2.2 Learning by Non-adaptive Membership Queries

We say that an algorithm $\mathcal{A}$ uses MQs *non-adaptively* if it can be split into two stages. The first stage, given all the parameters of learning, (n, ε and a bound on the size of the target concept) and access to points randomly sampled with respect to the target distribution, generates a set of points $S \subseteq \{0,1\}^n$. The second stage, given the labels of the random points and the answers from $\mathrm{MEM}(c)$ on points in S, that is, the set $\{(x, c(x)) \mid x \in S\}$, computes a hypothesis (or, in general, performs some computation). Neither of the stages has any other access to $\mathrm{MEM}(c)$.

We note that in the general definition of PAC learning we did not assume that size of the target concept (or a bound on it) is given to the learning algorithm. When learning with adaptive queries a good bound can be found via the "guess-and-double" technique, but for non-adaptive algorithms we will assume that this bound is always given. To emphasize this we specify the parameters that have to be given to a non-adaptive algorithm in the name of the algorithm. Clearly the same "guess-and-double" technique can be used to produce a sequence of independent and non-adaptive executions of the learning algorithm.

The immediate consequence of non-adaptiveness is that in order to parallelize a non-adaptive learning algorithm only the usual computation has to be parallelized since all the MQs can be made in parallel. Non-adaptiveness is also useful when learning ℓ concepts from the same concept class in parallel. The fact that queries are independent of the target concept implies that same set of points can be used for learning different concepts. To achieve probability of success $1/2$ in learning of all ℓ concepts we will have to learn with each concept with probability of success $1-1/(2\ell)$. This implies that the number of points needed for learning might grow by a factor of $\log \ell$ whereas in the general case ℓ times more examples might be required.

Results of Goldreich et al. (1986) imply that if one-way functions exist then the concept class of all polynomial circuits is not learnable even with respect to $\mathcal{U}$ and with access to a MQ oracle (Kearns and Valiant, 1994). By modifying the values of each circuit to encode the circuit itself in a polynomial number of fixed points one can make this class learnable by non-adaptive MQs but not learnable from random and uniform examples only (the modification is very unlikely to be detected by random examples yet MQs to the fixed points will reveal the circuit). Similarly, by placing the encoding of the circuit in some location that is encoded in a fixed location, one can create a function class learnable by adaptive membership queries but not learnable by the non-adaptive ones (if one-way functions exist). Further details of these simple separations are left to the reader.

3. Learning of Parities and Binary Linear Codes

In this section we show that attribute-efficient learning of parities with respect to the uniform distribution from random examples only is likely to be hard by proving that it is equivalent to an open

problem in coding theory. Unlike in the rest of the paper in this section and the following section parity functions will be functions to $\{0,1\}$. To emphasize this we use $\dot{\chi}$ instead of χ.

3.1 Background on Linear Codes

We say that a code C is an $[m,n]$ code if C is a binary linear code of block length m and message length n. Any such code can be described by its $n \times m$ *generator matrix* G as follows: $C = \{xG \mid x \in \{0,1\}^n\}$. Equivalently, a code can be described by its *parity-check* matrix H of size $m \times (m-n)$ by $C = \{y \mid yH = 0^{m-n}\}$. It is well-known that $G \cdot H = 0^{n \times (m-n)}$ and decoding given a corrupted message y is equivalent to decoding given the *syndrome* of the corrupted message. The syndrome equals to yH and the decoding consists of finding a vector e of Hamming weight at most w such that $y \oplus e = xG$, where $w = \lfloor (d-1)/2 \rfloor$ and d is the *distance* of the code (cf. the book by van Lint, 1998). For a linear code C the distance equals to the Hamming weight of a non-zero vector with the smallest Hamming weight.

By saying that C is a *random* $[m,n]$ *code* we mean that C is defined by choosing randomly, uniformly, and independently n vectors in $\{0,1\}^m$ that form the basis of C. Alternatively, we can say that the generator matrix G of C was chosen randomly with each entry equal to 1 with probability $1/2$ independently of others. We denote this distribution by $\mathcal{U}_{n \times m}$. Some authors restrict the random choice of G's to matrices of full rank n. As we will see, this definitions would only make our proofs simpler.

Binary linear codes generated randomly meet the Gilbert-Varshamov bound with high probability, that is, they achieve the best known *rate* (or n/m) versus distance trade-off (cf. the lecture notes by Sudan, 2002). However decoding a random linear code or even determining its distance is a notorious open problem in coding theory. For example the McEliece cryptosystem is based, among other assumptions, on the hardness of this problem (McEliece, 1978). Besides that, while the average-case hardness of this problem is unknown, a number of worst-case problems related to decoding linear codes are NP-hard (Barg, 1997; Vardy, 1997; Sudan, 2002).

A potentially simpler version of this problem in which the errors are assumed to be random and independent with some rate η (and not adversarial as in the usual definition) is equivalent to learning of parities with random classification noise of rate η, a long-standing open problem in learning theory. In fact, Feldman et al. (2006) have proved that when learning parities from random and uniform examples, random classification noise of rate η is as hard as adversarial noise of rate η (up to a polynomial blowup in the running time). The only known non-trivial algorithm for learning parities with noise is a slightly subexponential algorithm by Blum et al. (2000). In our discussion η is very low (e.g., $\frac{\log n}{n}$), yet even for this case no efficient noise-tolerant algorithms are known.

Correcting a random linear $[m,n]$ from up to w errors is defined as follows.

Definition 3 **Input:** *An $n \times m$ binary generator matrix G randomly chosen according to $\mathcal{U}_{n \times m}$ and $y \in \{0,1\}^m$.*
Output: *$x \in \{0,1\}^n$ such that $\mathtt{dist}(xG, y) \leq w$ if there exists one.*

A successful algorithm for this problem is an algorithm that would allow to correct up to w errors in a "good" fraction of randomly created linear codes. That is, with non-negligible probability over the choice of G, and for every y, the algorithm should produce the desired output. Note that the algorithm can only be successful when the code generated by G has distance at least $2w+1$.

For simplicity, we will usually assume a constant probability of success but all the results can be translated to algorithms having the success probability lower-bounded by a polynomial (in m) fraction.

3.2 The Reduction

The equivalence of attribute-efficient learning of parities with respect to the uniform distribution and decoding of random linear codes relies on two simple lemmas. The first one, due to Hofmeister (1999), is that the syndrome decoding of a linear code implies attribute-efficient learning of parities. We include it with a proof for completeness.

Lemma 4 (Hofmeister) *Let H be a parity-check matrix of some $[m,n]$ w-error correcting code C. Let $\mathcal{A}$ be an algorithm that for any $y \in \{0,1\}^m$ such that $y = c \oplus e$ where $c \in C$ and $\mathtt{wt}(e) \leq w$, given the syndrome yH, finds e. Then $\mathcal{A}$ learns PAR(w) over $\{0,1\}^m$ given the values of an unknown parity on the columns of H.*

Proof The condition $y = c \oplus e$ for $c \in C$ implies that $yH = eH$. Therefore the syndrome yH is equal to the vector $eH = \dot{\chi}_e(H_1), \dot{\chi}_e(H_2), \ldots, \dot{\chi}_e(H_{m-n})$ where H_i is the i-th column of H. Therefore finding an error vector e of weight at most w using the syndrome yH is the same as finding a parity of length at most w given the values of the unknown parity on the columns of H. ■

This observation has lead Hofmeister to a simple ae.naMQ algorithm for learning parities that uses the columns of the parity check matrix of BCH code as MQs. We note that the converse of this lemma is only true if the learning algorithm is *proper*, that is, produces a parity function in PAR(w) as a hypothesis.

To obtain the claimed equivalence for the uniform distribution we first need to prove that generating a linear code by choosing a random and uniform parity check matrix (that is, from $\mathcal{U}_{n\times m-n}$) is equivalent to (or indistinguishable from) generating a linear code by choosing a random and uniform generator matrix (that is, from $\mathcal{U}_{n\times m}$).

Let $p(i,j)$ denote the probability that i vectors chosen randomly and uniformly from $\{0,1\}^j$ are linearly independent. Each $i \geq 1$ linearly independent vectors span subspace of size 2^i and therefore there are $2^j - 2^i$ vectors that are linearly independent of them. This implies that, $p(i+1,j) = p(i,j)(1-2^{-j+i})$. All vectors except for 0^j form a linearly independent set of size 1. Therefore $p(1,j) = (1-2^{-j})$. Hence

$$p(i,j) = (1-2^{-j}) \cdot (1-2^{-j+1}) \cdots (1-2^{-j+i-1}) \ .$$

Note that

$$p(i,j) \geq 1 - 2^{-j} - 2^{-j+1} - \cdots - 2^{-j+i-1} > 1 - 2^{-j+i} \tag{1}$$

and for $i = j$, $p(j,j) = \frac{1}{2}p(j,j-1) > \frac{1}{2}(1-\frac{1}{2}) = \frac{1}{4}$. This means that for any $i \leq j$, $p(i,j) > 1/4$.

Let $\mathcal{V}_{n\times m}$ denote the distribution on matrices of size $n \times m$ resulting from the following process. Choose randomly and uniformly a $m \times (m-n)$ matrix H of rank $m-n$ and then choose randomly and uniformly a matrix G of size $n \times m$ of rank n such that $GH = 0^{n\times(m-n)}$. To generate G's like this we find a basis $b_1, \ldots, b_n$ for the subspace of $\{0,1\}^m$ that is "orthogonal" to H in the standard (and efficient) way. Let G_0 denote the matrix whose rows are the vectors $b_1, \ldots, b_n$. It is easy to see that any matrix G of rank n such that $GH = 0^{n\times(m-n)}$, can be represented uniquely as $F \cdot G_0$ where F is a matrix of size $n \times n$ and full rank $(*)$. Therefore we can generate G's as above by choosing

randomly and uniformly a matrix F of rank n. If we choose a random matrix F according $\mathcal{U}_{n\times n}$, with probability at least $p(n,n) > 1/4$, it will have the full rank. We can repeatedly sample from $\mathcal{U}_{n\times n}$ to get a full-rank F with any desired probability. This implies that we can generate a matrix according to $\mathcal{V}_{n\times m}$ with probability $1-\delta$ in time $O(m^3 \log(1/\delta))$ (or less if a non-trivial matrix multiplication algorithm is used).

All we need to prove now is that $\mathcal{V}_{n\times m}$ is "close" to $\mathcal{U}_{n\times m}$. More specifically, the *statistical distance* between two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ over X is defined to be $\Delta(\mathcal{D}_1, D_2) = \frac{1}{2}\sum_{x\in X} |\mathcal{D}_1(x) - \mathcal{D}_2(x)|$. It is well known and easy to see that for any event $E \subseteq X$, $|\mathbf{Pr}_{\mathcal{D}_1}[x \in E] - \mathbf{Pr}_{\mathcal{D}_2}[x \in E]| \leq \Delta(\mathcal{D}_1, D_2)$.

Lemma 5 *The distribution $\mathcal{V}_{n\times m}$ is uniform over matrices of size $n \times m$ and rank n. In particular, $\Delta(\mathcal{V}_{n\times m}, \mathcal{U}_{n\times m}) \leq 2^{-m+n}$.*

Proof Let G be any matrix of size $n \times m$ with linearly independent rows. Its probability under $\mathcal{U}_{n\times m}$ is $\mathcal{U}_{n\times m}(G) = 2^{-mn}$. When sampling with respect to $\mathcal{V}_{n\times m}$, G can be obtained only if all the columns of H are "orthogonal" to rows of G, that is belong to a linear subspace of $\{0,1\}^m$ of dimension $m-n$. The total number of H's like these of rank $m-n$ is $2^{(m-n)^2} p(m-n, m-n)$ (as follows from $(*)$) and the total number of matrices size $m \times (m-n)$ of rank $m-n$ is $2^{m(m-n)} p(m-n, m)$. Therefore the probability of getting each H like this is $2^{-n(m-n)} \frac{p(m-n,m-n)}{p(m-n,m)}$. Given H the total number of matrices of size $n \times m$ and rank n that are "orthogonal" to H is $p(n,n)2^{n^2}$ (as follows from $(*)$) and therefore G will be generated with probability $2^{-n^2}/p(n,n)$. Hence the total probability of G under $\mathcal{V}_{n\times m}$ is $\mathcal{V}_{n\times m}(G) = 2^{-mn} \frac{p(m-n,m-n)}{p(m-n,m)p(n,n)}$. For every $i < j$, $p(j-i, j)p(i,i) = p(j,j)$. Therefore $\mathcal{V}_{n\times m}(G) = 2^{-mn}/p(n,m)$. This implies that $\mathcal{V}_{n\times m}$ is uniform over matrices of size $n \times m$ and rank n. The statistical distance between $\mathcal{V}_{n\times m}$ and $\mathcal{U}_{n\times m}$ equals to

$$\begin{aligned} \frac{1}{2} \sum_{G\in\{0,1\}^{n\times m}} |\mathcal{V}_{n\times m}(G) - \mathcal{U}_{n\times m}(G)| &= \\ \frac{1}{2}\left[\sum_{\mathtt{rank}(G)<n} 2^{-mn} + \sum_{\mathtt{rank}(G)=n} 2^{-mn}\left(\frac{1}{p(n,m)} - 1\right)\right] &= 1 - p(n,m). \end{aligned}$$

According to Equation (1), $1 - p(n,m) < 1 - (1 - 2^{-m+n}) = 2^{-m+n}$. ■

We can now prove that decoding of random linear codes implies attribute-efficient learning of parities from random examples only.

Theorem 6 *Assume that there exists an algorithm* `RandDec` *that corrects a random linear $[m,n]$ code from up to w errors with probability at least $1/2+\gamma$ for any constant γ. Then PAR(w) over $\{0,1\}^m$ is efficiently learnable from $m-n$ random examples.*

Proof Let $\dot{\chi}_e \in \mathrm{PAR}(w)$ be the unknown parity function and $z_1, z_2, \ldots, z_{m-n}$ be random and uniform examples given by the example oracle. Let H be the $m \times (m-n)$ matrix whose column i is equal to z_i for each $i \leq m-n$. If H does not have rank $m-n$ we return χ_{0^m}. Otherwise let G be a random matrix such that $GH = 0^{n\times(m-n)}$ generated as in the description of $\mathcal{V}_{n\times m}$ for $\delta = \gamma/2$. The values of $\dot{\chi}_e$ on z_i's give us the vector eH. Let y be any solution to the linear equation $yH = eH$. Clearly $(y \oplus e)H = 0^{m-n}$ and therefore $y \oplus e$ equals to xG for some $x \in \{0,1\}^n$. This means that `RandDec` (if

successful) will output x on input G and y. By the definition of x, $e = xG \oplus y$, giving us the desired parity function.

To analyze the success probability of the algorithm we observe that the procedure above generates G according to $\mathcal{V}_{n\times m}$ with probability at least $p(m-n,m)(1-\gamma/2) \geq 1-2^{-n}-\gamma/2$. According to Lemma 5, the statistical distance between the G generated as above and $\mathcal{U}_{n\times m}$ is at most 2^{-m+n}. `RandDec` is successful with probability $1/2+\gamma$ and therefore our algorithm will succeed with probability at least $1/2+\gamma-(\gamma/2+2^{-n}+2^{-m+n}) \geq 1/2$. ■

The transformation above produces an attribute-efficient algorithm only if $m-n$ is polynomial in w and $\log m$. According to the Gilbert-Varshamov bound, a random linear code will, with high probability, have distance $d = \Omega(\frac{m-n}{\log m})$. Therefore if the number of errors that `RandDec` can correct is at least $w = d^{\alpha}$ errors for some constant $\alpha > 0$ then the sample complexity of learning a parity of length at most w over m variables would equal $O(w^{1/\alpha}\log m)$. Therefore such an algorithm could be used to obtain an attribute-efficient algorithm for learning parities.

We have noted previously that using a parity learning algorithm to obtain a syndrome decoding algorithm requires the parity learning algorithm to be proper. When a distribution over examples is not restricted it is unknown whether proper learning of parities is harder than non-proper. Fortunately, when learning with respect to the uniform distribution any learning algorithm for parities can be converted to a proper and exact one (that is, with a hypothesis equal to the target function). We include a proof of this folklore fact for completeness.

Fact 7 *Let $\mathcal{A}$ be an algorithm that learns PAR(k) in time $t(n,k,\varepsilon)$ and with sample complexity $s(n,k,\varepsilon)$. Then there exists a probabilistic algorithm $\mathcal{A}'$ that learns PAR(k) properly and exactly in time $t(n,k,1/5)+\tilde{O}(nk)$ and using $s(n,k,1/5)$ samples.*

Proof We assume for simplicity that if $\mathcal{A}$ is probabilistic then it succeeds with probability at least $3/4$. Let h be the output of $\mathcal{A}$ when running on an unknown parity $\dot{\chi}_e \in \mathrm{PAR}(k)$ with $\varepsilon = 1/5$. Given h that is correct on $4/5$ of all the points we can use it simulate membership queries to $\dot{\chi}_e(x)$ as follows. Let $y \in \{0,1\}^n$ be any point and let x be a randomly and uniformly chosen point. Then $h(x) = \dot{\chi}_e(x)$ with probability at least $4/5$ and $h(x\oplus y) = \dot{\chi}_e(x\oplus y)$ with probability at least $4/5$. Therefore with probability at least $3/5$, $h(x)\oplus h(x\oplus y) = \dot{\chi}_e(x)\oplus\dot{\chi}_e(x\oplus y) = \dot{\chi}_e(y)$. We can increase the confidence in the label to $1-\delta$ by repeating this procedure for $O(\log(1/\delta))$ independent x's. Given these membership queries we can use a proper and exact MQ algorithm for learning $\mathrm{PAR}(k)$. A number of such algorithms are known running in time $\tilde{O}(nk)$ and using $O(k\log n)$ MQs (including `AEParityStat`(k) given in Theorem 9). In order to get correct answers to all the membership queries with probability at least $3/4$ we need each of the MQs to be correct with probability $1-\delta$ for $\delta = \Omega(\frac{1}{k\log n})$. This means that making $O(k\log n)$ MQs will take $O(nk\log n\log(k\log n)) = \tilde{O}(nk)$ steps. Altogether we get algorithm $\mathcal{A}'$ that succeeds with probability at least $1/2$ and has the claimed complexity bounds. ■

We can now assume that algorithms for learning parity with respect to the uniform distribution are proper and exact (and in particular do not require parameter ε) and use this to obtain the other direction of the equivalence.

Theorem 8 *Assume that there exists an algorithm* `AELearnPar`$_{\mathcal{U}}$`(k)` *that efficiently learns PAR(k) over $\{0,1\}^m$ using at most $q(m,k)$ random examples. Then there exists an algorithm* `RandDec` *that corrects a random linear $[m,m-q(m,k)]$ code from up to k errors with probability at least $1/2-\gamma$ for any constant $\gamma > 0$.*

Proof Let G and y be the input of $\texttt{RandDec}$, $n = m - q(m,k)$, x be the vector for which $y = xG \oplus e$ where $\texttt{wt}(e) \leq k$. If G is not of rank n we just return the vector 0^n. Otherwise let H be a random matrix such that $GH = 0^{n\times(m-n)}$ generated as rank $m-n$ we return χ_{0^m}. Otherwise let G be a random matrix such that $GH = 0^{n\times(m-n)}$ generated as in the description of $\mathcal{V}_{n\times m}$ for $\delta = \gamma/2$ (with the roles of G and H reversed).

The syndrome yH is equal to eH and gives the values of $\dot{\chi}_e$ on $q(m,k)$ columns of H. We feed these columns as random examples to $\texttt{AELearnPar}_{\mathcal{U}}(\texttt{k})$ and obtain $\dot{\chi}_e$ from it (if $\texttt{AELearnPar}_{\mathcal{U}}(\texttt{k})$ is successful). Given e we obtain x by solving the system of linear equations $xG = y \oplus e$. To analyze the success probability of the algorithm we observe that the procedure above generates H according to $\mathcal{V}_{m\times(m-n)}$ with probability at least $p(n,m)(1-\gamma/2) \geq 1 - 2^{-q(m,k)} - \gamma/2$. According to Lemma 5, the statistical distance between H's generated as above and $\mathcal{U}_{m\times(m-n)}$ is at most $2^{-m+(m-n)} = 2^{-n}$. Therefore $\texttt{AELearnPar}_{\mathcal{U}}(\texttt{k})$ will succeed with probability at least $1/2 - 2^{-n}$. This implies that $\texttt{RandDec}$ will return the correct x with probability at least $1/2 - (2^{-m+q(m,k)} + 2^{-q(m,k)} + \gamma/2) \geq 1/2 - \gamma$. ■

4. A Fast Randomized Algorithm for ae.naMQ Learning of Parities

We next present a simple randomized algorithm for ae.naMQ learning of parities. The only previously known ae.naMQ algorithm for learning parities is due to Hofmeister (1999) and is a deterministic algorithm based on constructing and decoding of BCH binary linear codes (see also Section 3.2). The algorithm we present is substantially simpler and has essentially the same asymptotic complexity as Hofmeister's.

The basic idea of our algorithm is to use a distribution over $\{0,1\}^n$ for which each attribute is correlated with the parity function if and only if it is present in the parity.

Theorem 9 *For each $k \leq n$ there exists an algorithm $\texttt{AEParityStat}(k)$ that ae.naMQ learns the class $PAR(k)$ in time $O(nk\log n)$ and asks $O(k\log n)$ MQs.*

Proof Let $\dot{\chi}_c$ be the target concept (such that $\texttt{wt}(c) \leq k$). We define $\mathcal{D}_{\frac{1}{t}}$ to be the product distribution such that for each i, $\mathbf{Pr}[x_i = 1] = \frac{1}{t}$. Let us draw a point x randomly according to distribution $\mathcal{D}_{\frac{1}{4k}}$. Then for each $i \leq n$

$$\begin{aligned}\mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[x_i = 1 \text{ and } \dot{\chi}_c(x) = 1] &= \mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\dot{\chi}_c(x) = 1 \mid x_i = 1]\,\mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[x_i = 1] \\ &= \frac{1}{4k}\mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\dot{\chi}_c(x) = 1 \mid x_i = 1]\ .\end{aligned}$$

Our second observation is that for any set of indices $B \subseteq [n]$ and the corresponding parity function $\dot{\chi}_b$,

$$\mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\dot{\chi}_b(x) = 1] \leq 1 - \mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\forall i \in B,\ x_i = 0] = 1 - (1 - \frac{1}{4k})^{|B|} \leq \frac{|B|}{4k}\ .$$

First examine the case that $c_i \neq 1$ and therefore does not influence $\dot{\chi}_c$. Then by the second observation,

$$\mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\dot{\chi}_c(x) = 1 \mid x_i = 1] = \mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\dot{\chi}_c(x) = 1] \leq \frac{k}{4k} \leq 1/4\ .$$

Now assume that $c_i = 1$ and let $c' = c \oplus e_i$. Then $\dot{\chi}_{c'}(x)$ is independent of x_i and $\dot{\chi}_c(x) = 1$ if and only if $\dot{\chi}_{c'}(x) = 0$. Therefore

$$\begin{aligned}\mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\dot{\chi}_c(x) = 1 \mid x_i = 1] &= \mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\dot{\chi}_{c'}(x) = 0 \mid x_i = 1] \\ &= 1 - \mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[\dot{\chi}_{c'}(x) = 1] \geq 1 - \frac{k-1}{4k} > 3/4\,.\end{aligned}$$

Hence estimation of $\mathbf{Pr}_{\mathcal{D}_{\frac{1}{4k}}}[x_i = 1 \text{ and } \dot{\chi}_c(x) = 1]$ within the half of the expectation can be used to find out whether $c_i = 1$. Lemma 1 for $\gamma = 1/2$ implies that by taking $O(k \log n)$ independent samples with respect to $\mathcal{D}_{\frac{1}{4k}}$ we will get that each estimate is correct with probability at least $1 - 1/(2n)$ and therefore we will discover c with probability at least $1 - n/(2n) = 1/2$. The running time of `AEParityStat`(k) is clearly $O(nk \log n)$. ■

5. Finding Fourier Coefficients and Weak DNF Learning

The original Jackson's algorithm for learning DNF expressions with respect to the uniform distribution is based on a procedure that weakly learns DNF with respect to the uniform distribution (Jackson, 1997). The procedure for weak learning is essentially an algorithm that, given a Boolean function f finds a significant Fourier coefficient of f, if one exist. Jackson's algorithm is based on a technique by Goldreich and Levin (1989) for finding a significant Fourier coefficient (also called the KM algorithm (Kushilevitz and Mansour, 1991)). Bshouty, Jackson, and Tamon (1999) used a later algorithm by Levin (1993) to give a significantly faster weak learning algorithm. In this section we will briefly describe Levin's algorithm with improvements by Bshouty et al.. Building on their ideas we then present an attribute-efficient and non-adaptive version of the improved Levin's algorithm. This algorithm will give us an ae.naMQ algorithm for weak learning of DNF expressions that will serve as the basis of our ae.naMQ algorithm for DNF learning.

A Fourier coefficient $\hat{\phi}(a)$ of a real-valued function ϕ over $\{0,1\}^n$ is said to be θ-*heavy* if $|\hat{\phi}(a)| \geq \theta$. For a Boolean f, $\mathbf{E}[f\chi_a] \geq \theta$ if and only if $\mathbf{Pr}[f = \chi_a] \geq 1/2 + \theta/2$. This means that $|\hat{f}(a)| \geq \theta$ is equivalent to either χ_a or $-\chi_a$ being a $(1/2 - \theta/2)$-approximator of f. Therefore finding a significant Fourier coefficient of f is sometimes called weak parity learning (Jackson, 1997). It can also be interpreted as a learning algorithm for parities in the agnostic learning framework of Haussler (1992) and Kearns et al. (1994) Feldman et al. (see the work of 2006, for details).

Definition 10 (Weak Parity Learning) *Let f be a Boolean function with at least one θ-heavy Fourier coefficient. Given $\theta > 0$ and access to MEM(f), the weak parity learning problem consists of finding a vector z such that $\hat{f}(z)$ is $\theta/2$-heavy.*

We will only consider algorithms for weak parity learning that are efficient, that is, produce the result in time polynomial in n, and θ^{-1}. In addition we are interested in weak parity learning algorithms that are attribute-efficient.

Definition 11 (Attribute-Efficient Weak Parity Algorithm) Attribute-efficient weak parity algorithm *is an algorithm that given k, θ, and MEM(f) for f that has a θ-heavy Fourier coefficient of degree at most k efficiently solves weak parity learning problem and asks polynomial in $k, \log n$, and θ^{-1} number of MQs.*

We follow the presentation of Levin's weak parity algorithm given by Bshouty et al. and refer the reader to their paper for detailed proofs of all the statements and smaller remarks (we use the same definitions and notation to simplify the reference). Levin's algorithm is based on estimating a Fourier coefficient $\hat{f}(a)$ by sampling f on randomly-chosen pairwise independent points. More specifically, the following pairwise independent distribution is generated. For a fixed m, a random m-by-n 0-1 matrix R is chosen and the set $Y = \{pR \mid p \in \{0,1\}^m \setminus \{0^m\}\}$ is formed. For different vectors p_1 and p_2 in $\{0,1\}^m \setminus \{0^m\}$, p_1R and p_2R are pairwise independent. The variance σ^2 of a Boolean function is upper-bounded by 1 and thus Bienaymé-Chebyshev's inequality (Lemma 2) implies that

$$\mathbf{Pr}_R\left[|\frac{\sum_{x\in Y} f(x)\chi_a(x)}{2^m-1} - \hat{f}(a)| \geq \gamma\right] \leq \frac{1}{(2^m-1)\gamma^2} \tag{2}$$

Therefore using a sample for $m = \log(16\rho^{-1}\theta^{-2}+1)$, $\sum_{x\in Y} f(x)\chi_a(x)$ will, with probability at least $1-\rho$, approximate $\hat{f}(a)$ within $\theta/4$.

On the other hand, $\sum_{x\in Y} f(x)\chi_a(x)$ is a summation over all (but one[2]) elements of a linear subspace of $\{0,1\}^n$ and therefore can be seen as a Fourier coefficient of f restricted to subspace Y. That is, if we define $f_R(p) = f(pR)$ then, by definition of Fourier transform, for every $z \in \{0,1\}^m$

$$\widehat{f_R}(z) = 2^{-m} \sum_{p\in\{0,1\}^m} f_R(p)\chi_z(p) \ .$$

This together with equality $\chi_a(pR) = \chi_{aR^T}(p)$ implies that $\hat{f}(a)$ is approximated by $\widehat{f_R}(aR^T)$ (with probability at least $1-\rho$).

All the coefficients $\widehat{f_R}(z)$ can be computed exactly in time $O(m2^m)$ via the FFT algorithm giving estimations to all the Fourier coefficients of f.

Another key element of the weak parity algorithm is the following equation (Bshouty et al., 1999).

Lemma 12 *For $c \in \{0,1\}^n$ let $f_c(x) = f(x \oplus c)$. Then $\widehat{f_c}(a) = \hat{f}(a)\chi_a(c)$.*

Proof

$$\widehat{f_c}(a) = 2^{-n} \sum_{x\in\{0,1\}^n} f(x\oplus c)\chi_a(x) = 2^{-n} \sum_{x\in\{0,1\}^n} f(x)\chi_a(x\oplus c) = \hat{f}(a)\chi_a(c) \ .$$

■

Assuming that $\hat{f}(a) \geq \theta$ estimation of $\hat{f}(a)$ within $\theta/4$ (when successful) has the same sign as $\hat{f}(a)$. Similarly we can obtain the sign of $\widehat{f_c}(a)$. By Lemma 12, the sign of the product $\hat{f}(a)\widehat{f_c}(a)$ is equal to $\chi_a(c)$. This gives a way to make MQs for χ_a using the values $\widehat{f_{c,R}}(aR^T)$ for a random R. Levin and Bshouty et al. implicitly used this technique with a basic membership query algorithm for learning parities. The speed-up in Levin's algorithm is achieved by making each MQ to many χ_a's in parallel. Therefore only a non-adaptive membership query algorithm for learning parities can be used. In our next theorem we give an interpretation of improved Levin's algorithm that makes the use of a non-adaptive membership query algorithm explicit.

2. The value at 0^m does not influence the estimation substantially and therefore can be offset by slightly increasing the size of sample space Y (Bshouty et al., 1999).

Theorem 13 *Let $\mathcal{B}(k)$ be an ae.naMQ algorithm for learning parities that runs in time $t(n,k)$ and uses $q(n,k)$ MQs. There exists an attribute-efficient and non-adaptive algorithm* `AEBoundedSieve-`$\mathcal{B}(\theta,k)$ *that, with probability at least $1-\delta$, solves the weak parity learning problem.* `AEBoundedSieve-`$\mathcal{B}(\theta,k)$ *runs in time $\tilde{O}\left(\theta^{-2}t(n,k)\cdot q(n,k)\log(1/\delta)\right)$ and asks $\tilde{O}\left(\theta^{-2}q^2(n,k)\log(1/\delta)\right)$ MQs.*

Proof We assume for simplicity that $\mathcal{B}(k)$ succeeds with probability at least $3/4$. Besides that according to Fact 7, we can assume that $\mathcal{B}(k)$ is a proper algorithm.

Let S be the set of MQs for an execution of $\mathcal{B}(k)$. Choose randomly an m-by-n matrix R for $m=\log(16\theta^{-2}\cdot 4\cdot(q(n,k)+1)+1)$ and compute the Fourier transforms of $f_R=f_{0^n,R}$ and $f_{y,R}$ for each $y\in S$ via the FFT algorithm. Then, for each $z\in\{0,1\}^m$, we run $\mathcal{B}(k)$ with the answer to MQ $y\in S$ equal to $\mathtt{sign}(\widehat{f_R}(z)\widehat{f_{y,R}}(z))$. If the output of $\mathcal{B}(k)$ is a parity function χ_a of length at most k then we test that $(i):|\widehat{f_R}(z)|\geq 3\theta/4$ and $(ii):aR^T=z$. If both conditions are satisfied we add a to the set of hypotheses H.

By Equation (2), for a such that $|\hat{f}(a)|\geq\theta$ and $\mathtt{wt}(a)\leq k$, with probability at least $1-\frac{1}{4(q(n,k)+1)}$, each of the estimations $\widehat{f_{y,R}}(aR^T)$ for $y\in S\cup\{0^n\}$ will be within $\theta/4$ of $\widehat{f_y}(a)$. In particular, with probability at least $3/4$, for all $y\in S\cup\{0^n\}$, $\mathtt{sign}(\widehat{f_y}(a))=\mathtt{sign}(\widehat{f_{y,R}}(aR^T))$. If all the signs are correct then by Lemma 12, $\mathtt{sign}(\widehat{f_R}(z)\widehat{f_{y,R}}(z))=\chi_a(y)$ and as a result $\mathcal{B}(k)$ will succeed with probability at least $3/4$. Therefore a will satisfy both conditions (i) and (ii) and will be added as a possible hypothesis with probability at least $1/2$. Note that $\mathcal{B}(k)$ is executed on up to 2^m possible hypotheses while using the same set of queries S. This is only possible for a non-adaptive algorithm $\mathcal{B}(k)$.

On the other hand, for any fixed b such that $|\hat{f}(b)|<\theta/2$, if $bR^T=z$ (condition (ii)) then with probability at least $1-\frac{1}{4(q(n,k)+1)}\geq 7/8$, $\widehat{f_R}(z)$ approximates $\hat{f}(b)$ within $\theta/4$. This implies that $|\widehat{f_R}(z)|<3\theta/4$ and therefore condition (i) will be failed with probability at least $7/8$. This implies that b can be added to the set of hypotheses with probability at most $1/8$.

Now we use a simple method of Bshouty et al. (1999) to remove all "bad" (not $\theta/2$-heavy) hypotheses from the set of hypotheses without removing the "good" ones (θ-heavy). We repeat the described algorithm ℓ times for independent choices of R and S generating ℓ sets of hypotheses (each of size at most 2^m). This procedure generates at most $\ell 2^m$ hypotheses. According to Chernoff's bound (Lemma 1) each "good" hypothesis appears in at least $1/3$ of all the sets with probability at least $1-2^{-\alpha\ell}$ and each fixed "bad" hypothesis appears in at least $1/3$ of all the sets with probability at most $2^{-\alpha\ell}$, for a fixed constant α (since $1/8<1/3<1/2$). Note that we need to fix a "bad" hypothesis to apply this argument. A hypothesis can be fixed as soon as it has appeared in a set of hypotheses. We then exclude the first set in which a hypothesis has appeared when counting the fraction of sets in which the hypothesis has appeared (Chernoff bound is now on $\ell-1$ trials but this is insubstantial). By setting $\ell=(m+\log m+2\log(1/\delta)+3)/\alpha$ we will get that $\ell 2^m 2^{-\alpha\ell-1}\leq\delta/2$. Therefore the probability that a "bad" hypothesis will appear in $1/3$ of the sets is at most $\delta/2$. Similarly all "good" hypotheses will appear in $1/3$ of the sets with probability at least $1-\delta/2$. Thus by picking any a that appears in at least $1/3$ of all the sets we will find a $\theta/2$-heavy coefficient with probability at least $1-\delta$.

Computing each of the Fourier transforms takes $O(m2^m)=\tilde{O}(\theta^{-2}\cdot q(n,k))$ time. They are performed for each of $q(n,k)$ MQs of $\mathcal{B}$ and this is repeated $\ell=O(m+\log(1/\delta))$ times giving the total bound of $\tilde{O}(\theta^{-2}q^2(n,k)\log(1/\delta))$. For each of the 2^m values of z we run $\mathcal{B}(k)$ and tests (i) and (ii).

This takes $O(2^m(t(n,k)+mn)) = \tilde{O}(\theta^{-2}t(n,k)\cdot q(n,k))$ time and is repeated $\ell = O(m+\log(1/\delta))$ times. Therefore the total running time is $\tilde{O}(\theta^{-2}\cdot t(n,k)\cdot q(n,k)\log(1/\delta))$. Similarly we observe that each of the estimations via FFT uses 2^m examples and $\ell\cdot(q(n,k)+1)$ such estimations are done. This implies that the sample complexity of the algorithm is $\tilde{O}\left(\theta^{-2}q^2(n,k)\log(1/\delta)\right)$. It can also be easily seen that all MQs are non-adaptive. ■

Another way to see Theorem 13 is as a way to convert an ae.naMQ algorithm for learning of parities to an ae.naMQ algorithm for agnostic learning of parities.

By plugging $\texttt{AEParityStat}(k)$ algorithm (Theorem 9) into Theorem 13 we obtain our weak parity learning algorithm.

Corollary 14 *There exists an attribute-efficient and non-adaptive weak parity learning algorithm* $\texttt{AEBoundedSieve}(\theta,\texttt{k})$ *that succeeds with probability at least* $1-\delta$*, runs in time* $\tilde{O}\left(nk^2\theta^{-2}\log(1/\delta)\right)$*, and asks* $\tilde{O}\left(k^2\log^2 n\cdot\theta^{-2}\log(1/\delta)\right)$ *MQs.*

Jackson (1997) has proved that for every distribution $\mathcal{D}$, every DNF formula f has a parity function that weakly approximates f with respect to $\mathcal{D}$. A refined version of this claim by Bshouty and Feldman (2002) shows that f has a short parity that weakly approximates f if the distribution is not too far from the uniform. More formally, for a real-valued function ϕ we define $L_\infty(\phi) = \max_x\{|\phi(x)|\}$ and we view a distribution $\mathcal{D}$ as a function over $\{0,1\}^n$ that for a point x gives its probability weight under $\mathcal{D}$.

Lemma 15 *For any Boolean function f of DNF-size s and a distribution $\mathcal{D}$ over $\{0,1\}^n$ there exists a parity function χ_a such that*

$$|\mathbf{E}_{\mathcal{D}}[f\chi_a]| \geq \frac{1}{2s+1} \text{ and } \texttt{wt}(a) \leq \log\left((2s+1)L_\infty(2^n\mathcal{D})\right) .$$

By combining this fact with Corollary 14 we get an algorithm for weakly learning DNF.

Theorem 16 *There exist an algorithm* $\texttt{WeakDNF}_{\mathcal{U}}(s)$ *that for a Boolean function f of DNF-size s given n,s, and access to MEM(f), with probability at least $1/2$, finds a $(\frac{1}{2}-\Omega(\frac{1}{s}))$-approximator to f with respect to $\mathcal{U}$. Furthermore,* $\texttt{WeakDNF}_{\mathcal{U}}(s)$ *runs in time* $\tilde{O}\left(ns^2\right)$ *and asks* $\tilde{O}\left(s^2\log^2 n\right)$ *non-adaptive MQs.*

Proof Lemma 15 implies that there exists a parity χ_a on at most $\log(2s+1)$ variables such that $|\mathbf{E}_{\mathcal{U}}[f\chi_a]| = |\hat{f}(a)| \geq \frac{1}{2s+1}$. This means that f has a $\frac{1}{2s+1}$-heavy Fourier coefficient of degree at most $\log(2s+1)$. Using Corollary 14 for $\delta = 1/2$, we can find a $\frac{1}{2(2s+1)}$-heavy Fourier coefficient $\hat{f}(a')$ in time $\tilde{O}\left(ns^2\right)$ and using $\tilde{O}\left(s^2\log^2 n\right)$ non-adaptive MQs. The parity $\chi_{a'}$ or its negation $(\frac{1}{2}-\frac{1}{4(2s+1)})$-approximates f. ■

The algorithm for weakly learning DNFs by Bshouty et al. (1999) requires $\tilde{O}\left(ns^2\right)$ MQs and runs in time[3] $\tilde{O}\left(ns^2\right)$.

3. The running time bound is based on use of a membership query oracle, that given any two vectors $x,y \in \{0,1\}^n$, passed to it "by reference", returns $f(x\oplus y)$ in $O(1)$ time.

6. Learning DNF Expressions

In this section we show an ae.naMQ algorithm for learning DNF expressions. Following Jackson's approach we first show how to generalize our weak DNF learning algorithm to other distributions (Jackson, 1997). We then use Freund's boosting algorithm to obtain a strong DNF learning algorithm (Freund, 1992). Besides achieving attribute-efficiency and non-adaptiveness we show a way to speed up the boosting process by exploiting several properties of our `WeakDNF` algorithm.

6.1 Weak DNF Learning with Respect to Any Distribution

The first step in Jackson's approach is to generalize a weak parity algorithm to work for any real-valued function. We follow this approach and give a generalization of our $\texttt{AEBoundedSieve}(\theta,\texttt{k})$ algorithm (Corollary 14) to any real-valued and also randomized functions.

Lemma 17 *There exists an algorithm* $\texttt{AEBoundedSieveRV}(\theta,\texttt{k},\texttt{V})$ *that for any real-valued randomized function* Ψ *with a* θ*-heavy Fourier coefficient of degree at most* k*, given* k*,* θ*,* $V \geq \mathbf{Var}_{\mathcal{U},\Psi}(\Psi(x))$*, and an oracle access to* Ψ*, finds, with probability at least* $1-\delta$*, a* $\theta/2$*-heavy Fourier coefficient of* Ψ *of degree at most* k*. The algorithm runs in time* $\tilde{O}\left(nk^2\theta^{-2}V\log(1/\delta)\right)$ *and asks* $\tilde{O}\left(k^2\log^2 n\cdot\theta^{-2}V\log(1/\delta)\right)$ *non-adaptive MQs.*

Proof By revisiting the proof of Theorem 13, we can see that the only place where we used the fact that f is Boolean and deterministic is when relying on Equation (2) in which the variance of the random variable $f(x)\in\{-1,+1\}$ was upper-bounded by 1. In this bound $f(x)$ is already treated as a random variable on pairwise independent x's. For any point x, $\Psi(x)$ is independent of any other evaluations of Ψ and therefore evaluations of Ψ on pairwise independent points are pairwise independent. This implies that in order to estimate $\widehat{\Psi}(a)$ within $\theta/4$ we only need to account for the fact that the variance of $\Psi(x)$ is not necessarily bounded by 1. This can be done by using $\mathbf{Var}(\Psi)\leq V$ times more samples, that is, we set $m=\log(16V\theta^{-2}\cdot 4\cdot(q(n,k)+1)+1)$. It is now straightforward to verify that the rest of the proof of Theorem 13 is unchanged. The increase in the required sample size increases the running time and the sample complexity of the algorithm by a factor $\tilde{O}(V)$ giving us the claimed bounds. ■

As in Jackson's work we use the generalized weak parity algorithm to obtain an algorithm that weakly learns DNF expressions with respect to any distribution. The algorithm is efficient only when the distribution function is "close" to the uniform and requires access to the value of the distribution function at any point x.

Theorem 18 *There exist an algorithm* $\texttt{WeakDNF}(\texttt{s},\texttt{B})$ *that for a Boolean function* f *of DNF-size* s *and any distribution* $\mathcal{D}$*, given* $n,s,B\geq L_\infty(2^n\mathcal{D}(x))$*, access to* $MEM(f)$*, and an oracle access to* $\mathcal{D}$*, with probability at least* $1-\delta$*, finds a* $(\frac{1}{2}-\Omega(\frac{1}{s}))$*-approximator to* f *with respect to* $\mathcal{D}$*. Furthermore,* $\texttt{WeakDNF}(\texttt{s},\texttt{B})$

- *runs in time* $\tilde{O}(ns^2B\log(1/\delta))$*;*
- *asks* $\tilde{O}(s^2\log^2 n\cdot B\log(1/\delta))$ *non-adaptive MQs;*
- *returns a parity function of length at most* $O(\log(sB))$ *or its negation.*

Proof Lemma 15 states that there exists a vector a of Hamming weight bounded by $O(\log(sL_\infty(2^n\mathcal{D})))$ such that $|\mathbf{E}_{\mathcal{D}}[f(x)\chi_a(x)]| = \Omega(1/s)$. But

$$\mathbf{E}_{\mathcal{D}}[f(x)\chi_a(x)] = \sum_x [f(x)\mathcal{D}(x)\chi_a(x)] = \mathbf{E}[f(x)2^n\mathcal{D}(x)\chi_a(x)] = \widehat{\psi}(a) \ , \tag{3}$$

where $\psi(x) = f(x)2^n\mathcal{D}(x)$. This means that $\psi(x)$ has a $\Omega(1/s)$-heavy Fourier coefficient of degree bounded by $O(\log(sL_\infty(2^n\mathcal{D}))) = O(\log(sB))$. We can apply `AEBoundedSieveRV` on $\psi(x)$ to find its $\Omega(1/s)$-heavy Fourier coefficient of degree $O(\log(sB))$. All we need to do this is to provide a bound V on the variance of $f(x)2^n\mathcal{D}(x)$.

$$\begin{aligned}
\mathbf{Var}(f(x)2^n\mathcal{D}(x)) &= \mathbf{E}[(f(x)2^n\mathcal{D}(x))^2] - \mathbf{E}^2[f(x)2^n\mathcal{D}(x)] \\
&\leq L_\infty(2^n\mathcal{D}(x))\mathbf{E}[2^n\mathcal{D}(x)] - \mathbf{E}^2[f(x)2^n\mathcal{D}(x)] \leq L_\infty(2^n\mathcal{D}(x))\mathbf{E}[2^n\mathcal{D}(x)] \\
&= L_\infty(2^n\mathcal{D}(x)) \leq B
\end{aligned} \tag{4}$$

This bound on variance relies essentially on the fact that $\mathcal{D}(x)$ is a distribution function [4] and therefore $\mathbf{E}[2^n\mathcal{D}(x)] = \mathbf{E}_{\mathcal{D}}[1] = 1$. This improves on $L_\infty^2(2^n\mathcal{D}(x))$ bound for an unrestricted function $\mathcal{D}(x)$ that was used in analysis of previous weak DNF learning algorithms (Jackson, 1997; Bshouty et al., 1999).

We can now run `AEBoundedSieveRV`$(\theta, \mathtt{k}, \mathtt{V})$ for $\theta = \Omega(1/s)$, $k = O(\log(sB))$, $V = B$, and a simulated oracle access to $\psi = f2^n\mathcal{D}$ to obtain a' such that $|\widehat{\psi}(a')| = \Omega(1/s)$ and $\mathtt{wt}(a') = O(\log(sB))$. By equation (3), we get that $|\mathbf{E}_{\mathcal{D}}[f(x)\chi_{a'}(x)]| = \Omega(1/s)$ and therefore $\chi_{a'}(x)$ or its negation $(\frac{1}{2} - \Omega(\frac{1}{s}))$-approximates f with respect to $\mathcal{D}$. The claimed complexity bounds can be obtained by using Lemma 17 for θ, k and V as above. ■

6.2 Background on Boosting a Weak DNF Learner

Jackson (1997) obtained his DNF learning algorithm by converting a weak DNF learning algorithm to a strong one via a *boosting* algorithm. *Boosting* is a general technique for improving the accuracy of a learning algorithm. It was introduced by Schapire (1990) who gave the first efficient boosting algorithm. Let $\mathcal{C}$ be a concept class and let $\mathtt{WL}_\gamma$ be a weak learning algorithm for $\mathcal{C}$ that for any distribution $\mathcal{D}$, produces a $(1/2 - \gamma)$-approximating hypothesis. Known boosting algorithms have the following structure.

- At stage zero $\mathtt{WL}_\gamma$ is run on $\mathcal{D}_0 = \mathcal{D}$ to obtain h_0.
- At stage i a distribution $\mathcal{D}_i$ is constructed using $\mathcal{D}$ and previous weak hypotheses $h_0, \ldots, h_{i-1}$. The distribution $\mathcal{D}_i$ usually favors the points on which the previous weak hypotheses do poorly. Then random examples from $\mathcal{D}_i$ are simulated to run $\mathtt{WL}_\gamma$ with respect to $\mathcal{D}_i$ and obtain h_i.
- After repeating this for a number of times an ε-approximating hypothesis h is created using all the generated weak hypotheses.

4. Actual $\mathcal{D}(x)$ given to a weak learner will be equal to $c\mathcal{D}'(x)$ where $\mathcal{D}'(x)$ is a distribution and c is a constant in $[2/3, 4/3]$ (Bshouty et al., 1999). This modifies the bound above by a small constant factor.

Jackson's use of Freund's boosting algorithm slightly deviates from this scheme as it provides the weak learner with the oracle that returns the density of the distribution function $\mathcal{D}_i$ at any desired point instead of simulating random examples with respect to $\mathcal{D}_i$. The `WeakDNF` algorithm also requires oracle access to $\mathcal{D}_i(x)$ and therefore we will use a boosting algorithm in the same way. The running time of Jackson's (and our) algorithm for weak learning of DNF expression depends polynomially on $L_\infty(2^n\mathcal{D})$ and therefore it can only be boosted by a boosting algorithm that produces distributions that are *polynomially-close* to the uniform distribution; that is, the distribution function is bounded by $p2^{-n}$ where p is a polynomial in learning parameters (such boosting algorithms are called *p-smooth*). In Jackson's result Freund's (1990) boost-by-majority algorithm is used to produce distribution functions bounded by $O(\varepsilon^{-2})$. More recently, Klivans and Servedio (2003) have observed that a later boosting algorithm of Freund (1992) produces distribution functions bounded by $\tilde{O}(1/\varepsilon)$, thereby improving the dependence of running time and sample complexity on ε. This improvement together with improved weak DNF learning algorithm of Bshouty et al. (1999) gives DNF learning algorithm that runs in $\tilde{O}(ns^6/\varepsilon^2)$ time and has sample complexity of $\tilde{O}(ns^4/\varepsilon^2)$.

Remark 19 *Bshouty et al. claimed sample complexity of $\tilde{O}(ns^2/\varepsilon^2)$ based on erroneous assumption that sample points for weak DNF learning can be reused across boosting stages. A distribution function $\mathcal{D}_i$ in i-th stage depends on hypotheses produced in previous stages. The hypotheses depend on random sample points and therefore in i-th stage the same set of sample points cannot be considered as chosen randomly and independently of $\mathcal{D}_i$ (Jackson, 2004). This implies that new and independent points have to be sampled for each boosting stage and increases the sample complexity of the algorithm by Bshouty et al. by a factor of $O(s^2)$.*

As in the work of Klivans and Servedio (2003), we use Freund's (1992) `B-Comb` boosting algorithm to boost the accuracy of our weak DNF learning algorithm. We will now briefly describe the `B-Comb` boosting algorithm (see also the work of Klivans and Servedio (2003) for a detailed discussion on application of `B-Comb` to learning DNF expressions).

6.2.1 FREUND'S `B-Comb` BOOSTING ALGORITHM

`B-Comb` boosting algorithm is based on a combination of two other boosting algorithms. The first one in an earlier `F1` algorithm due to Freund (1990) and is used to boost from accuracy $\frac{1}{2}-\gamma$ to accuracy $1/4$. Its output is the function equal to the majority vote of the weak hypotheses that it received. This algorithm is used as a weak learner by the second boosting algorithm `B-Filt`. At stage k `B-Filt` sets h_ℓ to be either the output of a weak learner or a random coin flip (that is a randomized function equal to either 1 or -1, each with probability $1/2$). Accordingly the distribution function generated at stage i depends on random coin flips and the final hypothesis is a majority vote over hypotheses from the weak learner and random coin flips. As it is done by Freund (1992), we analyze the algorithm for a fixed setting of these coin flip hypotheses. Freund's analysis shows that with overwhelming probability over the coin flips the randomized hypothesis produced by the boosting algorithm ε-approximates the target function.

Each of the executions of `F1` has $O(\gamma^{-2})$ stages and `B-Filt` has $O(\log(1/\varepsilon))$ stages. We denote the distribution function generated at stage i of `F1` during stage ℓ of `B-Filt` as $\mathcal{D}^{\texttt{Comb}}_{\ell,i}$. In both boosting algorithms $\mathcal{D}_i(x) = \beta(i, N(x))\mathcal{D}/\alpha$, where $N(x)$ is the number of previous hypotheses that are correct on x, β is a fixed function from a pair of integers to the interval $[0,1]$ computable in polynomial (in the length of its input) time, and α is the normalization factor equal to $\mathbf{E}_{\mathcal{D}}[\beta(i, N(x))]$.

We can therefore say that

$$\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x) = \beta(\ell, N_{\texttt{Filt}}(x)) \cdot \beta(i, N_{\texttt{F1}}(x))\mathcal{D}(x)/(\alpha_\ell \alpha_{\ell,i}) \ , \tag{5}$$

where $N_{\texttt{Filt}}(x)$ and $N_{\texttt{F1}}(x)$ count the correct hypotheses so far for `B-Filt` and `F1` respectively. The normalization factor α_ℓ equals $\mathbf{E}_{\mathcal{D}}[\beta(\ell, N_{\texttt{Filt}}(x))]$ and

$$\alpha_{\ell,i} = \mathbf{E}_{\mathcal{D}}[\beta(\ell, N_{\texttt{Filt}}(x)) \cdot \beta(i, N_{\texttt{F1}}(x))\mathcal{D}(x)/\alpha_\ell] \ .$$

The analysis by Freund implies that for every ℓ and i,

$$L_\infty(2^n \mathcal{D}^{\texttt{Comb}}_{\ell,i}) \leq 1/(\alpha_\ell \alpha_{\ell,i}) = \tilde{O}(1/\varepsilon) \ .$$

In Figure 6.2.1 we include the pseudocode of `B-Comb` algorithm simplified and adapted to our setting.

6.3 Optimized Boosting

We now use Freund's (1992) `B-Comb` boosting algorithm to boost the accuracy of our weak DNF learning algorithm. Unlike in the previous work, we will exploit several properties of `WeakDNF` to achieve faster execution of each boosting stage. Specifically, we note that evaluation of the distribution function $\mathcal{D}_i(x)$ at boosting stage i involves evaluation of $i-1$ previous hypotheses on x and therefore, in a general case, for a sample of size q will require $\Omega(i \cdot q)$ steps, making the last stages of boosting noticeably slower. Our goal is to show that for our `WeakDNF` algorithm and the `B-Comb` boosting algorithm the evaluation of $\mathcal{D}_i(x)$ for the whole sample needed by `WeakDNF` can be made more efficiently.

The idea of the speed-up is to use Equation (5) together with the facts that weak hypotheses are parities and MQs of `WeakDNF` come from a "small" number of low-dimension linear subspaces. Let g be a function that is equal to a linear combination of short parity functions. We start by showing a very efficient way to compute the values of g on a linear subspace of $\{0,1\}^n$. We will assume that vectors of Hamming weight at most w are represented by the list of indices where the vector is equal to 1 (as we did for parities). One can easily see that adding such vectors or multiplying them by any vector takes $O(w \log n)$ time.

Lemma 20 *Let $\{c_1, c_2, \ldots, c_i\}$ be a set of vectors in $\{0,1\}^n$ of Hamming weight at most w; $\bar{\alpha} \in \mathbb{R}^i$ be a real-valued vector, and R be a m-by-n 0-1 matrix. Then the set of pairs*

$$S = \{\langle p, \sum_{j \leq i} \alpha_j \chi_{c_j}(pR) \rangle \mid p \in \{0,1\}^m\}$$

can be computed in time $\tilde{O}(i \cdot w \log n + 2^m)$.

Proof We define $g(x) = \sum_{j \leq i} \alpha_j \chi_{c_j}(x)$ and for $p \in \{0,1\}^m$ we define $g_R(p) = g(pR)$ (as in Sect. 5). Our goal is to find the values of function g_R on all the points of $\{0,1\}^m$. The function g is given as a linear combination of parities, or in other words, we are given its Fourier transform. Given the Fourier transform of g we can derive the Fourier transform of g_R from the following equation:

$$g_R(p) = \sum_{j \leq i} \alpha_j \chi_{c_j}(pR) = \sum_{j \leq i} \alpha_j \chi_{c_j R^T}(p) = \sum_{z \in \{0,1\}^m} \left[(\sum_{j \leq i;\ c_j R^T = z} \alpha_j) \chi_z(p) \right] \ .$$

```
B-Comb(ε, δ, 𝒟, WL_γ)
k ← c_0 log(1/ε)
Θ ← c_1 ε/log(1/ε)
h_0 ← F1(1/4, δ/(2k+1), 𝒟, WL_γ)
for ℓ ← 1 to k
    N(x) ≡ |{h_j | 0 ≤ j ≤ ℓ−1 and h_j(x) = f(x)}|
    α'_ℓ ← EstExpRel(β(ℓ, N(x)), 𝒟, 1/3, δ/(2k+1))
    if α'_ℓ ≥ Θ then
        𝒟'_ℓ ≡ β(ℓ, N(x))/α'_ℓ
        h_ℓ ← F1(1/4, δ/(2k+1), 𝒟'_ℓ, WL_γ)
    else
        h_ℓ ← Random(1/2)
end for
return Majority(h_0, h_1, ..., h_k)
```

```
F1(ε, δ, 𝒟, WL_γ)
k ← c_2/γ^2
Θ ← c_3 ε^2
h_0 ← WL_γ(𝒟, δ/(2k+1))
for i ← 1 to k
    N(x) ≡ |{h_j | 0 ≤ j ≤ i−1 and h_j(x) = f(x)}|
    α'_i ← EstExpRel(β(i, N(x)), 𝒟, 1/3, δ/(2k+1))
    if α'_i ≥ Θ then
        𝒟'_i ≡ β(i, N(x))/α'_i
        h_i ← WL_γ(𝒟'_i, δ/(2k+1))
    else
        k ← i−1
        break for
end for
return Majority(h_0, h_1, ..., h_k)
```

Figure 1: Pseudocode of B-Comb boosting algorithm. The first part is B-Filt with F1 used as a weak learner. $\mathtt{WL}_\gamma$ is a weak learning algorithm that has accuracy $\frac{1}{2}-\gamma$ and takes an oracle for a distribution $\mathcal{D}$ and confidence δ as parameters. $\mathtt{EstExpRel}(R,\mathcal{D},\lambda,\delta)$ produces estimates of the expectation of a random variable R with respect to a distribution $\mathcal{D}$ within relative accuracy λ and confidence δ (that is the estimate $v' \in [(1-\lambda)v,(1+\lambda)v]$, where v is the true expectation). Various unspecified constants are denoted by $c_0, c_1, \ldots$ The membership query oracle for the target function f is available to all procedures.

Hence $\widehat{g_R}(z) = \sum_{j \le i;\ c_j R^T = z} \alpha_j$. Given the Fourier transform of g_R we can use the FFT algorithm to perform the inverse Fourier transform of g_R giving us the desired values of $g_R(p)$ on all the points of $\{0,1\}^m$. This task can be performed in $O(m2^m)$ steps. To compute the Fourier transform of g_R we need to compute $c_j R^T$ for each $j \le i$ and sum the ones that correspond to the same z. Given that each c_j is of Hamming weight w, $c_j R^T$ can be computed in $O(wm \log n)$ steps (note that we do not read the entire matrix R). Therefore the computation of the Fourier transform and the inversion using the FFT algorithm will take $O(m(iw \log n + 2^m)) = \tilde{O}(i \cdot w \log n + 2^m)$ steps. ■

Note that a straightforward computation would take $\Omega(iw2^m \log n)$ steps. We apply Lemma 20 to speed up the evaluation of $\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x)$ on points at which `WeakDNF` asks non-adaptive MQs (here again we will rely on the non-adaptiveness of the weak learning algorithm). The speed-up is based on the following observations.

1. `WeakDNF` is based on estimating Fourier coefficients on a "small" number of linear subspaces of $\{0,1\}^n$ (as in Equation 2).

2. `WeakDNF` produces a short parity function (or its negation) as the hypothesis.

3. In computation of $\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x)$ the only information that is needed about the previous hypotheses is $N_{\texttt{Filt}}(x)$ and $N_{\texttt{F1}}(x)$, that is the number of hypotheses so far that are correct on the given point. The number of correct hypotheses is determined by $f(x)$ and the sum (in particular, a linear combination) of the values of the hypotheses on x.

Now we prove these observations formally and show a more efficient way to compute $\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x)$ given oracle access to $N_{\texttt{Filt}}(x)$, in other words, we show a more efficient way to compute $N_{\texttt{F1}}(x)$.

Lemma 21 *Let $\{b_1\chi_{c_1}, b_2\chi_{c_2}, \ldots, b_i\chi_{c_i}\}$ be the set of hypotheses returned by* `WeakDNF`(s,B) *in i first stages of* `F1` *boosting algorithm during stage ℓ of* `B-Filt`*, where $b_j \in \{-1,+1\}$ is the sign of χ_{c_j} (indicating whether or not it is negated). Let W be the set of queries for the $(i+1)$-th execution of* `WeakDNF`(s,B) *with confidence parameter δ and $B \ge L_\infty(2^n \mathcal{D}^{\texttt{Comb}}_{\ell,i})$. Then, given MEM$(f)$ and an oracle access to $N_{\texttt{Filt}}(x)$, the set of pairs $S = \{\langle x, \lambda \mathcal{D}^{\texttt{Comb}}_{\ell,i}(x)\rangle \mid x \in W\}$ for some constant $\lambda \in [2/3, 4/3]$, can be computed, with probability at least $1-\delta$, in time $\tilde{O}((i+s^2B)\log^2 n \log(1/\delta))$.*

Proof We start by proving our first observation. By revisiting the proof of Theorem 13 we can see that our weak parity algorithm asks queries on $Y = \{pR \mid p \in \{0,1\}^m\}$ for a randomly chosen R and then for each query z of a ae.naMQ parity algorithm it asks queries on points of the set $Y_z = \{z \oplus y \mid y \in Y\}$. The set Y_z is a subset of the linear subspace of dimension $m+1$ spanned by the rows of R and vector y. These queries are then repeated $O(m + \log(1/\delta))$ times to single out "good" Fourier coefficients. Therefore by substituting the parameters of `WeakDNF`(s,B) into the proofs of Lemma 17 and Theorem 13, we can see that W can be decomposed into $\tilde{O}(\log^2(sB)\log n \log(1/\delta))$ linear subspaces of dimension $m = \log T$ for $T = \tilde{O}(s^2 B \log n)$.

Our second observation is given by Theorem 18 and states that for each $j \le i$, χ_{c_j} is a parity on at most $\log sB$ variables.

Our next observation is that the number of hypotheses from $\{b_1\chi_{c_1}, b_2\chi_{c_2}, \ldots, b_i\chi_{c_i}\}$ that agree with f on x equals to

$$N_{\texttt{F1}}(x) = \frac{f(x)\left(\sum_{j \le i} b_j \chi_{c_j}(x)\right)}{2} + \frac{i}{2}\ ,$$

that is, given $\sum_{j\le i} b_j\chi_{c_j}(x)$ and $f(x)$, $N_{\mathtt{F1}}(x)$ can be computed in $O(1)$ steps. According to Lemma 20, we can compute $\sum_{j\le i} b_j\chi_{c_j}(x)$ on a linear subspace of dimension m in time $\tilde{O}(iw\log n+2^m)$. Together with the first observation this implies that computing $N_{\mathtt{F1}}(x)$ for all points in W can be done in time

$$\tilde{O}(\log^2(sB)\log n\log(1/\delta))\tilde{O}(i\cdot\log(sB)\log n+s^2B\log n)=\tilde{O}((i+s^2B)\log^2 n\log(1/\delta))\ .$$

Equation (5) implies that for every point x, given $N_{\mathtt{F1}}(x)$, oracle access to $N_{\mathtt{Filt}}(x)$ and $\alpha_\ell\alpha_{\ell,i}$ we obtain $\mathcal{D}^{\mathtt{Comb}}_{\ell,i}(x)$. The normalization factor $\alpha_{\ell,i}$ is estimated with relative accuracy $1/3$ and therefore instead of the true $\mathcal{D}^{\mathtt{Comb}}_{\ell,i}(x)$ we will obtain $\lambda\mathcal{D}^{\mathtt{Comb}}_{\ell,i}(x)$ for some constant $\lambda\in[2/3,4/3]$. ■

Lemma 21 assumes oracle access to $N_{\mathtt{Filt}}(x)$. In the next lemma we show that this oracle can be simulated efficiently.

Lemma 22 *Let $\{h_0,h_1,\ldots,h_{\ell-1}\}$ be the set of hypotheses obtained by* B-Comb *in ℓ first stages of boosting. Let W be the set of queries for the $(i+1)$-th execution of* WeakDNF(s,B) *with confidence parameter δ and $B\ge L_\infty(2^n\mathcal{D}^{\mathtt{Comb}}_{\ell,i})$. Then, given $MEM(f)$, the set of pairs $S=\{\langle x,N_{\mathtt{Filt}}(x)\rangle \mid x\in W\}$ can be computed, with probability at least $1-\delta$, in time $\tilde{O}(\ell s^2B\cdot\log^2 n\log(1/\delta))$.*

Proof For each $j\le\ell-1$, h_j is an output of F1 or a random coin flip hypothesis. WeakDNF(s,B) returns $(\frac{1}{2}-\Omega(\frac{1}{s}))$-approximate hypotheses and therefore each hypothesis generated by F1 is a majority vote of $O(\gamma^-2)=O(s^2)$ short parities (or their negations). A majority vote of these parities and their negations is simply the sign of their sum, and in particular is determined by a linear combination of parity functions. Hence, as in Lemma 21, $h_j(x)$ for all points in W can be computed $\tilde{O}((s^2+s^2B)\log^2 n\log(1/\delta))$ time. Therefore for any stage ℓ, $h_0,h_1,\ldots,h_{\ell-1}$ can be computed on points in W in $\tilde{O}(\ell s^2B\log^2 n\log(1/\delta))$ steps giving the required oracle $N_{\mathtt{Filt}}(x)$. ■

Remark 23 *In this simulation of* B-Comb *we ignored the complexity of procedure* EstExpRel *that is used to evaluate the normalization factors. The factor $\alpha_\ell=\mathbf{E}[\beta(\ell,N_{\mathtt{Filt}}(x))]$ needs to be estimated within relative accuracy $1/3$ and its value is only used when the estimate $\alpha'_\ell\ge\Theta=c_1\varepsilon/\log(1/\varepsilon)$ for some constant c_1 since otherwise* B-Comb *uses a random coin flip hypothesis (see line 7 of the pseudocode). This implies that the estimate is only used when $\alpha_\ell\ge 3\Theta/4$. The Chernoff bound (Lemma 1) implies that if $\alpha_\ell\ge 3\Theta/4$ then using $M=O(\frac{log(1/\varepsilon)\log(1/\delta)}{\varepsilon})$ random uniform samples will be sufficient to estimate α_ℓ within relative accuracy $1/3$ with confidence $1-\delta$. If $\alpha_\ell<3\Theta/4$ then with probability $1-\delta$ the obtained estimate α'_ℓ will be less than Θ and therefore will not be used. Evaluating $N_{\mathtt{Filt}}(x)$ on each of these points will take $O(\ell ns^2)$ steps and therefore each of these estimation will run in time $\tilde{O}(\ell ns^2\log(1/\delta)/\varepsilon)$.*

At each stage of the F1 *boosting algorithm we need to estimate*

$$\alpha_{\ell,i}=\mathbf{E}[\beta(\ell,N_{\mathtt{Filt}}(x))\cdot\beta(i,N_{\mathtt{F1}}(x))/\alpha'_\ell]$$

to within relative accuracy $1/3$ and its value is only used when the estimate $\alpha'_{\ell,i}\ge c$ for some constant c. Therefore it is sufficient to estimate $\alpha_{\ell,i}$ to within constant additive accuracy. With probability at least $1-\delta$ this can be achieved by using a sample of $O(\log(1/\delta))$ random uniform points. Estimating both $N_{\mathtt{Filt}}(x)$ and $N_{\mathtt{F1}}(x)$ on each point takes $O(\ell ns^2)$ steps and therefore each of these estimations runs in time $O(\ell ns^2\log(1/\delta))$.

We are now ready to describe the resulting ae.naMQ algorithm for learning DNF expressions.

Theorem 24 *There exists an algorithm* AENALearnDNF(s) *that for any Boolean function f of DNF-size s, given n, s, ε, and access to MEM(f), with probability at least $1/2$, finds an ε-approximator to f with respect to $\mathcal{U}$. Furthermore,* AENALearnDNF(s) *runs in time $\tilde{O}\left(ns^4/\varepsilon\right)$ and asks $\tilde{O}\left(s^4 \log^2 n/\varepsilon\right)$ non-adaptive MQs.*

Proof As we know from the description of B-Filt, it has $O(\log(1/\varepsilon))$ stages and for each ℓ and i, $L_\infty(2^n \mathcal{D}^{\mathtt{Comb}}_{\ell,i}) = \tilde{O}(1/\varepsilon)$. Therefore the running time of each execution of WeakDNF(s,B) is $\tilde{O}(ns^2/\varepsilon)$. In particular, for every boosting stage of F1, it dominates the running time of computing the distribution function $\mathcal{D}^{\mathtt{Comb}}_{\ell,i}$ (Lemmas 21 and 22) and estimations of α_ℓ and $\alpha_{\ell,i}$ (Remark 23). There are total $O(s^2 \log(1/\varepsilon))$ executions of WeakDNF and therefore the total running time of AENALearnDNF(s) is $\tilde{O}\left(ns^4/\varepsilon\right)$ and the total number of non-adaptive MQs used is $\tilde{O}\left(s^4 \log^2 n/\varepsilon\right)$. ■

The improvements to the algorithm by Bshouty et al. (1999) are summarized below.

- The use of attribute-efficient weak learning improves the total sample complexity from $\tilde{O}\left(ns^4/\varepsilon^2\right)$ to $\tilde{O}\left(s^4 \log^2 n/\varepsilon^2\right)$ and the same running time is achieved without assumptions on the MQ oracle (see Theorem 16).
- Faster computation of distribution functions used in boosting improves the total running time from $\tilde{O}\left(ns^6/\varepsilon^2\right)$ to $\tilde{O}\left(ns^4/\varepsilon^2\right)$ (see Lemmas 20, 21 and 22).
- Tighter estimation of variance improves the dependence of running time and sample complexity on ε from $1/\varepsilon^2$ to $1/\varepsilon$ (Equation 4).

Remark 25 *While the analysis of the speedup was done for Freund's* B-Comb *booster the same idea works for any other booster in which estimation of new weight function is based on a linear combination of previous hypotheses. In particular, for the other known boosting algorithms that produce smooth distributions:* SmoothBoost *by Servedio (2003) and* AdaFlat *by Gavinsky (2003).*

7. Handling Noise

Now we would like to show that our DNF learning algorithm can be modified to tolerate random persistent classification noise in MQs. To simplify the proof we first show that we can assume that we are dealing with random and independent classification noise.

Lemma 26 *The probability that* AENALearnDNF(s) *asks an MQ for the same point more than once is upper bounded by $P \cdot 2^{-n/\log Q}$ where P and Q are polynomial in n, s and $1/\varepsilon$.*

Proof We start by observing that in the algorithm AENALearnDNF(s) all the points that are given to the MQ oracle are chosen uniformly and the points that are used in different executions of WeakDNF are independent. As can be seen from the proof of Theorem 13, the generated points are of the form $pR \oplus y$, where R is a randomly and uniformly chosen matrix, y is chosen randomly according to $\mathcal{D}_{\frac{1}{4k}}$ (defined in Theorem 9) or equal to 0^n, and $p \in \{0,1\}^m$. Points generated for two randomly chosen R_1 and R_2 are independent of each other and uniformly distributed. Let $y_0 = 0^n$, q be the number of samples taken from $\mathcal{D}_{\frac{1}{4k}}$, and $y_1, y_2, \ldots, y_q$ denote the samples.

For some randomly chosen R, let $x_1 = p_1 R \oplus y_i$ and $x_2 = p_2 R \oplus y_j$ be two different sample points. For two different sample points either $i \neq j$ or $p_1 \neq p_2$. If $i \neq j$ then either $i \neq 0$ or $j \neq 0$. Without loss of generality we assume that $i \neq 0$. Then

$$\mathbf{Pr}_{y_i \sim \mathcal{D}_{\frac{1}{4k}}}[p_1 R \oplus y_i = p_2 R \oplus y_j] = \mathbf{Pr}_{y_i \sim \mathcal{D}_{\frac{1}{4k}}}[y_i = p_1 R \oplus p_2 R \oplus y_j] \leq (1 - \frac{1}{4k})^n \leq e^{-n/(4k)} .$$

If $p_1 \neq p_2$ then $\mathbf{Pr}_{R \sim \mathcal{U}_{m \times n}}[(p_1 \oplus p_2)R = y_i \oplus y_j] = 2^{-n}$. This implies that for any two MQs made by $\texttt{AENALearnDNF}(s)$, probability that they are equal is at most $e^{-n/(4k)}$. As it can be seen from the analysis of $\texttt{AENALearnDNF}(s)$, $k = O(\log(s/\varepsilon))$ and the total number of MQs used is polynomial in n, s and $1/\varepsilon$. ■

If an algorithm does not ask a MQ for the same point again then persistent classification noise can be treated as random and independent.

7.1 Boosting Weak Parity Learning Algorithm in the Presence of Noise

The main part of the modification is to show an algorithm that can locate heavy Fourier coefficients of any randomized function can be used to learn DNFs in the presence of noise. Our method can be applied in more general setting. In particular, it could be used to prove that Jackson's original algorithm is resistant to persistent noise in MQs and was recently used to produce a noise tolerant DNF learning algorithm by Feldman et al. (2006). Previous methods to produce noise-tolerant DNF learning algorithms gave statistical query analogues of Jackson's algorithm and then simulated statistical queries[5] in the presence of noise (Jackson et al., 1997; Bshouty and Feldman, 2002). Our approach is more direct and the resulting algorithm is substantially more efficient than the previous ones.

The goal of a weak DNF learning algorithm at stage i of boosting is to find a parity correlated with the function $2^n \mathcal{D}_i(x) f(x)$ given an oracle access to values of $\mathcal{D}_i(x)$ and the oracle for f with noise of rate $\eta < 1/2$ instead of $\mathrm{MEM}(f)$. Handling the noisy case is further complicated by the fact that the computation of $\mathcal{D}_i(x)$ by the boosting algorithm uses the value $f(x)$ (in particular, `B-Comb` and `B-Filt` need the value of $f(x)$ to compute $N(x)$) which is not available in the noisy case. To make this dependence explicit we define $\mathcal{D}_i(x,b)$ (for $b \in \{-1,+1\}$) to be the value of $\mathcal{D}_i$ on x when the boosting algorithm is supplied with the value b in place of $f(x)$ to compute $D_i(x)$ (in particular, $\mathcal{D}_i(x) = \mathcal{D}_i(x, f(x))$). We will now show a general method to compute a Fourier coefficient of a function that depends on $f(x)$ given a noisy oracle for f.

Lemma 27 *Let $g(x,b)$ be any real-valued function over $\{0,1\}^n \times \{-1,+1\}$ and let Φ^η denote a randomized function such that for every x, $\Phi^\eta(x) = f(x)$ with probability $1-\eta$ and $\Phi^\eta(x) = -f(x)$ with probability η. Then for each $a \in \{0,1\}^n$, $[\widehat{g(x, f(x))}](a) = \widehat{\Psi_{g,\eta}}(a)$, where $\Psi_{g,\eta}$ is a randomized function defined as*

$$\Psi_{g,\eta}(x) = \frac{1}{2}\left(\frac{1}{1-2\eta}(g(x,1) - g(x,-1)) \cdot \Phi^\eta(x) + g(x,1) + g(x,-1)\right) .$$

5. They used stronger versions of statistical queries than those introduced by Kearns (1998).

Proof We use the following observation due to Bshouty and Feldman (2002). For any real-valued function $\psi(x,b)$

$$\psi(x,f(x)) = \psi(x,-1)\frac{1-f(x)}{2} + \psi(x,1)\frac{1+f(x)}{2} =$$
$$\frac{1}{2}((\psi(x,1)-\psi(x,-1))f(x) + \psi(x,1) + \psi(x,-1)) \ .$$

Then

$$\mathbf{E}_{x,\Phi^{\eta}(x)}[\frac{1}{2}(\psi(x,1)-\psi(x,-1))\cdot\Phi^{\eta}(x)] = (1-2\eta)\mathbf{E}_x[\frac{1}{2}(\psi(x,1)-\psi(x,-1))f(x)] \ ,$$

and therefore we can offset the effect of noise in $g(x,f(x))$ as follows.

$$\begin{aligned}
[\widehat{g(x,f(x))}](a) &= \mathbf{E}[g(x,f(x))\chi_a(x)] \\
&= \frac{1}{2}\left(\mathbf{E}_x[(g(x,1)-g(x,-1))\chi_a(x)f(x)] + \mathbf{E}_x[(g(x,1)+g(x,-1))\chi_a(x)]\right) \\
&= \frac{1}{2}\left(\frac{1}{1-2\eta}\mathbf{E}_{x,\Phi^{\eta}(x)}[(g(x,1)-g(x,-1))\chi_a(x)\cdot\Phi^{\eta}(x)] + \mathbf{E}_x[(g(x,1)+g(x,-1))\chi_a(x)]\right) \\
&= \mathbf{E}_{x,\Phi^{\eta}(x)}\left[\frac{1}{2}\left(\frac{1}{1-2\eta}(g(x,1)-g(x,-1))\cdot\Phi^{\eta}(x) + g(x,1)+g(x,-1)\right)\chi_a(x)\right] = \widehat{\Psi}(a)
\end{aligned}$$

■

An oracle for $\Phi^{\eta}(x)$ is exactly the membership query oracle for $f(x)$ with noise of rate η that is given to us (by Lemma 26 we can ignore the persistency of noise). Therefore Lemma 27 gives a way to find heavy Fourier coefficients using an oracle for $\Phi^{\eta}(x)$ instead of the membership query oracle for $f(x)$. We apply it to `WeakDNF` and obtain our noise-tolerant ae.naMQ DNF learning algorithm.

Theorem 28 *There exists an algorithm* $\texttt{AENALearnDNF}(s,\eta)$ *that for any Boolean function f of DNF-size s, given n,s,η,ε, and access to MEM(f) corrupted by random persistent classification noise of rate η, with probability at least $1/2$, finds an ε-approximator to f with respect to $\mathcal{U}$. Furthermore,* $\texttt{AENALearnDNF}(s,\eta)$ *runs in time* $\tilde{O}\left(ns^4/(\varepsilon(1-2\eta)^2)\right)$ *and asks* $\tilde{O}\left(s^4\log^2 n/(\varepsilon(1-2\eta)^2)\right)$ *non-adaptive MQs.*

Proof Section 6.3 gives a way to efficiently compute $\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x)$ given the label $f(x)$. This computation defines the oracle for $\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x,b)$ where b is the supposed label of $f(x)$. Let $g(x,b) = b\cdot 2^n\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x,b)$ and let $\Psi_{g,\eta}(x)$ be defined as in Lemma 27. Given the oracle for $\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x,b)$ and oracle access to $\Phi^{\eta}(x)$ we use $\texttt{AEBoundedSieveRV}(\theta,\texttt{k},\texttt{V})$ on $\Psi_{g,\eta}(x)$ in the same way it was used on $\psi(x)$ by $\texttt{WeakDNF}(s,B)$ (see the proof of Theorem 18). By Lemma 27, $\Psi_{g,\eta}(x)$ has the same Fourier coefficients as $f(x)2^n\mathcal{D}^{\texttt{Comb}}_{\ell,i}(x,f(x))$. Therefore this modified weak learning algorithm will produce an equivalent hypothesis. We can deal with the noise while estimating the normalization factor $\alpha_{\ell,i}$ in exactly the same way.

Furthermore, the definition of $\Psi_{g,\eta}$ and Equation (4) imply that

$$L_\infty(\Psi_{g,\eta}) \le \frac{2}{1-2\eta}L_\infty(2^n\mathcal{D}^{\texttt{Comb}}_{\ell,i}) \text{ and } \mathbf{Var}(\Psi_{g,\eta}) \le \frac{4}{(1-2\eta)^2}L_\infty(2^n\mathcal{D}^{\texttt{Comb}}_{\ell,i}) \ .$$

By substituting these bounds into Theorem 18 we obtain that the running time and the sample complexity of each execution of the modified weak learner will grow by $(1-2\eta)^2$. They also imply that $\texttt{WeakDNF}(s,B)$ will produce parities on $\log(s^2/(\varepsilon(1-2\eta)))$ variables (this change is absorbed by $\tilde{O}$ notation). ■

8. Conclusions and Open Problems

In this work we have demonstrated equivalence of attribute-efficient learning of parities from random and uniform examples and decoding of random linear binary codes. This result appears to be the only known evidence of hardness of attribute-efficient learning for a natural concept class. Many other problems remain open in this area. For example it is unknown whether decision lists or linear thresholds are learnable attribute-efficiently.

Our results show that some of the most important concepts classes that are learnable attribute efficiently with respect to the unform distribution using membership queries are also learnable by significantly weaker non-adaptive MQs. We believe that it is interesting to understand if similar results can be obtained in the distribution-independent setting. In particular whether monotone DNF formulae and decision trees can be learned attribute-efficiently using non-adaptive MQs in the distribution-independent PAC model.

We have also shown an improved algorithm for learning DNF expressions with respect to the uniform distribution. In addition to being the most efficient known algorithm for learning DNF, it is attribute-efficient, noise tolerant, and uses membership queries non-adaptively. All known efficient algorithms for learning DNF are based on Jackson's (1997) approach to learning DNF expressions. It would be interesting to find other approaches to learning DNF, possibly avoiding some of the overheads of the current approach (such as boosting a weak DNF learning algorithm).

Acknowledgments

We thank Leslie Valiant for his advice and encouragement of this research. We are grateful to Jeffrey Jackson for discussions and clarifications on the DNF learning algorithm of Bshouty et al.. We also thank Alex Healy, Dmitry Gavinsky, and anonymous COLT and JMLR reviewers for valuable and insightful comments.

References

A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley, 1974.

D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2:343–370, 1988.

A. Barg. Complexity issues in coding theory. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(046), 1997.

A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *Journal of Computer and System Sciences*, 50:32–40, 1995.

A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *Proceedings of STOC*, pages 435–440, 2000.

N. Bshouty and V. Feldman. On using extended statistical queries to avoid membership queries. *Journal of Machince Learning Research*, 2:359–395, 2002.

N. Bshouty and L. Hellerstein. Attribute efficient learning with queries. *Journal of Computer and System Sciences*, 56:310–319, 1998.

N. Bshouty, J. Jackson, and C. Tamon. More efficient PAC learning of DNF with membership queries under the uniform distribution. In *Proceedings of COLT*, pages 286–295, 1999.

N. Bshouty, E. Mossel, R. O'Donnell, and R. Servedio. Learning DNF from random walks. In *Proceedings of FOCS*, pages 189–199, 2003.

H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statist.*, 23:493–507, 1952.

J. Cooley and J. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Computat.*, 19:297–301, 1965.

P. Damaschke. Adaptive versus nonadaptive attribute-efficient learning. In *Proceedings of STOC*, pages 590–596, 1998.

S. Decatur, O. Goldreich, and D. Ron. Computational sample complexity. *SIAM Journal on Computing*, 29(3):854–879, 1999.

M. Farach, S. Kannan, E. Knill, and S. Muthukrishnan. Group testing problems in experimental molecular biology. In *Proceedings of Sequences '97*, 1997.

V. Feldman, P. Gopalan, S. Khot, and A. Ponuswami. New Results for Learning Noisy Parities and Halfspaces. In *Proceedings of FOCS*, pages 563–574, 2006.

Y. Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216, 1990.

Y. Freund. An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 391–398, 1992.

D. Gavinsky. Optimally-smooth adaptive boosting and application to agnostic learning. *Journal of Machine Learning Research*, 4:101–117, 2003.

S. Goldman, M. Kearns, and R. Schapire. Exact identification of read-once formulas using fixed points of amplification functions. *SIAM Journal on Computing*, 22(4):705–726, 1993.

O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of STOC*, pages 25–32, 1989.

O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

D. Guijarro, V. Lavin, and V. Raghavan. Exact learning when irrelevant variables abound. In *Proceedings of EuroCOLT '99*, pages 91–100, 1999a.

D. Guijarro, J. Tarui, and T. Tsukiji. Finding relevant variables in PAC model with membership queries. *Lecture Notes in Artificial Intelligence*, 1720:313 – 322, 1999b.

D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

T. Hofmeister. An application of codes to attribute-efficient learning. In *Proceedings of EuroCOLT*, pages 101–110, 1999.

J. Jackson. Personal communication, 2004.

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

J. Jackson, E. Shamir, and C. Shwartzman. Learning with queries corrupted by classification noise. In *Proceedings of the Fifth Israel Symposium on the Theory of Computing Systems*, pages 45–53, 1997.

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

M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

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

M. Kearns, R. Schapire, and L. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17 (2-3):115–141, 1994.

A. Klivans and R. Servedio. Boosting and hard-core set construction. *Machine Learning*, 51(3): 217–238, 2003.

A. Klivans and R. Servedio. Toward attribute efficient learning of decision lists and parities. In *Proceedings of COLT*, pages 234–248, 2004.

E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. In *Proceedings of STOC*, pages 455–464, 1991.

L. Levin. Randomness and non-determinism. *Journal of Symbolic Logic*, 58(3):1102–1103, 1993.

Y. Mansour. Learning boolean functions via the fourier transform. In V. P. Roychodhury, K. Y. Siu, and A. Orlitsky, editors, *Theoretical Advances in Neural Computation and Learning*, pages 391–424. Kluwer, 1994.

J. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inform. Theory*, 15:122–127, 1969.

R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42-44, 1978.

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

R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

R. Servedio. Computational sample complexity and attribute-efficient learning. *Journal of Computer and System Sciences*, 60(1):161–178, 2000.

R. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003.

M. Sudan. Essential coding theory (lecture notes). Available at `http://theory.lcs.mit.edu/~madhu/FT02/`, 2002.

R. Uehara, K. Tsuchida, and I. Wegener. Optimal attribute-efficient learning of disjunction, parity, and threshold functions. In *Proceedings of EuroCOLT '97*, pages 171–184, 1997.

L. Valiant. A neuroidal architecture for cognitive computation. *Journal of the ACM*, 47(5):854–882, 2000.

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

L. Valiant. *Circuits of the Mind*. Oxford University Press, 1994.

L. G. Valiant. Knowledge infusion. In *Proceedings of AAAI*, 2006.

J.H. van Lint. *Introduction to Coding Theory*. Springer, Berlin, 1998.

A. Vardy. Algorithmic complexity in coding theory and the minimum distance problem. In *Proceedings of STOC*, pages 92–109, 1997.