On Exact Learning from Random Walk

Iddo Bentov

On Exact Learning from Random Walk

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Iddo Bentov

Submitted to the Senate of
the Technion — Israel Institute of Technology
Tevet 5770 Haifa December 2009

The research thesis was done under the supervision of prof. Nader Bshouty in the Computer Science Department.

The generous financial support of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

The well known learning models in Computational Learning Theory are either adversarial, meaning that the examples are arbitrarily selected by the teacher, or i.i.d., meaning that the teacher generates the examples independently and identically according to a certain distribution. However, it is also quite natural to study learning models in which the teacher generates the examples according to a stochastic process.

A particularly simple and natural time-driven process is the random walk stochastic process. We consider exact learning models based on random walk, and thus having in effect a more restricted teacher compared to both the adversarial and the uniform exact learning models. We investigate the learnability of common concept classes via random walk, and give positive and negative separation results as to whether exact learning in the random walk models is easier than in less restricted models.

Abbreviations and Notations

log	—	Natural logarithm.
$\mathcal{O}$	—	Asymptotic upper bound.
$\tilde{\mathcal{O}}$	—	$\mathcal{O}$ with logarithmic factors ignored.
o	—	Upper bound that is not asymptotically tight.
ω	—	Lower bound that is not asymptotically tight.
$poly$	—	Polynomial.
$\mathbb{N}$	—	$\{1, 2, 3, \ldots\}$.
$\mathrm{Ham}(x, y)$	—	Hamming distance between x and y.
$x^{(k)}$	—	The example that the learner receives at the k^{th} trial.
X_n	—	$\{0, 1\}^n$.
U_n	—	Uniform distribution on X_n.
C_n	—	Class of boolean functions defined on X_n.
C	—	Class of boolean functions of the form $C = \cup_{n=1}^{\infty} C_n$.
$size_C(f)$	—	The representation size of f in the class C.
Exact	—	The Exact learning model.
Online	—	The Online learning model.
UROnline	—	The Uniform Random Online learning model.
RWOnline	—	The Random Walk Online learning model.
URWOnline	—	The Uniform Random Walk Online learning model.
PAC	—	The Probably Approximately Correct learning model.
uniform PAC	—	PAC under the uniform distribution.
k-junta	—	Boolean function that depends on only k of its variables.
RSE	—	Ring-Sum Expansion (k-term RSE is the parity of k monotone terms).
DNF	—	Disjunctive Normal Form.
read-k DNF	—	DNF in which every variable appears at most k times.
RO-DNF	—	read-once DNF.
ROM-DNF	—	read-once monotone DNF.

Chapter 1

Introduction and Overview

1.1 Introduction

While there is a variety of ways to model a learning process, there are widely shared properties that various learning models have in common. Generally speaking, we say that learning a boolean function $f : \{0,1\}^n \to \{0,1\}$ is the process of identifying the target function f by receiving from a teacher evaluations of f at some inputs $x_i \in X_n \triangleq \{0,1\}^n$, and deducing who f is from the examples $(x_i, f(x_i))$ that were received.

Our objective is for the learning process to be efficient. For the learner to be defined as efficient we require that certain polynomial bounds, specified in accordance with the particular learning model in question, must hold. Since it is exponentially hard to learn an arbitrary function out of the 2^{2^n} possible functions on X_n without any further assumptions, we usually refer to learning a concept class, meaning that the learner knows in advance that the target function f belongs a fixed class of functions.

There are well known learning models, such as PAC, which specify the additional relaxation that the objective of the learner is to obtain a hypothesis h whose statistical distance from the target function f is small, rather than obtaining f itself. However, in this work we concern ourselves with learning models in which the learner has the more difficult task of *exactly* and efficiently identifying the target function f.

The major open questions in Computational Learning Theory revolve around the learnability of natural concept classes, such as polynomial-size DNF formulas. In this work we investigate whether several such concept

classes can be learned in models that are more restricted than the general learning models, i.e. models in which the teacher is more restricted in the way that he/she is allowed to select the examples that are provided to the learner.

1.2 Learning Models

The two most well known exact learning models are the Exact Learning Model of Angluin [A87] and Online Learning Model of Littlestone [L87].

In the Exact model, the learner asks the teacher *equivalence queries* by providing the teacher with a hypothesis h, and the teacher answers "Yes" if the hypothesis is equivalent to the target function f, otherwise the teacher provides the learner with a counterexample x for which $h(x) \neq f(x)$. The goal of the learner is to minimize the number of equivalence queries, under the constraint of generating each equivalence query in polynomial time.

In the Online model, at each *trial* the teacher sends a point x to the learner, and the learner has to predict $f(x)$. The learner returns to the teacher the prediction y. If $f(x) \neq y$ then the teacher returns "mistake" to the learner. The goal of the learner is to minimize the number of prediction mistakes, under the constraint of computing the answer at each trial in polynomial time.

Another way to model a learning process is by allowing the learner to actively perform *membership queries*, meaning that the learner asks for the value of the target function f at certain inputs x_i, and the teacher provides him with the answers $f(x_i)$. The learner seeks to minimize the number of membership queries that would be asked.

Let us also mention the most widely known non-exact learning model, the Probably Approximately Correct (PAC) model. In the PAC model, the teacher selects samples $x_i \in X_n$ according to a fixed distribution $\mathcal{D}$ that is unknown to the learner, and provides the learner with examples $(x_i, f(x_i))$ upon the learner's request. The learner has a confidence parameter δ and an accuracy parameter ε, and for any fixed distribution $\mathcal{D}$ the learner must achieve at least $1 - \delta$ success probability in obtaining a hypothesis h that satisfies $\Pr_{x \in \mathcal{D}}(h(x) \neq f(x)) \leq \varepsilon$. The running time of the learner must not exceed $poly(\frac{1}{\varepsilon}, \frac{1}{\delta}, n, size_C(f))$, which also bounds the number of examples that the learner receives.

It is well known and easy to see that the Exact and Online models are equivalent. An Online algorithm can be regarded as having hypotheses $h_0, h_1, h_2, \ldots$ that it uses to make predictions, i.e. at start it uses h_0 to make predictions, then after the first prediction mistake it uses h_1, and so on. Each hypothesis h_i can be regarded as a polynomial-size circuit by Ladner's theorem ($\mathbf{P} \subseteq \mathbf{P/poly}$). To see that Online $\Longrightarrow$ Exact, each h_i can be sent to the teacher as an equivalence query, so that the counterexample provided by the teacher can be used to compute h_{i+1}. To see in the other direction that Exact $\Longrightarrow$ Online, the hypothesis generated for each equivalence query can be used to make predictions, and when a prediction mistake occurs it can be provided back as a counterexample. Under these simulations the number of equivalence queries is one more than the number of prediction mistakes, and therefore it follows that the Exact and Online models are equivalent.

It is also well known that learnability in the Exact model implies learnability in the PAC model [A87], but not vice versa under the cryptographically weak assumption that one-way functions exist [A94].

By restricting the power of the teacher to select examples, many variants based on these general learning models can be defined, thus allowing to consider models in which it is easier for the learner to achieve his objective. Of particular interest are the uniform Online model (UROnline) [B97], the random walk Online model (RWOnline) [BFH95], and the uniform random walk Online model (URWOnline) [BFH95]. The UROnline is the Online model where examples are generated independently and uniformly randomly. In the RWOnline model successive examples differ by exactly one bit, and in the URWOnline model the examples are generated by a uniform random walk stochastic process on X_n.

It is of both theoretical and practical significance to examine learning models in which the learner receives correlated examples that are generated by a time-driven process. From a theoretical standpoint, these are natural passive learning models that can be strictly easier than standard passive models where the examples are generated independently [ELSW07], and yet strictly harder than the less realistic active learning models that use membership queries [BMOS03], under common cryptographic assumptions. From a practical standpoint, in many situations the learner doesn't obtain independent examples, as assumed in models such as PAC and UROnline. In particular, successive examples that are generated by a physical process

tend to differ only slightly, e.g. trajectory of robots, and therefore learning models that are based on random walk or similar stochastic processes are more appropriate in such cases.

1.3 Previous Results

Following the results in [D88, BFH95, BMOS03], it is simple to show (see Appendix A for a precise statement) that learnability in the UROnline model with a mistake bound q implies learnability in the URWOnline model with a mistake bound $\tilde{\mathcal{O}}(qn)$. Obviously, learnability in the Online model implies learnability in all the other models that are based on it with the same mistake bound, and learnability in the RWOnline model implies learnability in the URWOnline model with the same mistake bound. Therefore we have the following:

$$\begin{array}{ccc} \text{Online} & \Rightarrow & \text{RWOnline} \\ \Downarrow & & \Downarrow \\ \text{UROnline} & \Rightarrow & \text{URWOnline} \end{array}$$

In [BFH95] Bartlett et. al. developed efficient algorithms for exact learning boolean threshold functions, 2-term RSE, and 2-term DNF in the RWOnline model. Those classes are already known to be learnable in the Online model [L87, FS92], but the algorithms in [BFH95] achieve a better mistake bound (for threshold functions).

The fastest known algorithm for learning polynomial-size DNF formulas in the PAC model under the *uniform* distribution runs in $n^{\mathcal{O}(\log n)}$ time [V90]. In [HM91] it is shown that the read-once DNF class can be learned in the uniform PAC model in polynomial time, but that does not imply UROnline learnability since the learning is not exact (see also Appendix B). The fastest known algorithm for exact learning of general DNF formulas in adversarial settings, i.e. in the Exact or Online models, runs in $2^{\tilde{\mathcal{O}}(n^{1/3})}$ time [KS01].

In [BMOS03] Bshouty et. al. show that DNF is learnable in the *uniform random walk* PAC model, but here again that does not imply that DNF is learnable in the URWOnline model, since the learning is not exact.

1.4 Our Results

We will present a negative result, showing that for all classes that possess a simple natural property, if the class is learnable in the RWOnline model, then it is learnable in the Online model with the same (asymptotic) mistake bound. Those classes include: read-once DNF, k-term DNF, k-term RSE, decision list, decision tree, DFA and halfspaces.

To study the relationship between the UROnline model and the URWOnline model, we then focus our efforts on studying the learnability of some classes in the URWOnline model that are not known to be polynomially learnable in the UROnline model. In particular, it is unknown whether the class of functions of $\mathcal{O}(\log n)$ relevant variables can be learned in the UROnline model with a polynomial mistake bound (this is an open problem even for $\omega(1)$ relevant variables [MOS04]), but it is known that this class can be learned with a polynomial number of membership queries. We will present a positive result, showing that the information gathered from consecutive examples that are generated by a random walk process can be used in a similar fashion to the information gathered from membership queries, and thus we will prove that this class is learnable in the URWOnline model.

We then establish another result that shows that URWOnline learnability can indeed be easier, by proving that the class of read-once DNF formulas can be learned in the URWOnline model. It is a major open question whether this class can be learned in the Online model, as that implies that the general DNF class can also be learned in the Online and PAC models [KLPV87, PW90]. Therefore, this result separates the Online and the RWOnline models from the URWOnline model, unless DNF is Online learnable. With the aforementioned hardness assumptions regarding the learnability of the class of functions of $\mathcal{O}(\log n)$ relevant variables and the class of DNF formulas, we now have:

$$\begin{array}{ccc} \text{Online} & \equiv & \text{RWOnline} \\ \Downarrow & & \Downarrow\not\Uparrow \\ \text{UROnline} & \begin{array}{c}\not\Leftarrow \\ \Rightarrow\end{array} & \text{URWOnline} \end{array}$$

1.5 Outline of the Thesis

Chapter 2 gives the precise definitions of the learning models and concept classes that we explore throughout this work. Chapter 3 presents a simple negative result showing that the RWOnline and Online models are practically equivalent. In Chapter 4 we present a relatively straightforward positive result, by showing how to learn functions that depend on $\mathcal{O}(\log n)$ of their n variables in the URWOnline model. Under the conjecture that $\mathcal{O}(\log n)$-juntas are not UROnline learnable, this result separates the URWOnline model from the UROnline and Online models. Chapter 5 presents a positive result which is rather more involved, showing that the class of read-once DNF formulas is learnable in the URWOnline model. Under the widely believed conjecture that DNF is not Online learnable, this result separates the URWOnline and Online models. In Chapter 6 we present a negative result which shows that certain classes are unlikely to be learnable in the URWOnline model, in the sense that if read-3 DNF can be learned in the URWOnline model (and some reasonable assumptions hold), then any DNF can be learned in the UROnline model.

Chapter 2

Definitions and Models

In this chapter we present the notation and give the precise definitions of the generic Online learning model and the random walk learning models that are based on it.

2.1 Boolean Functions

Here we give the basic notation and definitions that we require.

Boolean variables. A *boolean variable* is a variable having one of only two values. We denote these values by 0 and 1, and refer to them as false and true correspondingly.

Boolean functions. Let us use the notation $X_n \triangleq \{0,1\}^n$. A *boolean function* $f : X_n \to \{0,1\}$ is a function that depends on n boolean variables and having the boolean domain $\{0,1\}$.

Literals. A *literal* is a boolean variable or its negation. We denote the negation of the boolean variable x by $\bar{x}$. We use the notation $lit(x) \in \{x, \bar{x}\}$.

Terms. A *term* is a conjunction of literals. We use $\wedge$ to denote conjunction.

Monotone terms. A *monotone term* is a conjunction of positive literals, meaning that none of the variables in the term are negated.

DNF formulas. A DNF formula is a disjunction of terms. We use $\vee$ to denote disjunction.

k-term DNF formulas. A k-term DNF formula is a disjunction of at most k terms.

read-once DNF formulas. A *read-once* DNF formula (RO-DNF) is a DNF formula for which no variable appears in more than one term.

read-once monotone DNF formulas. A *read-once monotone* DNF formula (ROM-DNF) is a read-once DNF formula for which all the terms are monotone.

k-junta. A *k-junta* is a boolean function $f : X_n \to \{0,1\}$ that depends on only k of its n variables.

Halfspaces. A *Halfspace* over $\{0,1,2,\ldots,m\}^n$ is a function of the form

$$f(x_1,\ldots,x_n) = \begin{cases} 1 & a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq b \\ 0 & \text{otherwise} \end{cases}$$

where $a_1, a_2, \ldots, a_n, b$ are real numbers. If $m = 1$ then the variables are boolean. We use the notation $f(x_1,\ldots,x_n) = [\sum_{i=1}^{n} a_i x_i \geq b]$.

2.2 Concept Classes

Let n be a positive integer and $X_n = \{0,1\}^n$. We consider the learning of classes of the form $C = \cup_{n=1}^{\infty} C_n$, where each C_n is a class of boolean functions defined on X_n. Each function $f \in C$ has some string representation $R(f)$ over some fixed alphabet Σ. The length $|R(f)|$ is denoted by $size_C(f)$.

2.3 The Online Learning Model

In the *Online learning model* (Online) [L87], the learning task is to *exactly* identify an unknown *target* function f that is chosen by a *teacher* from a fixed class C that is known to the learner. At each *trial* $t = 1, 2, 3, \ldots$, the teacher sends a point $x^{(t)} \in X_n$ to the *learner* and the learner has to predict $f(x^{(t)})$. The learner returns to the teacher the prediction y. If $f(x^{(t)}) \neq y$ then the teacher responds by sending a "mistake" message back to the learner. The goal of the learner is to minimize the number of prediction mistakes.

In the Online learning model we say that algorithm **A** of the learner *Online learns* the class C with a mistake bound q if for any $f \in C$ algorithm **A** makes no more than q mistakes. The *hypothesis* of the learner is denoted by h, and the learning is called *exact* because we require that $h \equiv f$ after q mistakes. We say that C is *Online learnable* if there ex-

ists a learner that Online learns C with a $poly(n, size_C(f))$ mistake bound, and the running time of the learner for each prediction is $poly(n, size_C(f))$. The learner may depend on a *confidence* parameter δ, by having a mistake bound $q = poly(n, size_C(f), \frac{1}{\delta})$, and probability that $h \not\equiv f$ after q mistakes smaller than δ. However, it is also the case that repetitive iterations of the learning algorithm result in an exponential decay of the failure probability, thus allowing a tighter bound of the form $q = \log \frac{1}{\delta} \cdot poly(n, size_C(f))$ to be obtained.

2.4 Uniform and Random Walk Learning Models

We now turn to define the particular learning models that we consider in this work. The following models are identical to the Online model, with various constraints on successive examples that are presented by the teacher at each trial:

Uniform Random Online (UROnline) In this model successive examples are independent and randomly uniformly chosen from X_n.

Random Walk Online (RWOnline) In this model successive examples differ by at most one bit.

Uniform Random Walk Online (URWOnline) This model is identical to the RWOnline learning model, with the added restriction that

$$\Pr(x^{(t+1)} = y \mid x^{(t)}) = \begin{cases} \frac{1}{n+1} & \text{if } \mathrm{Ham}(y, x^{(t)}) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

where $x^{(t)}$ and $x^{(t+1)}$ are successive examples for a function that depends on n bits, and the Hamming distance $\mathrm{Ham}(y, x^{(t)})$ is the number of bits of y and $x^{(t)}$ that differ. Starting at $x^{(1)} = (0, 0, \ldots, 0)$ or at $x^{(1)}$ that is distributed randomly uniformly, this conditional probability defines the *uniform random walk* stochastic process. For our purposes, the teacher is allowed to select $x^{(1)}$ arbitrarily as well.

Let us also define the *lazy random walk* stochastic process, which is identical to the uniform random walk, except that it is based on the following

probability distribution

$$\Pr(x^{(t+1)} = y \mid x^{(t)}) = \begin{cases} \frac{1}{2n} & \text{if } \mathrm{Ham}(y, x^{(t)}) = 1 \\ \frac{1}{2} & \text{if } y = x^{(t)} \\ 0 & \text{otherwise} \end{cases}$$

Finally, let us define the *simple random walk* stochastic process, which is also identical to the uniform random walk, but based on the following probability distribution

$$\Pr(x^{(t+1)} = y \mid x^{(t)}) = \begin{cases} \frac{1}{n} & \text{if } \mathrm{Ham}(y, x^{(t)}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

As a side note, we mention here that the simple random walk never converges to the uniform distribution, because the parity of the bits at odd steps is always the same as in the first step.

Because Online learning algorithms can always make the correct prediction when $x^{(t)} = x^{(t-1)}$, learning via lazy random walk is equivalent to learning via uniform random walk and via simple random walk.

Chapter 3

RWOnline versus Online

In [BFH95] Bartlett et. al. developed efficient algorithms for exact learning boolean threshold functions, 2-term Ring-Sum Expansion (parity of 2 monotone terms) and 2-term DNF in the RWOnline model. Those classes are already known to be learnable in the Online model [L87, FS92] (and therefore in the RWOnline model), but the algorithm in [BFH95] for boolean threshold functions achieves a better mistake bound. They show that this class can be learned by making no more than $n+1$ mistakes in the RWOnline model, improving on the $\mathcal{O}(n \log n)$ bound for the Online model proven by Littlestone in [L87].

Can we achieve a better mistake bound for other concept classes? We present a negative result, showing that for all classes that possess a simple natural property, the RWOnline model and the Online models have the same asymptotic mistake bound. Those classes include: read-once DNF, k-term DNF, k-term RSE, decision list, decision tree, DFA and halfspaces.

We first give the following

Definition 1. A class of boolean functions C has the *one variable override* property if for every $f(x_1, \ldots, x_n) \in C$ there exist constants $c_0, c_1 \in \{0,1\}$ and $g(x_1, \ldots, x_{n+1}) \in C$ such that

$$g \equiv \begin{cases} f & x_{n+1} = c_0 \\ c_1 & \text{otherwise} \end{cases} .$$

Common classes do possess the one variable override property. The following lemma illustrates several examples.

Lemma 1. The concept classes that possess the one variable override property include: read-once DNF, k-term DNF, k-term RSE, decision list, decision tree, DFA and halfspaces.

Proof.

- Consider the class of RO-DNF. For a RO-DNF formula $f(x_1, \ldots, x_n)$, define $g(x_1, \ldots, x_{n+1}) = x_{n+1} \vee f(x_1, \ldots, x_n)$. Then g is a RO-DNF, $g(x, 0) = f(x)$ and $g(x, 1) = 1$. The construction is also good for decision list, decision tree and DFA.

- For k-term DNF and k-term RSE we can take $g = x_{n+1} \wedge f$.

- For halfspace, consider the function $f(x_1, \ldots, x_n) = [\sum_{i=1}^{n} a_i x_i \geq b]$. Then $g(x_1, \ldots, x_{n+1}) = x_{n+1} \vee f(x_1, \ldots, x_n)$ can be expressed as $g(x_1, \ldots, x_{n+1}) = [(b + \sum_{i=1}^{n} |a_i|)x_{n+1} + \sum_{i=1}^{n} a_i x_i \geq b]$. □

Notice that the class of boolean threshold functions $f(x_1, \ldots, x_n) = [\sum_{i=1}^{n} a_i x_i \geq b]$ where $a_i \in \{0, 1\}$ does not have the one variable override property, because the value of any variable x_i can affect the sum by no more than ± 1.

In order to show equivalence between the RWOnline and Online models, we notice that a malicious teacher could set a certain variable to override the function's value, then choose arbitrary values for the other variables via random walk, and then reset this certain variable and ask the learner to make a prediction. Using this idea, we now prove

Theorem 1. Let C be a class that has the one variable override property. If C is learnable in the RWOnline model with a mistake bound $T(n)$ then C is learnable in the Online model with a mistake bound $4T(n + 1)$.

Proof. Suppose C is learnable in the RWOnline model by some algorithm $\mathbf{A}$, which has a mistake bound of $T(n)$. Let $f(x_1, \ldots, x_n) \in C$ and construct

$$g(x_1, \ldots, x_{n+1}) \equiv \begin{cases} f & x_{n+1} = c_0 \\ c_1 & \text{otherwise} \end{cases}$$

using the constants c_0, c_1 that exist due to the one variable override property of C. An algorithm $\mathbf{B}$ for the Online model will learn f by using algorithm $\mathbf{A}$ simulated on g according to these steps:

1. At the first trial
   (a) Receive $x^{(1)}$ from the teacher.
   (b) Send $(x^{(1)}, c_0)$ to $\mathbf{A}$ and receive the answer y.
   (c) Send the answer y to the teacher, and inform $\mathbf{A}$ in case of a mistake.
2. At trial t
   (a) receive $x^{(t)}$ from the teacher.
   (b) $\tilde{x}^{(t-1)} \leftarrow (x_1^{(t-1)}, x_2^{(t-1)}, \ldots, x_n^{(t-1)}, \overline{c_0})$, $\tilde{x}^{(t)} \leftarrow (x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}, \overline{c_0})$
   (c) Walk from $\tilde{x}^{(t-1)}$ to $\tilde{x}^{(t)}$, asking $\mathbf{A}$ for predictions, and informing $\mathbf{A}$ of mistakes in case it fails to predict c_1 after each bit flip.
   (d) Send $(x^{(t)}, c_0)$ to $\mathbf{A}$.
   (e) Let y be the answer of $\mathbf{A}$ on $(x^{(t)}, c_0)$.
   (f) Send the answer y to the teacher, and inform $\mathbf{A}$ in case of a mistake.

Obviously, successive examples given to $\mathbf{A}$ differ by exactly one bit, and the teacher that we simulated for $\mathbf{A}$ provides it with the correct "mistake" messages, since $g(x^{(t)}, c_0) = f(x^{(t)})$. Therefore, algorithm $\mathbf{A}$ will learn g exactly after at most $T(n+1)$ mistakes, and thus $\mathbf{B}$ also makes no more than $T(n+1)$ mistakes.

Observe that for common classes such as the ones mentioned in Lemma 1, the construction of g is straightforward. However, in case the two constants c_0, c_1 cannot easily be determined, it is possible to repeat this procedure after more than $T(n+1)$ mistakes were received, by choosing different constants. Thus the mistake bound in the worst case is $2^2 T(n+1) = 4T(n+1)$. □

Chapter 4

URWOnline versus UROnline

4.1 Learning $\mathcal{O}(\log n)$ Relevant Variables

In this section we present a probabilistic algorithm for the URWOnline model that learns the class of boolean functions of k relevant variables, i.e. boolean functions that depend on at most k of their n variables. We show that the algorithm makes no more than $\tilde{\mathcal{O}}(2^k)\log\frac{1}{\delta}$ mistakes, and thus in particular for $k=\mathcal{O}(\log n)$ the number of mistakes is polynomially bounded.

The learnability of functions that depend on $k \ll n$ variables, which are commonly referred to as k-juntas, is a challenging real-world task in the field of machine learning, which often deals with the issue of how to efficiently learn in the presence of irrelevant information. For example, suppose that each query represents a long DNA sequence, and the boolean target function is some biological property that depends only on a small (e.g. logarithmic) unknown active part of each DNA sequence.

There is another important motivation for investigating $\mathcal{O}(\log n)$-juntas, related to a major open question in computational learning theory: can polynomial-size DNF be learned efficiently? Since each $(c\cdot\log n)$-junta is in particular a n^c-term DNF, polynomial-size DNF learnability implies $\mathcal{O}(\log n)$-juntas learnability. Therefore, better understanding of the difficulties with learning $\mathcal{O}(\log n)$-juntas might shed light on DNF learnability as well. Conversely, any decision tree with k leaves is a k-junta, which means that learning k-juntas implies learning k-size decision trees. It also implies non-exact learning of k-term DNF in the uniform PAC model, under a slightly stronger assumption [MOS04].

Currently, polynomial time learning of k-term DNF and k-size decision trees, in uniform PAC, are open questions even for $k = \omega(1)$. Thus, it is unknown whether the class of k-juntas can be learned in polynomial time in the UROnline model, even for $k = \omega(1)$ (cf. Appendix B).

However, it is known that this class in learnable from only membership queries. Specifically, it is possible to construct in $2^k k^{\mathcal{O}(\log k)} \log n$ time a (n, k)-universal set $T \subseteq X_n$ of truth assignments, meaning that for any index set $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$, the projection of T onto S contains all of the 2^k combinations [NSS95]. Then T can be used to discover the relevant variables one at a time, by picking two assignments which differ on f but have identical values for all the relevant variable that were already discovered, and walking from one assignment to the other by toggling one of the undiscovered variables each time and asking a membership query on each intermediate assignment, until a new relevant variable is discovered when a toggle triggers a flip in the value of f. This algorithm achieves learnability in $2^k k^{\mathcal{O}(\log k)} \log n$ time, which implies that $\mathcal{O}(\log n)$-juntas can be learned in deterministic polynomial time from membership queries.

We use similar ideas in the URWOnline model, i.e. we exploit the random walk properties to reach an assignment for which the random walk triggers the discovery of a new relevant variable each time. Our algorithm is fairly simple, though its correctness proof involves certain effort and demonstrates some of the main tools that are used in the analysis the random walk stochastic process.

The URWOnline algorithm RVL(δ) for learning functions that depend on at most k variables, shown in figure 1, receives an example $x^{(t)}$ at each trial $t = 1, 2, 3, \ldots$ from the teacher, and makes a prediction for $f(x^{(t)})$.

RVL(δ):

1. $S \leftarrow \emptyset$
2. At the first trial, make an arbitrary prediction for $f(x^{(1)})$
3. Phase 1 - find relevant variables as follows:
   (a) At trial t, predict $h(x^{(t)}) = f(x^{(t-1)})$
   (b) In case of a prediction mistake, find the unique i such that $x^{(t-1)}$ and $x^{(t)}$ differ on the i^{th} bit, and perform $S \leftarrow S \cup \{x_i\}$
   (c) If S hasn't been modified after $\alpha(k, \delta)$ consecutive prediction mistakes, then assume that S contains all the relevant variables and goto (4)
   (d) If $|S| = k$ then goto (4), else goto (3.a)
4. Phase 2 - learn the target function:
   (a) Prepare a truth table with $2^{|S|}$ entries for all the possible assignments of the relevant variables
   (b) At trial t, predict on $x^{(t)}$ as follows:
      i. If $f(x^{(t)})$ is yet unknown because the entry in the table for the relevant variables of $x^{(t)}$ hasn't been determined yet, then make an arbitrary prediction and then update that table entry with the correct value of $f(x^{(t)})$
      ii. If the entry for the relevant variables of $f(x^{(t)})$ has already been set in the table, then predict $f(x^{(t)})$ according to the table value

Figure 4.1: The RVL(δ) Algorithm - Relevant Variables Learner

4.2 Complexity of RVL(δ)

Let us first calculate the mistake bound of the algorithm. We define[1]

$$\alpha(k,\delta) \triangleq 2^{k+1}k^2(1+\log k)\log\frac{k}{\delta}.$$

The maximal number of prediction mistakes in phase 1 before each time a new relevant variable is discovered is $\alpha(k,\delta)$, and therefore the total number of prediction mistakes possible in phase 1 is at most $k\alpha(k,\delta)$. We will prove in the next subsection that with probability of at least $1-\delta$ the first phase finds all the relevant variables.

In case phase 1 succeeds, the maximal number of prediction mistakes in phase 2 is 2^k. Thus the overall number of prediction mistakes that RVL(δ) would make is bounded by

$$2^k + k\alpha(k,\delta) \leq 2^k poly(k)\log\frac{1}{\delta}.$$

This implies

Corollary 1. For $k = \mathcal{O}(\log n)$, the number of mistakes that RVL(δ) makes is bounded by $poly(n)\log\frac{1}{\delta}$ with probability of at least $1-\delta$.

4.3 Correctness of RVL(δ)

We will show that the probability that the hypothesis generated by RVL(δ) is not equivalent to the target function is less than δ. This will be done using the fact that a uniform random walk stochastic process is similar to the uniform distribution. An accurate probability-theoretic formulation is stated in Lemma 2, but its proof requires either relatively powerful tools from the mathematical field of representation theory, or relatively advanced pure-probability (coupling) arguments. Fortunately, for clarity we can provide the easier albeit weaker Lemma 3, and the penalty factor for using Lemma 3 instead of Lemma 2 will be constant.

For Lemma 2, we first require the following definition

[1] log denotes the natural logarithm throughout this work.

Definition 2. Let U_n be the uniform distribution on X_n. A stochastic process $P = (Z_1, Z_2, Z_3, \ldots)$ is said to be *ε-close to uniform* if

$$P_{m|x}(b) = \Pr(Z_{m+1} = b \mid Z_i = x^{(i)}, i = 1, 2, \ldots, m)$$

is defined for all $m \in \mathbb{N}$, for all $b \in X_n$, and for all $x = (x^{(1)}, x^{(2)}, x^{(3)}, \ldots) \in X_n^{\mathbb{N}}$, and the following total variation distance bound

$$\max_{S \subseteq X_n} |P_{m|x}(S) - U_n(S)| = \frac{1}{2} \sum_{b \in X_n} |P_{m|x}(b) - U_n(b)| \leq \varepsilon$$

holds for all $m \in \mathbb{N}$ and for all $x \in X_n^{\mathbb{N}}$.

We now quote the following lemma, that is proven in [DS87, D88]

Lemma 2. For the uniform random walk stochastic process P and any $0 < \varepsilon < 1$, let Q_m be the stochastic process that corresponds to sampling P after at least m steps. Then Q_m is ε-close to uniform for

$$m = \frac{n+1}{4} \log \frac{n}{\log(2\varepsilon^2 + 1)}.$$

We note that a lower bound can also be shown, i.e. there is a cutoff phenomenon at $m \approx \frac{1}{4} n \log n$, after which Q_m very rapidly converges to U_n.

Let us now prove the following lemma, which is a different formulation of the well known "coupon collector's problem".

Lemma 3. The expected mixing time of the uniform random walk stochastic process P is smaller than $n(1 + \log n)$. More precisely, starting at any state of P, the expected number of consecutive steps after which sampling from P would be identical to sampling from U_n is less than $n(1 + \log n)$.

Proof. Consider the stochastic process on X_n where in each step an index i, $1 \leq i \leq n$, is selected uniformly with probability $\frac{1}{n}$, and then with probability $\frac{1}{2}$ the i^{th} bit is flipped. Observe that this stochastic process is identical to the lazy random walk stochastic process. For $1 \leq j \leq n$, let Y_j

be the random variable that counts the number of steps since $j-1$ unique indices were already selected until a new unique index is selected. Thus, each Y_j is a geometric random variable with success probability $\frac{n-j+1}{n}$, and all the bits are uniformly distributed after n unique indices were selected. Consequently, the expected mixing time is

$$\begin{aligned}\mathrm{Ex}\left[\sum_{j=1}^{n} Y_j\right] &= \sum_{j=1}^{n}\mathrm{Ex}[Y_j] = \sum_{j=1}^{n}\frac{n}{n-j+1} = n\cdot\sum_{j=1}^{n}\frac{1}{j} \\ &= n\cdot H_n \approx n(\gamma+\log n) < n(1+\log n),\end{aligned}$$

where H_n is the partial harmonic sum, and $\gamma \approx 0.57$ is Euler's constant. $\square$

Suppose the target function f depends on k variables. We can consider the 2^n possible assignments as 2^k equivalence classes of assignments, where each equivalence class consists of 2^{n-k} assignments under which f has the same value. We note that flipping an irrelevant variable x_i will not change the value of f, and therefore RVL(δ) cannot make prediction mistakes when such flips occur. Hence, we can ignore the irrelevant variables and analyze a uniform random walk stochastic process on the hypercube $\{0,1\}^k$ of the relevant variables. For any trial t and for any index $1 \le i \le k$, let us define the following events

$$\begin{aligned}\mathcal{A}^t &\triangleq \{\text{all the bits of } x^{(t)} \text{ became uniformly distributed}\},\\ \mathcal{B}_i^t &\triangleq \{f(x^{(t)}) \neq f(y) \text{ where } y \text{ is } x^{(t)} \text{ with the } i^{th} \text{ bit flipped}\},\\ \mathcal{C}_i^{t,t_0} &\triangleq \{x^{(t+t_0)} \text{ differs from } x^{(t)} \text{ in the } i^{th} \text{ bit, } x^{(t)} = x^{(t+1)} = \ldots = x^{(t+t_0-1)}\},\\ \mathcal{C}_i^t &\triangleq \bigcup_{t_0\ge 1} \mathcal{C}_i^{t,t_0}.\end{aligned}$$

Our analysis will show that the event $\mathcal{B}_i^t \cap \mathcal{C}_i^t$ occurs with significantly high probability in case x_i is a relevant variable, thus triggering RVL(δ) to discover that x_i is relevant.

Let Y_j be as in Lemma 3, and let $m = 2k(1+\log k)$. Suppose that starting at some trial t we ignore all the prediction mistakes that occur during $m-1$ consecutive trials, and consider $x^{(t+m-1)}$ as a newly sampled

example. By Markov's inequality,

$$\Pr\left(\overline{\mathcal{A}^{t+m-1}}\right) = \Pr\left(\sum_{j=1}^{k} Y_j \geq m\right) \leq \frac{\mathrm{Ex}[\sum_{j=1}^{k} Y_j]}{m} < \frac{\mathrm{Ex}[\sum_{j=1}^{k} Y_j]}{2\mathrm{Ex}[\sum_{j=1}^{k} Y_j]} = \frac{1}{2}.$$

Thus $\Pr(\mathcal{A}^{t+m-1}) > \frac{1}{2}$. Now, let us assume that x_i is relevant, which implies that there exist at least two truth assignments for which flipping x_i changes the value of f. In other words, the probability that a uniformly randomly chosen assignment belongs to an equivalence class in which flipping the i^{th} bit changes the value of f is at least $\frac{2}{2^k}$. This gives

$$\begin{aligned}
\Pr(\mathcal{B}_i^{t+m-1}) &\geq \Pr(\mathcal{B}_i^{t+m-1} \cap \mathcal{A}^{t+m-1}) \\
&= \Pr(\mathcal{B}_i^{t+m-1} | \mathcal{A}^{t+m-1}) \cdot \Pr(\mathcal{A}^{t+m-1}) \\
&\geq \frac{2}{2^k} \cdot \Pr(\mathcal{A}^{t+m-1}) > \frac{2}{2^k} \cdot \frac{1}{2} = \frac{1}{2^k}.
\end{aligned}$$

We now note that for any t the events $\mathcal{B}_i^t$ and $\mathcal{C}_i^t$ are independent. Also, for any $1 \leq i, j \leq k$ and any $t_0 \geq 1$ it holds that $\Pr(\mathcal{C}_i^{t,t_0}) = \Pr(\mathcal{C}_j^{t,t_0})$, which implies $\Pr(\mathcal{C}_i^t) = \Pr(\mathcal{C}_j^t) = \frac{1}{k}$. Therefore,

$$\begin{aligned}
\Pr(\mathcal{B}_i^{t+m-1} \cap \mathcal{C}_i^{t+m-1}) &= \Pr(\mathcal{B}_i^{t+m-1}) \cdot \Pr(\mathcal{C}_i^{t+m-1}) \\
&> \frac{1}{2^k} \cdot \Pr(\mathcal{C}_i^{t+m-1}) = \frac{1}{2^k} \cdot \frac{1}{k}.
\end{aligned}$$

Let us only consider the prediction mistakes that occur after at least m trials since the previously considered prediction mistake. In order to get the probability that x_i would not be discovered after d such prediction mistakes to be lower than $\frac{\delta}{k}$, we require

$$\left(1 - \frac{1}{k2^k}\right)^d \leq \frac{\delta}{k},$$

and using the fact that $1 - x \leq e^{-x}$, we get that

$$d = k2^k \log \frac{k}{\delta}$$

will suffice.

Therefore, if we allow $mk2^k \log \frac{k}{\delta}$ prediction mistakes while trying to

discover x_i, the probability of a failure is at most $\frac{\delta}{k}$. Now,

$$\begin{aligned}\Pr(\{\text{RVL}(\delta) \text{ fails}\}) &= \Pr(\{\text{finding } x_{i_1} \text{ fails}\} \vee \ldots \vee \{\text{finding } x_{i_k} \text{ fails}\}) \\ &\leq \sum_{q=1}^{k} \Pr(\{\text{finding } x_{i_q} \text{ fails}\}) \\ &\leq \sum_{q=1}^{k} \Pr(\{\text{finding } x_{i_k} \text{ fails}\}) \;\leq\; k\frac{\delta}{k} \;=\; \delta.\end{aligned}$$

Notice that

$$mk2^k \log \frac{k}{\delta} = 2k(1 + \log k)k2^k \log \frac{k}{\delta} = \alpha(k, \delta).$$

This is the maximal amount of prediction mistakes that the algorithm is set to allow while trying to discover a relevant variable, and thus the proof of the correctness of RVL(δ) is complete. □

4.4 Extensions

4.4.1 Unknown k

In case the learner knows that there exists some k for which the concept class contains functions that depend on k variables, but does not know what the value of k is, learning the class via RVL(δ) is still possible.

The learner can run RVL(δ) for $k' = 1, 2, 3, \ldots$, i.e. if less than k' relevant variables were discovered in phase 1, or more than $2^{k'}$ mistakes occurred in phase 2, then the learner restarts the learning process with $k' + 1$ instead of k'. However, rather than trying to discover k' relevant variables in phase 1 with each invocation of RVL(δ), the learner can store the set V of relevant variables that were already discovered in the previous invocations, and try to find only $k' - |V|$ relevant variables in phase 1 each time.

Assuming that f depends on exactly k variables, at some point during this process RVL(δ) will be invoked with $k' = k$, and will successfully learn f with probability at least $1 - \delta$ in this invocation. The total amount of

prediction mistake up to and including this invocation is bounded by

$$\sum_{i=1}^{k} 2^i poly(i) \log \frac{1}{\delta} \leq 2 \cdot 2^k poly(k) \log \frac{1}{\delta} = 2^k poly(k) \log \frac{1}{\delta},$$

and therefore with probability at least $1 - \delta$ the number of prediction mistakes that the learner makes is $\tilde{\mathcal{O}}(2^k) \log \frac{1}{\delta}$, as in the case where k is known in advance. □

4.4.2 Partially Observable Random Walk

Like the result in Chapter 5, the result here demonstrates that learning is possible in a weaker model, known as the "partially observable random walk" model (cf. [BMOS03]). In this model the examples are generated as in the URWOnline model, but the learner is only allowed to observe the location of the random walk by receiving two successive examples after at least $c_0 \cdot n$ steps, for some constant c_0.

4.4.3 Minimal Sensitivity

Under a further assumption regarding the sensitivity of f, phase 1 becomes more efficient. The influence of a variable x_i on f is defined as the probability that $f(x) \neq f(x \oplus e_i)$ where x is chosen uniformly randomly from X_n, and e_i is the standard basis vector that consists of 1 at the i^{th} index and 0 elsewhere. The minimal sensitivity of f is defined as the smallest influence among all of its (relevant) variables.

If the minimal sensitivity is $\frac{1}{S}$, then $\frac{2}{2^k} \leq \Pr(\mathcal{B}_i^{t+m-1} | \mathcal{A}^{t+m-1})$ can be replaced with $\frac{1}{S} \leq \Pr(\mathcal{B}_i^{t+m-1} | \mathcal{A}^{t+m-1})$ in the analysis of RVL(δ). If further knowledge is available with regard to the concept class that the target function belongs to, stage 2 of RVL(δ) can be replaced by an Online learning algorithm for that class, which would be executed on the relevant variables.

For example, the minimal sensitivity of the parity function is 1. Thus, if it is known in advance that the target function is a parity function, then stage 1 becomes exponentially faster relative to k, and stage 2 becomes redundant since the target function is already known once all its relevant variables were discovered.

Chapter 5

URWOnline versus Online

5.1 Learning Read-Once Monotone DNF

We now consider the *read-once monotone* DNF (ROM-DNF) class of boolean functions, i.e. DNF formulas in which each variable appears at most once, and none of the variables are negated.

If it is possible to learn this class in the Online model, then it can be shown using the Composition Lemma [PW90, KLPV87] that the general class of DNF functions is also learnable in the Online model. Since we have shown that proving such a result is not easier in the RWOnline model than in the Online model, we will now prove that we can learn the ROM-DNF class in the URWOnline model. This will give further evidence that learnability in the URWOnline can indeed be easier than in the RWOnline and Online models.

The Online learning algorithm **ROM-DNF-L**(δ), shown in figure 2, receives an example $x^{(t)}$ at each trial $t = 1, 2, 3, \ldots$ from the teacher, and makes a prediction for $f(x^{(t)})$. The algorithm begins by initializing sets T_{x_i}, which can be regarded as terms. At each trial and for each variable x_i, the term set T_{x_i} of the algorithm will be a superset of the set of variables that belong to the term $T^f_{x_i}$ in f that contains x_i. The initial set T_{x_i} is $\{x_1, x_2, \ldots, x_n\}$ for every i, which corresponds to $x_1 \wedge x_2 \wedge \cdots \wedge x_n$, i.e. to a full term. We will use the notation of terms interchangeably with these sets, e.g. $T_{x_j}(x^{(t)})$ denotes whether all the variables of the assignment $x^{(t)}$ that belong to T_{x_j} are satisfied.

In the algorithm we have the following eight cases:

Case I: $T_{x_i} = \emptyset$. Step 6 in the algorithm. In this case x_i is not a relevant variable so flipping x_i will not change the value of the target. So the algorithm predicts $h(x^{(t)}) = f(x^{(t-1)})$. No mistake will be received.

Case II: $f(x^{(t-1)}) = 0$, $x_i^{(t-1)} = 1$ and $x_i^{(t)} = 0$. Step (7a) in the algorithm. In this case $x^{(t)} < x^{(t-1)}$ and since f is monotone $f(x^{(t)}) = 0$. So the algorithm predicts 0. No mistake will be received.

Case III: $f(x^{(t-1)}) = 0$, $x_i^{(t-1)} = 0$, $x_i^{(t)} = 1$ and $T_{x_i}(x^{(t)}) = 1$. Step (7(b)i) in the algorithm. Since T_{x_i} is a superset of $T_{x_i}^f$ in f and $T_{x_i}(x^{(t)}) = 1$ then $T_{x_i}^f(x^{(t)}) = 1$ (if it exists in f) and $f(x^{(t)}) = 1$. So the algorithm predicts 1. If a mistake is received by the teacher then the algorithm knows that f is independent of x_i and then it sets $T_{x_i} \leftarrow \emptyset$ and removes x_i from all the other terms.

Case IV: $f(x^{(t-1)}) = 0$, $x_i^{(t-1)} = 0$, $x_i^{(t)} = 1$ and $T_{x_i}(x^{(t)}) = 0$. Step (7(b)ii) in the algorithm. Notice that since $f(x^{(t-1)}) = 0$, all the terms in f are 0 in $x^{(t-1)}$ and in particular $T_{x_i}^f(x^{(t-1)}) = 0$. If flipping the bit x_i from 0 to 1 changes the value of the function f to 1 then $T_{x_i}^f(x^{(t)}) = 1$. The algorithm predicts 0. In case of a mistake, we have $T_{x_i}(x^{(t)}) = 0$ and $T_{x_i}^f(x^{(t)}) = 1$ and therefore we can remove every variable x_j in T_{x_i} that satisfies $x_j^{(t)} = 0$. Notice that there is at least one such variable, and that after removing all such variables the condition that T_{x_i} is a superset of $T_{x_i}^f$ still holds. Also, if x_k is not in $T_{x_i}^f$ then x_i is not in $T_{x_k}^f$, so we can also remove x_i from any such set T_{x_k}.

Case V: $f(x^{(t-1)}) = 1$, $x_i^{(t-1)} = 0$ and $x_i^{(t)} = 1$. Step (8a) in the algorithm. In this case $x^{(t)} > x^{(t-1)}$ and since f is monotone $f(x^{(t)}) = 1$. So the algorithm predicts 1. No mistake will be received.

Case VI: $f(x^{(t-1)}) = 1$, $x_i^{(t-1)} = 1$, $x_i^{(t)} = 0$ and there is k such that $T_{x_k}(x^{(t)}) = 1$. Step (8(b)i) in the algorithm. This is similar to Case III.

Case VII: $f(x^{(t-1)}) = 1$, $x_i^{(t-1)} = 1$, $x_i^{(t)} = 0$, for every k, $T_{x_k}(x^{(t)}) = 0$ and $T_{x_i}(x^{(t-1)}) = 0$. Step (8(b)ii) in the algorithm. In this case if $f(x^{(t)}) = 0$ then since $f(x^{(t-1)}) = 1$, we must have $T_{x_i}^f(x^{(t-1)}) = 1$. So this is similar to Case IV.

Case VIII: $f(x^{(t-1)}) = 1$, $x_i^{(t-1)} = 1$, $x_i^{(t)} = 0$, for every k, $T_{x_k}(x^{(t)}) = 0$ and $T_{x_i}(x^{(t-1)}) = 1$. Step (8(b)iii) in the algorithm. In this case the algorithm can be in two modes, "A" or "B". The algorithm begins in mode "A", which assumes that T_{x_k} is correct, i.e. $T_{x_k}^f = T_{x_k}$ for every k. With this

assumption $f(x^{(t)}) = \vee_k T^f_{x_k}(x^{(t)}) = \vee_k T_{x_k}(x^{(t)}) = 0$ and the algorithm predicts 0. In case of a prediction mistake, we alternate between mode "A" and mode "B", where mode "B" assumes the opposite, i.e. it assumes that our lack of knowledge prevents us from seeing that some terms are indeed satisfied, so when we don't know whether some terms are satisfied while operating under mode "B", we assert that they are satisfied and set the algorithm to predict 1.

The most extreme possibility that requires mode "A" in order not to make too many mistakes is in case $f(x_1, x_2, \ldots, x_n) = x_1 \wedge x_2 \wedge \cdots \wedge x_n$. The most extreme possibility that requires mode "B" in order not to make too many mistakes is in case $f(x_1, x_2, \ldots, x_n) = x_1 \vee x_2 \vee \cdots \vee x_n$. After the algorithm has completed the learning and $h \equiv f$, it will always remain in mode "A", as the sets T_{x_i} will be accurate.

5.1.1 Correctness of ROM-DNF-L(δ)

We will find a $p = poly(n, \log \frac{1}{\delta})$ such that the probability of ROM-DNF-L(δ) making more than p mistakes is less than δ.

We note that the only prediction mistakes that ROM-DNF-L(δ) makes in which no new information is gained occur at step (8(b)iii). We will now bound the ratio between the number of assignments that could cause noninformative mistakes and the number of assignments that could cause informative mistakes during any stage of the learning process.

An assignment $x^{(t)}$ is called an *informative assignment* at trial t if there exists $x^{(t-1)}$ such that $x^{(t-1)} \rightarrow x^{(t)}$ is a possible random walk that forces the algorithm to make a mistake and to eliminate at least one variable from one of the term sets. An assignment $x^{(t)}$ is called a *noninformative assignment* at trial t if there exists $x^{(t-1)}$ such that $x^{(t-1)} \rightarrow x^{(t)}$ is a possible random walk that forces the algorithm to make a mistake in step (8(b)iii). Notice that $x^{(t)}$ can be informative and noninformative at the same time.

At trial t, let N be the number of informative assignments and N_A and N_B be the number of noninformative assignments in case the algorithm operates in mode "A" and "B", respectively. We want to show that $\min(N_A/N, N_B/N) \leq N_0$ for some constant N_0. This will show that for at least one of the modes "A" or "B", there is a constant probability that a prediction mistake can lead to progress in the learning, and thus the algorithm

ROM-DNF-L(δ):

1. For each variable x_i, $1 \leq i \leq n$, create the set $T_{x_i} \leftarrow \{x_1, x_2, \ldots, x_n\}$
2. $MODE \leftarrow$ "A"
3. **First Trial**: Make an arbitrary prediction for the value of $f(x^{(1)})$
4. **Trial t:** See if the teacher sent a "mistake" message in the previous trial, and thus determine $f(x^{(t-1)})$
5. Find the variable x_i on which the assignments $x^{(t-1)}$ and $x^{(t)}$ differ
6. If $T_{x_i} = \emptyset$ (meaning: x_i isn't relevant), then predict $h(x^{(t)})$=$f(x^{(t-1)})$
7. Otherwise, if $f(x^{(t-1)}) = 0$
   (a) If x_i flipped $1 \rightarrow 0$, then predict 0
   (b) Otherwise, x_i flipped $0 \rightarrow 1$
      i. If $T_{x_i}(x^{(t)}) = 1$, then predict 1
      On mistake do: $T_{x_i} \leftarrow \emptyset$, and update the other term sets by removing x_i from them.
      ii. Otherwise, predict 0
      On mistake do: update the set T_{x_i} by removing the unsatisfied variables of $x^{(t)}$ from it, since they are unneeded, and update the rest of the term sets by removing x_i from any term set T_{x_k} such that x_k was an unneeded variable in T_{x_i}
8. Otherwise, $f(x^{(t-1)}) = 1$
   (a) If x_i flipped $0 \rightarrow 1$, then predict 1
   (b) Otherwise, x_i flipped $1 \rightarrow 0$
      i. If some $T_{x_k}(x^{(t)}) = 1$, then predict 1
      On mistake do: for each k such that $T_{x_k}(x^{(t)}) = 1$, do $T_{x_k} \leftarrow \emptyset$, and remove the irrelevant variable x_k from the rest of the term sets
      ii. Otherwise, if $T_{x_i}(x^{(t-1)}) = 0$, then predict 1
      On mistake do: update the set T_{x_i} by removing the unsatisfied variables of $x^{(t-1)}$ from it, since they are unneeded, and update the rest of the term sets by removing x_i in any term set T_{x_k} such that x_k was an unneeded variable in T_{x_i}
      iii. Otherwise, if $MODE =$ "A", then predict 0
      On mistake do: $MODE \leftarrow$ "B"
      Otherwise, $MODE =$ "B", then predict 1
      On mistake do: $MODE \leftarrow$ "A"
9. Goto 4

Figure 5.1: The ROM-DNF-L(δ) Algorithm - ROM-DNF Learner

achieves a polynomial mistake bound.

At trial t, let $f = f_1 \vee f_2$ where

1. $f_1 = \hat{T}_1^f \vee \hat{T}_2^f \vee \cdots \vee \hat{T}_{k_1}^f$ are the terms in f where for every term $\hat{T}_\ell^f$ there exists a variable x_j in that term such that $T_{x_j} = \hat{T}_\ell^f$. Those are the terms that have been discovered by the algorithm.

2. $f_2 = T_1^f \vee T_2^f \vee \cdots \vee T_{k_2}^f$ are the terms in f where for every term T_ℓ^f and every variable x_j in that term, we have that T_{x_j} is a proper super-term of T_ℓ^f. Those are the terms of f that haven't been discovered yet by the algorithm. In other words, for each variable x_i that belongs to such a term, the set T_{x_i} contains superfluous variables.

Denote by V_1 and V_2 the set of variables of f_1 and f_2, respectively, and let V_3 be the set of irrelevant variables. Let $a_\ell = |\hat{T}_\ell^f|$ be the number of variables in $\hat{T}_\ell^f$, $b_\ell = |T_\ell^f|$ be the number of variables in T_ℓ^f, and $d = |V_3|$ be the number of irrelevant variables.

First, let us assume that the algorithm now operates in mode "A". Noninformative mistakes can occur only when: $f(x^{(t-1)}) = 1$, $x_i^{(t-1)} = 1$, $x_i^{(t)} = 0$, for every k, $T_{x_k}(x^{(t)}) = 0$ and $T_{x_i}(x^{(t-1)}) = 1$. The algorithm predict 0 but $f(x^{(t)}) = 1$.

We will bound from above N_A, the number of possible assignments $x^{(t)}$ that satisfy the latter conditions. Since $T_{x_k}(x^{(t)}) = 0$ for every k and for every $\hat{T}_\ell^f$ there is x_j such that $\hat{T}_\ell^f = T_{x_j}$, we must have $\hat{T}_\ell^f(x^{(t)}) = 0$ for every ℓ, and therefore $f_1(x^{(t)}) = 0$. Since $1 = f(x^{(t)}) = f_1(x^{(t)}) \vee f_2(x^{(t)})$, we must have $f_2(x^{(t)}) = 1$. Therefore, the number of such assignments is at most

$$\begin{aligned} N_A &\leq |\{x^{(t)} \in X_n \mid f_1(x^{(t)}) = 0 \text{ and } \ f_2(x^{(t)}) = 1\}| \\ &= c2^d \left(\prod_{i=1}^{k_2} 2^{b_i} - \prod_{i=1}^{k_2} (2^{b_i} - 1) \right). \end{aligned}$$

Here $c = \prod_{i=1}^{k_1} (2^{a_i} - 1)$ is the number of assignments to V_1 where $f_1(x) = 0$, 2^d is the number of assignments to V_3, and $\prod_{i=1}^{k_2} 2^{b_i} - \prod_{i=1}^{k_2} (2^{b_i} - 1)$ is the number of assignments to V_2 where $f_2(x) = 1$.

We now show that the number of informative assignments is at least

$$N \geq \frac{1}{2}c2^d \sum_{j=1}^{k_2} \prod_{i \neq j}^{k_2} (2^{b_i} - 1) \tag{5.1}$$

and therefore

$$\begin{aligned} \frac{N_A}{N} &\leq \frac{c2^d \left(\prod_{i=1}^{k_2} 2^{b_i} - \prod_{i=1}^{k_2} (2^{b_i} - 1) \right)}{\frac{1}{2}c2^d \sum_{j=1}^{k_2} \prod_{i \neq j}^{k_2} (2^{b_i} - 1)} \\ &= \frac{2(\prod_{i=1}^{k_2} 2^{b_i} - \prod_{i=1}^{k_2} (2^{b_i} - 1))}{\sum_{j=1}^{k_2} \prod_{i \neq j}^{k_2} (2^{b_i} - 1)}. \end{aligned}$$

To prove (5.1), consider (Case IV) which corresponds to step (7(b)ii) in the algorithm. In case $x^{(t)}$ is informative there exist i and $x^{(t-1)}$ such that $f(x^{(t-1)}) = 0$, $x_i^{(t-1)} = 0$, $x_i^{(t)} = 1$, $T_{x_i}(x^{(t)}) = 0$, and $f(x^{(t)}) = 1$. Notice that since $f(x^{(t-1)}) = 0$, all the terms $T_{x_\ell}^f$ satisfy $T_{x_\ell}^f(x^{(t-1)}) = 0$, and therefore all the term sets T_{x_ℓ} satisfy $T_{x_\ell}(x^{(t-1)}) = 0$. Since $f(x^{(t)}) = 1$ and $x^{(t)}$ differ from $x^{(t-1)}$ only in x_i, it follows that $T_{x_i}^f$ is the only term that satisfies $T_{x_i}^f(x^{(t)}) = 1$.

One case in which this may occur is when $f_1(x^{(t)}) = 0$, and exactly one term $T_{x_i}^f \equiv T_\ell^f$ in f_2 satisfies $x^{(t)}$, and some variable x_j that is in T_{x_i} and is not in $T_{x_i}^f$ is 0 in $x^{(t)}$. We will call such an assignment a *perfect* assignment. An assignment $x^{(t)}$ where $f_1(x^{(t)}) = 0$ and exactly one term $T_{x_i}^f \equiv T_\ell^f$ in f_2 satisfies $x^{(t)}$ is called a *good* assignment. Notice that since f is monotone, for every good assignment $x^{(t)}$ in which every x_j that is in T_{x_i} and is not in $T_{x_i}^f$ is 1 in $x^{(t)}$, we can choose the smallest index j_0 such that x_{j_0} is in T_{x_i} and is not in $T_{x_i}^f$, and flip x_{j_0} to 0 in order to get a perfect assignment. Therefore, the number of perfect assignments is at least 1/2 the number of good assignments.

To count the number of good assignments, note that $\sum_{j=1}^{k} \prod_{i \neq j}^{k} (2^{b_i} - 1)$ is the number of assignments to V_2 in which exactly one of the terms in f_2 is satisfied. As previously denoted, c is the number of assignments to V_1 in which $f_1 = 0$, and 2^d is the number of assignments to the irrelevant variables. This gives (5.1).

Second, let us assume that the algorithm now operates in mode "B". Noninformative mistakes can occur only when: $f(x^{(t-1)}) = 1$, $x_i^{(t-1)} = 1$,

$x_i^{(t)} = 0$, for every k, $T_{x_k}(x^{(t)}) = 0$ and $T_{x_i}(x^{(t-1)}) = 1$. But now the algorithm predict 1 though $f(x^{(t)}) = 0$.

Using the same reasoning, an upper bound for N_B can be obtained when neither f_1 nor f_2 are satisfied, thus

$$N_B \leq |\{x^{(t)} \in X_n \mid f_1(x^{(t)}) = 0 \text{ and } \ f_2(x^{(t)}) = 0\}| = c2^d \prod_{i=1}^{k_2} (2^{b_i} - 1).$$

Therefore we have

$$\begin{aligned} \frac{N_B}{N} &\leq \frac{c2^d \prod_{i=1}^{k_2}(2^{b_i}-1)}{\frac{1}{2}c2^d \sum_{j=1}^{k_2} \prod_{i\neq j}^{k_2}(2^{b_i}-1)} \\ &= \frac{2\prod_{i=1}^{k_2}(2^{b_i}-1)}{\sum_{j=1}^{k_2} \prod_{i\neq j}^{k_2}(2^{b_i}-1)}. \end{aligned}$$

We now show that at least one of the above bounds is smaller than 3. Therefore, in at least one of the two modes, the probability to select a noninformative assignment is at most 3 times greater than the probability to select an informative assignment under the uniform distribution.

Consider

$$w_i := 2^{b_i} - 1, \quad \alpha := \frac{\prod_{i=1}^{k}(w_i+1) - \prod_{i=1}^{k} w_i}{\sum_{j=1}^{k} \prod_{i\neq j}^{k} w_i}, \quad \beta := \frac{\prod_{i=1}^{k} w_i}{\sum_{j=1}^{k} \prod_{i\neq j}^{k} w_i}.$$

Then

$$\beta = \frac{\prod_{i=1}^{k} w_i}{\left(\prod_{i=1}^{k} w_i\right) \sum_{i=1}^{k} \frac{1}{w_i}} = \frac{1}{\sum_{i=1}^{k} \frac{1}{w_i}}$$

and

$$\begin{aligned}
\alpha &= \frac{\prod_{i=1}^{k}(w_i+1)-\prod_{i=1}^{k} w_i}{\left(\prod_{i=1}^{k} w_i\right)\sum_{i=1}^{k}\frac{1}{w_i}} \\
&= \frac{1}{\sum_{i=1}^{k}\frac{1}{w_i}}\left(\frac{\prod_{i=1}^{k}(w_i+1)}{\prod_{i=1}^{k} w_i}-1\right) \\
&= \beta\left(\prod_{i=1}^{k}\left(1+\frac{1}{w_i}\right)-1\right) \\
&\leq \beta\left(\prod_{i=1}^{k} e^{\frac{1}{w_i}}-1\right) \\
&= \beta\left(e^{\sum_{i=1}^{k}\frac{1}{w_i}}-1\right) = \beta(e^{\frac{1}{\beta}}-1).
\end{aligned}$$

Therefore

$$\min\left(\frac{N_A}{N},\frac{N_B}{N}\right) = 2\min(\alpha,\beta) \;\leq\; 2\min(\beta(e^{\frac{1}{\beta}}-1),\beta) = 2\frac{1}{\log 2} < 3.$$

5.1.2 The analysis for δ

For variance, we shall use Lemma 2. However, similarly to Section 4.3, the penalty factor for using Lemma 3 would have been constant here as well.

Let P_U be the probability under the uniform distribution that an assignment that caused a prediction mistake is informative. We have shown that during any trial, in at least one of the modes "A" or "B", we have $P_U \geq \frac{1}{4}$.

For Lemma 2, let us now choose $\varepsilon = \frac{1}{8}$, and thus

$$m = \frac{n+1}{4}\log\frac{n}{\log(2/8^2)+1)} = \frac{n+1}{4}\log(C_0 n), \quad C_0 \approx 32.5\ .$$

When looking at prediction mistakes that occur after at least m trials, we will be ε-close to the uniform distribution. Therefore, in the algorithm the probability P_A that corresponds to P_U is at least

$$P_A \geq P_U - \varepsilon \geq \frac{1}{8}.$$

Let us now analyse a phase of the learning process by considering groups of m consecutive trials each, i.e. $G_1 = \{x^{(t+1)}, x^{(t+2)}, \ldots, x^{(t+m)}\}$, $G_2 = \{x^{(t+m+1)}, x^{(t+m+2)}, \ldots, x^{(t+2m)}\}$, $G_3 = \{x^{(t+2m+1)}, x^{(t+2m+2)}, \ldots, x^{(t+3m)}\}$, and so on, in which w is the longest chain of prediction mistakes that occur at trials whose distance is a multiple of m. Thus, for $w' \geq w$ and $1 \leq j \leq m$ the mistakes in such a chain of m-leaps $\{x^{(t+im+j)}\}_{i=0}^{w'-1}$ are not necessarily consecutive, but the total number of mistakes for a phase $G_1, G_2, \ldots, G_{w'}$ is still wm at the most. For such a phase, let us assume that only noninformative mistakes occured, let $\hat{a}$ denote the maximal number of mode "A" mistakes that occured in a certain chain of m-leaps, and let $\hat{b}$ denote the maximal number of mode "B" mistakes that occured in a certain chain of m-leaps. Thus, the total number of mode "A" mistakes is no more than $\hat{a}m$, and the total number of mode "B" mistakes is no more than $\hat{b}m$. Since there are at least w mistakes, and since the algorithm alternates between modes after each noninformative mistake, it follows that there are at least $\frac{w-1}{2}$ mistakes for each mode, and therefore

$$min(\hat{a}m, \hat{b}m) \geq \frac{w-1}{2} \implies min(\hat{a}, \hat{b}) \geq \frac{w-1}{2m}.$$

Let us consider a chain of prediction mistakes that occur while under a mode with the bounded uniform distribution failure probablity, and note that the probability that a noninformative mistake indeed occurs in each trial is $(1 - \frac{1}{n}P_A)$ at the most. This is because the probability that a variable whose flip in the previous trial would cause an informative mistake is at least $\frac{1}{n}P_A$. Therefore, the probability of having q consecutive mistakes in this mode is at most

$$\left(1 - \frac{1}{n}P_A\right)^q = \left(1 - \frac{1}{8n}\right)^q.$$

In order to obtain a suitable bound by finding q that is large enough, we require

$$\left(1 - \frac{1}{8n}\right)^q \leq \frac{\delta}{n^2},$$

and therefore

$$q = 8n\left(2\log n + \log\frac{1}{\delta}\right).$$

If we now constrain w so that $\frac{w-1}{2m} \geq q$, we obtain for $w = 2mq + 1$ that $min(\hat{a}, \hat{b}) \geq \frac{w-1}{2m} = q$. This implies that $w = 2mq + 1$ is a sufficient bound for a phase, meaning that the probability of failure to gain information after a phase is $\frac{\delta}{n^2}$ at the most, and in each such phase there are no more than $wm = (2mq + 1)m$ prediction mistakes.

We now get

$$\begin{aligned}
\Pr(\{\text{ROM-DNF-L}(\delta) \text{ fails}\}) &\leq \Pr(\{\text{phase 1 fails}\} \vee \ldots \vee \{\text{phase } n^2 \text{ fails}\}) \\
&\leq \sum_{i=1}^{n^2} \Pr(\{\text{phase } i \text{ fails}\}) \\
&\leq n^2 \frac{\delta}{n^2} \;=\; \delta,
\end{aligned}$$

and the total number of prediction mistakes that ROM-DNF-L(δ) makes is bounded by

$$\begin{aligned}
n^2 wm &= n^2(2mq+1)m \\
&= n^2 2 \left(\frac{n+1}{4}\log(C_0 n)\right)^2 8n \left(2\log n + \log\frac{1}{\delta}\right) + n^2 \left(\frac{n+1}{4}\log(C_0 n)\right) \\
&= poly(n)\log\frac{1}{\delta}.
\end{aligned}$$

□

5.2 Learning Read-Once DNF

With a small modification to the ROM-DNF-L(δ) algorithm, learning non-monotone functions is possible as well. That is, it is also possible to learn the *read-once* DNF (RO-DNF) class in the URWOnline model.

On the first step of the algorithm, we initialize T_{x_i} to $\{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \ldots, \tilde{x}_n\}$, meaning that we do not know yet whether the variables of the term that contains x_i are negated or not. We now always predict $h(x^{(t)}) = f(x^{(t-1)})$ when x_i flips and T_{x_i} contains variables that are marked as unknown. If we

make a mistake on such predictions, we can immediately update variables in relation to x_i as follows:

- if $f(x^{(t)}) = x_i^{(t)}$ then $T_{x_i} \leftarrow (T_{x_i} \setminus \{\tilde{x}_i\}) \cup \{x_i\}$ else $T_{x_i} \leftarrow (T_{x_i} \setminus \{\tilde{x}_i\}) \cup \{\bar{x}_i\}$
- for each $j \neq i$
  - if $f(x^{(t)}) = x_i^{(t)}$
    - if $\bar{x}_i \in T_{x_j}$ then $T_{x_j} \leftarrow T_{x_j} \setminus \{\bar{x}_i\}$ else $T_{x_j} \leftarrow (T_{x_j} \setminus \{\tilde{x}_i\}) \cup \{x_i\}$
  - else $f(x^{(t)}) \neq x_i^{(t)}$
    - if $x_i \in T_{x_j}$ then $T_{x_j} \leftarrow T_{x_j} \setminus \{x_i\}$ else $T_{x_j} \leftarrow (T_{x_j} \setminus \{\tilde{x}_i\}) \cup \{\bar{x}_i\}$
- for each $j \neq i$
  - if $x_j^{(t)} = 1$
    - if $\bar{x}_j \in T_{x_i}$ then $T_{x_i} \leftarrow T_{x_i} \setminus \{\bar{x}_j\}$ else $T_{x_i} \leftarrow (T_{x_i} \setminus \{\tilde{x}_j\}) \cup \{x_j\}$
  - else $x_j^{(t)} = 0$
    - if $x_j \in T_{x_i}$ then $T_{x_i} \leftarrow T_{x_i} \setminus \{x_j\}$ else $T_{x_i} \leftarrow (T_{x_i} \setminus \{\tilde{x}_j\}) \cup \{\bar{x}_j\}$

The rest of the ROM-DNF-L(δ) algorithm should be modified in the obvious way.

Now the earlier note about flipping one bit in a good assignment in order to obtain a perfect assignment no longer holds, as the flip might cause another term to become true. However, for a good assignment $x^{(t)}$ in which a flip of each superfluous bit $lit(x_j) \in T_{x_i}$ satisfies $T_{x_j}^f$, it holds that x_j is negated in T_{x_i} or in $T_{x_j}^f$, but not in both, and therefore $\tilde{x}_j \in T_{x_j}$. Thus, with probability $\frac{1}{n^2}$ we will gain information at trial $t+2$, in case $x^{(t)} \xrightarrow{lit(x_i)\ 1\to 0} x^{(t+1)} \xrightarrow{lit(x_j)\ 0\to 1} x^{(t+2)}$, i.e. $f(x^{(t+1)}) = 0$ and $T_{x_j}^f(x^{(t+2)}) = 1$.

Therefore, the number of informative assignments during any stage of the learning process is at least $N \geq \frac{1}{2}(g-b) + b \geq \frac{1}{2}g$, where g is the total number of good assignments and b is the number of good assignments that cannot be made perfect by a single bit flip. It follows that for the same ratio calculations the probability of making an informative mistake on either $x^{(t)}$ or $x^{(t+2)}$ is at least $\frac{1}{n^2}P_A$, and since the maximal number of updates for each term is still n at the most, polynomial bounds similar to those in the ROM-DNF analysis are maintained. □

Chapter 6

URWOnline Limitations

We have shown that under the widely believed assumption that DNF formulas are not Online learnable, the URWOnline model is easier than the Online model. We have also shown that under the reasonably moderate assumption that $\mathcal{O}(\log n)$-juntas are not UROnline learnable, the URWOnline model is easier than the UROnline model. In other words, these results indicate that some classes can be learned in the URWOnline model, but not in these two more generic models. Could we expect the learner in the URWOnline model to be powerful enough to learn any reasonable concept class? In particular, is it possible to learn the class of all DNF formulas in the URWOnline model? We now answer these questions in the negative, under extra assumptions. Specifically, we prove that DNF learnability in the URWOnline model implies DNF learnability in the UROnline model, in case the following two conditions hold with regard to the URWOnline DNF learning algorithm that is assumed to exist:

- The URWOnline algorithm does not modify its state after a successful prediction. Notice that this assumption typically holds for Online learning algorithms, including the URWOnline algorithms in this work.

- The URWOnline algorithm is given as a white box, which is susceptive to manual inclusion of new knowledge. This means that the prediction algorithm can be given additional information in an efficient manner, and if this information is consistent with the target function, then all subsequent predictions will retain and utilize this extra information.

For example, suppose that the URWOnline DNF algorithm maintains a list of terms, similarly to the ROM-DNF-L algorithm that we presented in the previous section. As an inclusion of new knowledge, we can simply add some of the terms of the target function to the hypothesis, and if the hypothesis only makes predictions that are consistent with its terms list, and only updates terms that are inconsistent with its last prediction, then the 2nd condition holds.

While it is not trivial to assume the 2nd condition, for certain learning algorithms it is quite natural. Particularly, in the example mentioned, observe that this condition indeed holds for ROM-DNF-L, because it never makes a prediction that is inconsistent with its current terms list, and upon a prediction mistake it either updates terms that are inconsistent with its last prediction, or refrains from updating any of the terms (and instead updates an auxiliary $MODE$ variable).

Given the above assumptions, denote by RWDNF-L(n, δ) the algorithm that learns any DNF formula $f : X_n \to \{0,1\}$ in $poly(n, size_{\text{DNF}}(f), \frac{1}{\delta})$ time in the URWOnline model. An algorithm UDNF-L(n, δ) that learns an arbitrary $f(x_1, x_2, \ldots, x_n)$ in the UROnline model will operate as follows. Let $k = 9n$, and consider the following DNF formula

$$g(\overbrace{x_{11}, x_{12}, \ldots, x_{1k}}, \overbrace{x_{21}, x_{22}, \ldots, x_{2k}}, \ldots, \overbrace{x_{n1}, x_{n2}, \ldots, x_{nk}}) \triangleq$$
$$f(x_{11}, x_{21}, \ldots, x_{n1}) \vee (\bigvee_{i \neq j}(x_{1i} \wedge \bar{x}_{1j})) \vee (\bigvee_{i \neq j}(x_{2i} \wedge \bar{x}_{2j})) \vee \ldots \vee (\bigvee_{i \neq j}(x_{ni} \wedge \bar{x}_{nj})).$$

Notice that $size_{\text{DNF}}(g) \leq size_{\text{DNF}}(f) + \mathcal{O}(k^2 n)$, and that

$$g \equiv \begin{cases} 1 & \exists i_0, i_1, i_2 : x_{i_0 i_1} \neq x_{i_0 i_2} \\ f(x_{11}, x_{21}, \ldots, x_{n1}) & \text{otherwise} \end{cases} .$$

Initially, UDNF-L(n, δ) will insert RWDNF-L$(kn, \frac{\delta}{2})$ the knowledge to predict 1 on all queries in which $\exists x_{i_0 i_1} \neq x_{i_0 i_2}$, e.g. by adding the $k(k-1)n$ needed terms to the RWDNF-L$(kn, \frac{\delta}{2})$ white box. Then, UDNF-L(n, δ) will begin to simulate RWDNF-L$(kn, \frac{\delta}{2})$ on g, simply by taking each uniformly distributed query received from the actual teacher, choosing randomly uniformly an index $1 \leq i \leq kn$ as if it was the last to flip, duplicating each variable k times, invoking RWDNF-L$(kn, \frac{\delta}{2})$ on each such expanded query, returning the prediction of RWDNF-L$(kn, \frac{\delta}{2})$ to the teacher, and updating

the hypothesis according to the RWDNF-L($kn, \frac{\delta}{2}$) algorithm in case of a prediction mistake.

Because RWDNF-L($kn, \frac{\delta}{2}$) neither makes mistakes nor updates its state whenever $\exists x_{i_0 i_1} \neq x_{i_0 i_2}$, this simulation makes the assumption that every time that the random walk stochastic process reaches an assignment for which $\nexists x_{i_0 i_1} \neq x_{i_0 i_2}$, and RWDNF-L($kn, \frac{\delta}{2}$) makes a prediction mistake on it, that assignment is uniformly distributed. We will show that this assumption holds with a very high probability. Note that RWDNF-L($kn, \frac{\delta}{2}$) determines how to predict according to $g(x^{(t-1)})$, the flipped bit, $x^{(t)}$, and its state. This poses no problems for the simulation, because at each invocation the value $g(x^{(t-1)}) = 1$ is known to RWDNF-L($kn, \frac{\delta}{2}$), its state remained constant, and $x^{(t)}$ and the flipped bit are provided by UDNF-L(n, δ). Thus, if $\mathcal{A}_{\text{bad-dist}}$ denotes the event that this assumption failed to hold, we have

$$\begin{aligned}
&\Pr(\{\text{UDNF-L(n,}\delta\text{) fails, i.e. makes more than } poly(size_{\text{DNF}}(f), kn, \tfrac{2}{\delta}) \text{ mistakes}\}) \\
&= \Pr(\{\text{UDNF-L(n,}\delta\text{) fails}\} \cap \mathcal{A}_{\text{bad-dist}}) + \Pr(\{\text{UDNF-L(n,}\delta\text{) fails}\} \cap \overline{\mathcal{A}_{\text{bad-dist}}}) \\
&\leq \Pr(\mathcal{A}_{\text{bad-dist}}) + \Pr(\{\text{UDNF-L(n,}\delta\text{) fails}\} \mid \overline{\mathcal{A}_{\text{bad-dist}}}) \\
&= \Pr(\mathcal{A}_{\text{bad-dist}}) + \Pr(\{\text{RWDNF-L}(kn, \tfrac{\delta}{2}) \text{ fails}\}) \\
&\leq \Pr(\mathcal{A}_{\text{bad-dist}}) + \frac{\delta}{2}
\end{aligned}$$

We now prove that $\Pr(\mathcal{A}^1_{\text{bad-dist}}) \ll \frac{1}{2^{3n}}$, where $\mathcal{A}^1_{\text{bad-dist}}$ denotes the event that the random walk process reached a certain non-uniformly distributed assignment for which $\nexists x_{i_0 i_1} \neq x_{i_0 i_2}$, which is different than the last assignment for which the condition $\nexists x_{i_0 i_1} \neq x_{i_0 i_2}$ held. Since the assignments are different, suppose that $x_{d1} = x_{d2} = \ldots = x_{dk}$ were the last group of k variables to reach the same value, which is different than their previous value.

Let us define the following two random variables,

$$\begin{aligned}
Y &\triangleq \{\text{number of steps on } x_{11}, \ldots, x_{nk} \text{ (without } x_{d1}, \ldots, x_{dk}\text{) until all were selected}\} \\
Z &\triangleq \{\text{number of steps on } x_{d1}, \ldots, x_{dk} \text{ until reaching } x_{d1} = x_{d2} = \ldots = x_{dk} \text{ flipped}\}
\end{aligned}$$

That is, during the lazy random walk on $\{x_{11}, \ldots, x_{nk}\}$, Z counts only the steps in which variables from $\{x_{d1}, \ldots, x_{dk}\}$ were selected, thus it effectively counts the time to reach the exact opposite assignment while walking on the hypercube HYP_k of k variables. Likewise, Y counts only steps in which

variables from $\{x_{11},\ldots,x_{nk}\}\setminus\{x_{d1},\ldots,x_{dk}\}$ were selected, until all of them were selected (and flipped with probability $\frac{1}{2}$), i.e. until the uniform distribution on $\{x_{11},\ldots,x_{nk}\}\setminus\{x_{d1},\ldots,x_{dk}\}$ has been reached.

Our proof relies on the fact that $\mathrm{Ex}[Z]$ is exponential in k. This follows from the observation that the expected return-time of the simple random walk on HYP_k is exactly 2^k. The simple random walk on any graph is a time-homogeneous Markov chain such that with each transition we travel to a vertex that is uniformly selected among the vertices incident to the current vertex. Let us first make the well known yet remarkable observation that in an infinite random walk on a connected graph, each edge will be traversed the same proportion of the time. This follows from the fact that the probability of being at any particular vertex is proportional to its degree. More precisely, for any connected graph $G=(V,E)$ with transition matrix P, the vector π whose elements are $\pi_v=\frac{deg(v)}{2|E|}$ is the stationary distribution. To see this, observe that

$$\sum_{x\in V} deg(x)P(x,y)=\sum_{(x,y)\in E}\frac{deg(x)}{deg(x)}=deg(y).$$

Thus for $\tilde{\pi}=(deg(v_1),deg(v_2),\ldots,deg(v_{|V|})$ it holds that $\tilde{\pi}=\tilde{\pi}P$, and therefore the normalized probability vector $\pi=\frac{\tilde{\pi}}{2|E|}$ is the stationary distribution. This means (cf. [LPW09]) that the expected return-time of any vertex $v\in V$, i.e. the expectation of the number of steps to reach v in a simple random walk that originates from v, is $\frac{1}{\pi_v}=\frac{2|E|}{deg(v)}$.

Now, for the hypercube HYP_k with 2^k vertices, the expected return-time for any vertex is $\frac{2\cdot\frac{1}{2}k2^k}{k}=2^k$. Therefore, if we denote by $hit_k(i)$ the expected time to reach $\overrightarrow{0}=\overbrace{(0,0,\ldots,0)}^{k}$ from an assignment with exactly i bits whose value is 1, then $hit_k(i)$ is monotone increasing in i, and it holds for the return time $ret_k(\overrightarrow{0})$ that $2^k=ret_k(\overrightarrow{0})=1+hit_k(1)$. Thus $hit_k(1)=2^k-1$, and therefore $\mathrm{Ex}[Z]=\underbrace{2}_{\text{lazy walk}}\cdot\ hit_k(k)>2\cdot hit_k(1)>2^k$, as claimed.

Let Y' denote the number of steps on $\{x_{11},\ldots,x_{nk}\}\setminus\{x_{d1},\ldots,x_{dk}\}$ until $\{x_{d1},\ldots,x_{dk}\}$ reached the exact opposite assignment. Now,

$$\begin{aligned}
\Pr(\overline{\mathcal{A}^1_{\text{bad-dist}}}) &\geq \Pr(\overline{\mathcal{A}^1_{\text{bad-dist}}} \mid Y' > 2^{5n}) \cdot \Pr(Y' > 2^{5n}) \\
&\geq \Pr(Y < 2^{5n}) \cdot \Pr(Y' > 2^{5n}) \\
&\geq (1 - \frac{\text{Ex}[Y]}{2^{5n}}) \cdot \Pr(Y' > 2^{5n}) \\
&> (1 - \frac{9n^2\log(9n^2)}{2^{5n}}) \cdot \Pr(Y' > 2^{5n} \mid Z > 2^{5n}) \cdot \Pr(Z > 2^{5n}) \\
&> (1 - \frac{1}{2^{4n}}) \cdot \Pr(Y' > 2^{5n} \mid Z = 2^{5n}) \cdot \Pr(Z > 2^{5n})
\end{aligned}$$

Notice that for $X \sim \text{NegBin}(2^{5n}, \frac{1}{n})$, we have

$$\begin{aligned}
\Pr(Y' \leq 2^{5n} \mid Z = 2^{5n}) = \Pr(X \leq 2^{5n}), \\
\text{Ex}[X] = 2^{5n}(n-1),\ \text{Var}[X] = 2^{5n}(n-1)n.
\end{aligned}$$

Thus, by Chebyshev's inequality,

$$\begin{aligned}
\Pr(X \leq 2^{5n}) &\leq \Pr(\text{Ex}[X] - X \geq \frac{1}{2}\text{Ex}[X]) \\
&\leq \Pr(|X - \text{Ex}[X]| \geq \frac{1}{2}\text{Ex}[X]) \leq \frac{2^{5n}(n-1)n}{\frac{1}{4}2^{10n}(n-1)^2} < \frac{1}{2^{4n}}.
\end{aligned}$$

Finally, since we only calculated the expectation for the random variable Z, in order to obtain a lower bound for it, we shall use a "reversed" variation of the Markov inequality (cf. Appendix C):

$$\Pr(Z \leq 2^{5n} | Z \leq 2^{9n} + 2^{5n}) \leq \frac{2^{9n} + 2^{5n} - \text{Ex}[Z]}{2^{9n} + 2^{5n} - 2^{5n}} < \frac{2^{9n} + 2^{5n} - 2^{9n}}{2^{9n} + 2^{5n} - 2^{5n}} = \frac{1}{2^{4n}}.$$

And therefore,

$$\Pr(Z \leq 2^{5n}) < \frac{\Pr(Z \leq 2^{5n})}{\Pr(Z \leq 2^{9n} + 2^{5n})} = \Pr(Z \leq 2^{5n} | Z \leq 2^{9n} + 2^{5n}) < \frac{1}{2^{4n}}.$$

Let us note that by symmetry, the index d is uniformly distributed. Thus, if we combine the bounds that we calculated, we obtain

$$\Pr(\overline{\mathcal{A}^1_{\text{bad-dist}}}) > (1 - \frac{1}{2^{4n}})^3 \approx 1 - \frac{3}{2^{4n}} > 1 - \frac{4}{2^{4n}} \gg 1 - \frac{1}{2^{3n}}.$$

To bound $\Pr(\overline{\mathcal{A}_{\text{bad-dist}}})$, we require that the random walk process reaches the uniform distribution between each invocation of RWDNF-L($kn, \frac{\delta}{2}$) that causes a prediction mistake, and the previous invocation. Thus, if the polynomial mistake bound of RWDNF-L($kn, \frac{\delta}{2}$) is $p = poly(size_{DNF}(f), kn, \frac{2}{\delta})$, we now have

$$\begin{aligned}\Pr(\mathcal{A}_{\text{bad-dist}}) &\leq \Pr(\{\text{non-uniform at } 1^{st} \text{ mistake}\} \cup \ldots \cup \{\text{non-uniform at } p^{th} \text{ mistake}\}) \\ &\leq \sum_{j=1}^{p} \Pr(\{\text{non-uniform at } j^{th} \text{ mistake}\}) \ll \frac{p}{2^{3n}} < \frac{1}{2^{2n}}.\end{aligned}$$

For simplicity, we assumed that $\frac{1}{\delta} = o(2^n)$, but in fact any $\delta = \frac{1}{2^{poly(n)}}$ can be handled by choosing a bigger $k = poly(n)$.

Thus, we obtained overall that

$$\Pr(\{\text{UDNF-L(n,}\delta\text{) fails}\}) \leq \Pr(\mathcal{A}_{\text{bad-dist}}) + \frac{\delta}{2} \ll \frac{1}{2^{2n}} + \frac{\delta}{2} < \delta. \quad \square$$

6.1 Assuming Read-3 DNF Learnability

By a slight modification to the previous proof, we can now obtain DNF learnablity in UROnline under the weaker assumption that read-3 DNF is URWOnline learnable. The read-k DNF notation denotes DNF formulas in which every variable appears at most k times.

Suppose $f(x_1, x_2, \ldots, x_n) = T_1 \vee T_2 \vee \ldots \vee T_q$ is a general DNF formula that consists of q terms, where q is unknown, and define

$$\begin{aligned} R3(f,k) &\triangleq f'(\overbrace{x_{11}, x_{12}, \ldots, x_{1k}}, \overbrace{x_{21}, x_{22}, \ldots, x_{2k}}, \ldots, \overbrace{x_{n1}, x_{n2}, \ldots, x_{nk}}) \\ &= \tilde{T}_1 \vee \tilde{T}_2 \vee \ldots \vee \tilde{T}_q \vee x_{11}\bar{x}_{12} \vee x_{12}\bar{x}_{13} \vee \ldots \vee x_{1(k-1)}\bar{x}_{1k} \vee x_{1k}\bar{x}_{11} \\ &\quad \vee x_{21}\bar{x}_{22} \vee x_{22}\bar{x}_{23} \vee \ldots \vee x_{2(k-1)}\bar{x}_{2k} \vee x_{2k}\bar{x}_{21} \\ &\quad \vee \ldots \\ &\quad \vee x_{n1}\bar{x}_{n2} \vee x_{n2}\bar{x}_{n3} \vee \ldots \vee x_{n(k-1)}\bar{x}_{nk} \vee x_{nk}\bar{x}_{n1}, \end{aligned}$$

where each $\tilde{T}_i$ contains only variables from $\{x_{1i}, x_{2i}, \ldots, x_{ni}\}$, corresponding to the variables from $\{x_1, x_2, \ldots, x_n\}$ that T_i contains.

Note that $R3(f,k)$ is a read-3 DNF formula of size $\mathcal{O}(q + kn)$, and that for $k \geq q$ we have

$$R3(f,k) \equiv \begin{cases} 1 & \exists i_0, i_1, i_2 : x_{i_0 i_1} \neq x_{i_0 i_2} \\ f(x_{11}, x_{21}, \ldots, x_{n1}) & \text{otherwise} \end{cases}.$$

Let RWR3-L(n,δ) be the read-3 DNF algorithm for the URWOnline model that is assumed to exist, and let $p(n,\delta)$ be its polynomial mistake bound. Note that $size_{\text{DNF}}(g(x_1, \ldots, x_n)) \leq 3n$ if g is a read-3 DNF formula.

Now, UDNF-L(n,δ) can start by guessing that $k = 9n$ is sufficient, and invoke RWR3-L$(kn, \frac{\delta}{2})$ as in the previous proof. Upon making more than $p(kn, \frac{\delta}{2})$ prediction mistakes, UDNF-L(n,δ) will replace k with $2k$ and restart the process.

Eventually, UDNF-L(n,δ) will invoke RW3DNF-L$(kn, \frac{\delta}{2})$ with $k \geq q$, and in this invocation $k \leq 2q$, the failure probability is small than δ, and the sum of all the prediction mistakes up to and including this invocation is bounded by

$$\begin{aligned} p(9n^2, \frac{2}{\delta}) + p(18n^2, \frac{2}{\delta}) + \ldots + p(2q, \frac{2}{\delta}) &< p(9n^2, \frac{2}{\delta}) + p(2q, \frac{2}{\delta}) \cdot \log q \\ &= poly(q, n, \frac{1}{\delta}). \end{aligned}$$

Therefore, with probability of at least $1 - \delta$ it holds that UDNF-L(n,δ) makes less than $poly(size_{\text{DNF}}(f), n, \frac{1}{\delta})$ prediction mistakes, thus achieving UROnline learnability. $\square$

Moreover, we note that instead of repeated invocations of RWR3-L($kn, \frac{\delta}{2}$) with increasingly bigger values of k, we may simply assume that $q \leq n^{d_0}$ for some known constant d_0, and invoke RWR3-L($kn, \frac{\delta}{2}$) just once with $k = n^{d_0}$. Such an assumption is legitimate for reductions with negative implications, because DNF learnability is an open question even if a polynomial bound n^{d_0} is known to the learner. Observe that under this assumption, knowing the polynomial mistake bound $p(n, \delta)$ of RWR3-L(n, δ) is no longer needed.

Chapter 7

Open Questions

In Chapter 5, we have shown that the class of read-once DNF formulas is learnable in the URWOnline model. In Chapter 6, we have shown that under fairly credible assumptions the class of read-3 DNF formulas cannot be learned in the URWOnline model, unless DNF is learnable in the UROnline model (which would imply DNF learnability in the uniform PAC model, see Appendix B).

There remains a gap with regard to whether the class of read-twice DNF formulas is learnable in the URWOnline model. We conjecture that the answer is yes (compare with [AL91]).

Appendix A

We describe here why the implication UROnline ⇒ URWOnline is correct.

First let us note that the equivalence between Angluin's Exact model and Littlestone's Online model is not as straightforward when discussing the uniform (or arbitrarily distributed) variations of these models, instead of the adversarial models (cf. [BJT02]). That is, if we consider UROnline and URExact, where URExact is the model in which every counterexample to an equivalence query is chosen randomly uniformly by the teacher, then for the implication UROnline ⇒ URExact to work, we have to require that the UROnline algorithm can learn by receiving only queries on which the current hypothesis fails to predict correctly, i.e. queries that are chosen randomly uniformly among $\{x|\ f(x) \neq h(x)\}$. A simple and natural way to meet this requirement is to assume that the UROnline algorithm does not modify its state after a successful prediction, which effectively means that successful predictions are irrelevant to the learning process. As mentioned in Chapter 6, this assumption is realistic since it typically holds for Online algorithms.

Thus, given a UROnline algorithm **A** that meets the requirement of being suitable for the URExact model, it is simple to use **A** in order to achieve URWOnline learnability. To see this, an algorithm **B** for the URWOnline model will invoke **A** until the first mistake, then make arbitrary predictions until the uniform distribution is reached, then invoke **A** again until the next prediction mistake, and so on.

$$\cdots \underbrace{S\ S\ F} \overbrace{\square\square\cdots\square}^{\approx n\log n} \underbrace{S\ F} \overbrace{\square\square\cdots\square}^{\approx n\log n} \underbrace{S\ S\ S\ F} \overbrace{\square\square\cdots\square}^{\approx n\log n} \underbrace{F} \overbrace{\square\square\cdots\square}^{\approx n\log n} \cdots$$

An illustration of the sequence of (S)uccessful and (F)ailed predictions

Note that **B** can determine when the uniform distribution is reached, by uniformly selecting a random index $1 \leq i \leq n$ whenever $x^{(t)} = x^{(t-1)}$, and keeping track of these indices as well as the indices that are selected by the teacher's flips, until all the indices are selected (cf. Lemma 3).

Since **B** can abstain from invoking **A** on queries that **A** would predict successfully (in case **A** is given as a white box), or simply invoke **A** anyway in case the assumption that **A** does not modify its state after correct predictions hold, it follows that in this simulation all the queries that **A** sees are uniformly distributed.

Thus, the probability that **A** makes more than $poly(size_C(f), n, \frac{1}{\delta'})$ prediction mistakes is less than δ', thereby for $\delta' = \frac{1}{2}$ it follows from linearity of expectation that with probability at least $\frac{1}{2}$ the expected total number of mistakes that **B** makes is bounded by $n(1 + \log n) \cdot poly(size_C(f), n)$.

Since the simulation makes sure that **A** sees only uniformly distributed queries, the event that **A** successfully learned the target function f is independent of the rest of the simulation. Therefore, the probability that both **A** successfully learned, and the total mistakes of **B** is less than twice the expectation, is at least $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

Thus we obtained that the probability that algorithm **B** make more than $2n(1 + \log n) \cdot poly(size_C(f), n) = poly(size_C(f), n)$ mistakes is less than $\frac{3}{4}$, and therefore by $\frac{\log \frac{1}{\delta}}{\log \frac{4}{3}}$ repetitions **B** achieves learnability for an arbitrary δ in the URWOnline model. $\square$

Appendix B

The fastest known algorithm for learning k-juntas under the uniform distribution is presented in [MOS04], and its running time is $\approx n^{.704 \cdot k}$, i.e. an improvement over the trivial $\mathcal{O}(n^k)$ bound, but still superpolynomial even for $k = \omega(1)$. The algorithm in [MOS04] works in the uniform PAC model, but it actually achieves exact learning ($\varepsilon = 0$).

In general, exact learnability under uniform PAC is strictly harder than UROnline learnability (consider e.g. the class of singletons). This could be taken to mean that our proof that the $\mathcal{O}(\log n)$-juntas class is URWOnline learnable only separates the URWOnline model from exact learning under uniform PAC, rather than separating URWOnline from UROnline. However, it is also unknown whether non-exact learning of $\mathcal{O}(\log n)$-juntas is possible in the uniform PAC model, and UROnline learnability indeed implies uniform PAC learnability. Let us justify here this last statement, i.e. UROnline $\Longrightarrow$ uniform PAC, which also helps in elaborating on some of the other implications that we referenced, specifically [HM91] in Section 1.3.

Assume that $\mathbf{A}(\delta)$ is a UROnline algorithm with a polynomial mistake bound p that can learn a certain concept class C, and let us construct an algorithm $\mathbf{B}(\varepsilon, \delta)$ that learns C in the uniform PAC model. $\mathbf{B}(\varepsilon, \delta)$ will continually draw samples $(x^{(t)}, f(x^{(t)}))$ from the teacher, invoke $\mathbf{A}(\frac{\delta}{2})$ on each $x^{(t)}$, and inform $\mathbf{A}(\frac{\delta}{2})$ about prediction mistakes whenever it fails to predict $f(x^{(t)})$ correctly. In case $\mathbf{A}(\frac{\delta}{2})$ makes more than $p(size_C(f), n, \frac{2}{\delta})$ mistakes, or makes $\frac{1}{\varepsilon} \log \frac{2}{\delta}$ consecutive predictions without a mistake, $\mathbf{B}(\varepsilon, \delta)$ halts and outputs the current hypothesis h of $\mathbf{A}(\frac{\delta}{2})$. Notice that the queries that $\mathbf{A}(\frac{\delta}{2})$ sees during this simulation are distributed randomly uniformly.

Let $\mathcal{D}$ be the uniform distribution, let $\mathcal{D}(f,h) \triangleq \Pr_{x\in\mathcal{D}}(f(x) \neq h(x))$, and define the following two events

$$\begin{aligned}\mathcal{A}_1 &\triangleq \{\mathbf{A}(\tfrac{\delta}{2}) \text{ makes more than } p(size_C(f), n, \tfrac{2}{\delta}) \text{ mistakes}\}\\ \mathcal{A}_2 &\triangleq \{\text{the output hypothesis } h \text{ satisfies } \mathcal{D}(f,h) = \varepsilon_1 > \varepsilon\}\end{aligned}$$

Now,

$$\begin{aligned}\Pr(\mathcal{A}_1) &< \frac{\delta}{2}\\ \Pr(\mathcal{A}_2 \mid \overline{\mathcal{A}_1}) &\leq (1-\varepsilon_1)^{\frac{1}{\varepsilon}\log\frac{2}{\delta}} < (1-\varepsilon)^{\frac{1}{\varepsilon}\log\frac{2}{\delta}} \leq e^{-\varepsilon\cdot\frac{1}{\varepsilon}\log\frac{2}{\delta}} = \frac{\delta}{2}\\ \Pr(\{\mathbf{B}(\varepsilon,\delta) \text{ fails}\}) &= \Pr(\{\mathbf{B}(\varepsilon,\delta) \text{ fails}\} \cap \mathcal{A}_1) + \Pr(\{\mathbf{B}(\varepsilon,\delta) \text{ fails}\} \cap \overline{\mathcal{A}_1})\\ &\leq \Pr(\mathcal{A}_1) + \Pr(\{\mathbf{B}(\varepsilon,\delta) \text{ fails}\} \mid \overline{\mathcal{A}_1})\\ &= \Pr(\mathcal{A}_1) + \Pr(\mathcal{A}_2 \mid \overline{\mathcal{A}_1})\\ &< \frac{\delta}{2} + \frac{\delta}{2} = \delta.\end{aligned}$$

The total number of samples that $\mathbf{B}(\varepsilon,\delta)$ receives from the teacher is bounded by $\frac{1}{\varepsilon}\log\frac{2}{\delta}\cdot p(size_C(f), n, \frac{2}{\delta})$, and the probability that $\mathcal{D}(f,h) > \varepsilon$ for the output hypothesis h is smaller than δ. Therefore, $\mathbf{B}(\varepsilon,\delta)$ achieves PAC learning under the uniform distribution. □

Appendix C

For completeness, we give here the simple "reversed" variation of the Markov inequality, that we required in Chapter 6.

Claim. Let X be a real-valued random variable, and let $a_0 \in \mathbb{R}$.
If $X \leq a_0$ always holds, then for $b < a_0$ we have $\Pr(X \leq b) \leq \frac{a_0 - \mathrm{Ex}[X]}{a_0 - b}$.

Proof. Define $Y \triangleq \begin{cases} 0, & x > b \\ \frac{a_0 - b}{b}, & x \leq b \end{cases}$. Note that $Y \leq \frac{a_0 - X}{b}$.

Now, $\mathrm{Ex}[Y] = \frac{a_0 - b}{b} \cdot \Pr(X \leq b)$, and also $\mathrm{Ex}[Y] \leq \mathrm{Ex}[\frac{a_0 - X}{b}] = \frac{a_0 - \mathrm{Ex}[X]}{b}$.
Thus, we obtain $\Pr(X \leq b) \leq \frac{b}{a_0 - b} \cdot \frac{a_0 - \mathrm{Ex}[X]}{b} = \frac{a_0 - \mathrm{Ex}[X]}{a_0 - b}$. □

Bibliography

[A87] D. Angluin. Queries and concept learning. *Machine Learning*, 2, pp. 319-342, 1987.

[A94] A. Blum: Separating Distribution-Free and Mistake-Bound Learning Models over the Boolean Domain. *SIAM J. Comput*, Vol. 23, No. 5, 990-1000 (1994)

[AL91] H. Aizenstein and L. Pitt. Exact learning of read-twice DNF formulas. *Proc. 32th Annu. IEEE Sympos. Found. Comput. Sci.*, IEEE Computer Society Press, pages 170-179, 1991.

[B97] N. H. Bshouty: Simple Learning Algorithms Using Divide and Conquer. *Computational Complexity*, 6(2): 174-194 (1997)

[BFH95] P. L. Bartlett, P. Fischer and K. Höffgen. Exploiting Random Walks for Learning. *Information and Computation*, 176: 121-135 (2002).

[BJT02] N. H. Bshouty, J. C. Jackson and C. Tamon. Exploring Learnability between Exact and PAC. *15th Annual Conference on Computational Learning Theory, Sydney, Australia, July 8-10, Proceedings* (COLT 2002).

[BMOS03] N. H. Bshouty, E. Mossel, R. O'Donnell and R. A. Servedio. Learning DNF from Random Walks. FOCS 2003: 189-

[D88] P. Diaconis. Lecture notes: group representations in probability and statistics. *Institute of Mathematical Statistics*, volume 11 (1988).

[DS87] P. Diaconis and M. Shahshahani. Time to reach stationarity in the Bernoulli–Laplace diffusion model. *SIAM Journal on Mathematical Analysis*, 18: 208–218 (1987).

[ELSW07] A. Elbaz, H. K. Lee, R. A. Servedio and A. Wan. Separating Models of Learning from Correlated and Uncorrelated Data. *Journal of Machine Learning Research*, 8, pp. 277-290 (2007).

[FS92] P. Ficher and H. Simon. On learning ring-sum expansions. *SIAM J. Comput.* 21: 181–192, 1992.

[HM91] T. Hancock and Y. Mansour. Learning Monotone $k\mu$ DNF Formulas on Product Distributions. *Proc. 4th Ann. Workshop on Comp. Learning Theory* (1991), 179-183.

[KLPV87] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the Learnability of Boolean Formulae. In Proceedings of the 19th ACM Symposium on the Theory of Computing, 285-195, 1987.

[KS01] A. R. Klivans and R. A. Servedio. Learning DNF in time $2^{\tilde{\mathcal{O}}(n^{1/3})}$. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 258-265, 2001.

[LPW09] D. A. Levin, Y. Peres, and E. L. Wilmer: Markov chains and mixing times. *American Mathematical Society*, Providence, RI (2009).

[L87] N. Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. *Machine Learning*, **2**, No. 4, 285–318, 1987.

[MOS04] E. Mossel, R. O'Donnell and R. A. Servedio. Learning juntas. STOC 2003: 206-212. Learning functions of k relevant variables. *Journal of Computer and System Sciences* 69(3), 2004, pp. 421-434

[NSS95] M. Naor, L. J. Schulman and A. Srinivasan. Splitters and Near-Optimal Derandomization. *36th Annual Symposium on Foundations of Computer Science*, pages 182-193 (1995).

[PW90] L. Pitt and M. K. Warmuth. Prediction-preserving reducibility. *Journal of Computer and System Science*, 41(3), pp. 430–467, (1990).

[V90] K. Verbeurgt: Learning DNF Under the Uniform Distribution in Quasi-Polynomial Time. *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 314-326 (1990).

על למידה מדויקת על-ידי הילוך אקראי

עידו בן-טוב

על למידה מדויקת על-ידי הילוך אקראי

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

עידו בן-טוב

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
טבת ה'תש"ע חיפה דצמבר 2009

המחקר נעשה בהנחיית פרופ' נאדר בשותי בפקולטה למדעי המחשב.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

תקציר

למידה חישובית היא תחום תאורטי במדעי המחשב אשר ניתן לתארו באופן כל-לי כתחום בו יש לפתח אלגוריתם למידה יעיל שצריך לגלות מושג מסוים מתוך דוגמאות הניתנות מהסביבה, או באופן שקול ממורה אשר יודע מראש את המושג אותו צריך ללמוד.

ישנן דרכים שונות להגדיר מודל למידה פורמלי, כאשר חלקן מאפשרות ללומד להשיג הצלחה רבה יותר בפיתוח אלגורתמי למידה יעילים לעומת מודלים נוקשים יותר. לדוגמא, מודלים רבים מקלים את הדרישה שאלגוריתם הלמידה צריך לגלות במדויק את המושג הידוע למורה, ובמקום זאת דורשים שהמרחק הסטטיסטי בין המושג שנלמד לבין המושג שבידי המורה יהיה קטן. תזה זו מתמקדת בלמידה מדויקת של פונקציות בוליאניות במודלים המבוססים על מודל מפורסם הקרוי אונליין (Online), ודנה בשאלה האם הגבלות של מודל זה בהן המורה מייצר את הדוגמאות על-ידי תהליך סטוכסטי של הילוך אקראי מקלות על הלומד. מודל האונליין הכללי, בו המורה בוחר את הדוגמאות כרצונו, מהווה מודל למידה מה-נוקשים ביותר, אשר שקול למודל מפורסם נוסף הקרוי Exact, וכן ניתן להוכיח תחת הנחות קריפטוגרפיות חלשות שלמידה במודל האונליין היא ממש קשה יותר מלמידה במודל ה-PAC, שהוא המודל המפורסם ביותר ללמידה לא-מדויקת.

השאלות הפתוחות החשובות בתחום של למידה חישובית מתרכזות בלמידה של מחלקות מוכרות של פונקציות, כגון נוסחאות DNF הניתנות לייצוג באורך פולינומי במספר המשתנים. עבור מודל האונליין, ניתן להוכיח באמצעות למת הרכבה של-מידה של תת-מחלקות מסוימות של DNF, כגון ROM-DNF שהיא המחלקה של נוסחאות DNF מונוטוניות בהן כל משתנה מופיע לכל היותר פעם אחת, היא קשה באותה מידה כמו למידת DNF במקרה הכללי. במצב הידע הנוכחי בתחום, לא ידוע האם למידה במודל אונליין בו הדוגמאות נבחרות בהתפלגות אחידה היא ישיגה גם עבור תת-מחלקות של DNF כמו חונטות בסדר גודל לוגריתמי במספר המשתנים, כלומר אפילו תחת הנחות מקלות לגבי מודל הלמידה ומחלקת הפונקציות אותה

יש ללמוד, השאלה האם ניתן להשיג למידה יעילה נותרת כשאלה פתוחה.

במקום לבחון את המקרה של מורה הסתברותי אשר בוחר דוגמאות באופן אחיד, כאן אנו בוחנים מורה הסתברותי אשר בוחר דוגמאות על-פי תהליך סטוכסטי של הילוך אקראי, כלומר תהליך בו דוגמאות עוקבות נבדלות בערכו של משתנה יחיד. מודל זה הוא בעל חשיבות תאורטית מכיוון שתחת הנחות קריפטוגרפיות מקובלות זהו מודל בו הלמידה היא ממש קשה יותר מהמודל בו ללומד מותר לבצע שאילתות אקטיביות כדי לדעת את הערך של דוגמאות לפי בחירתו, אך זהו עדיין מודל פסיבי, כלומר מודל בו המורה בוחר את הדוגמאות, אשר הלמידה בו ממש קלה יותר מהמודל הפסיבי בו המורה בוחר את הדוגמאות בהתפלגות אחידה. בנוסף, למידה על-ידי הילוך אקראי היא גם בעלת חשיבות מעשית, שכן ההנחה במודלים כגון אונליין אחיד או PAC לגבי אי-תלות בין דוגמאות עוקבות איננה תמיד מתקיימות בפועל, ופרט עבור למידה של תהליכים פיסיים, כגון מסלולים של רובוטים, דוגמאות עוקבות נוטות להשתנות רק במעט, ולכן מודלים של הילוך אקראי יכולים להתאים יותר במקרים האלה.

ראשית אנו מוכיחים שמודל האונליין הכללי שקול למודל האונליין עם מורה עוין אשר בוחר את הדוגמאות כרצונו תחת ההגבלה שהדוגמאות חייבות להוות הילוך אקראי, כלומר המורה חייב לבחור דוגמאות עוקבות הנבדלות במשתנה יחיד. ההוכחה מסתמכת על תכונה טבעית שמתקיימת במחלקות המוכרות של פונקציות בוליאניות, לפיה ניתן למצוא פונקציה במחלקה השקולה לפונקציה נתונה עבור ערך של משתנה מסוים נוסף, ושקולה לפונקציה קבועה כאשר מציבים את הערך ההפוך עבור המשתנה המסוים הזה. מכאן אנו מראים כיצד ניתן לבצע סימולציה שממירה אלגוריתם עבור מודל האונליין עם הילוך אקראי ומורה עוין לאלגוריתם עבור מודל האונליין הכללי, תוך שימוש באיפוס וקביעה מחדש של המשתנה המסוים הנוסף לצורך הדמיה של הילוך אקראי עבור האלגוריתם הנתון.

מכיוון שלמידה עם הילוך אקראי ומורה עוין שקולה ללמידה במקרה הכללי, אנו מתמקדים כעת במודל האונליין עם הילוך אקראי ומורה הסתברותי, כלומר עבור כל דוגמא המורה צריך להפוך את ערכו משתנה אחד שנבחר בהסתברות אחידה מבין כל המשתנים. אנו מראים תחילה שקל ללמוד ביעילות חונטות בסדר גודל לוגריתמי במודל זה, בניגוד למודלים שהוזכרו לעיל. תוצאה זאת נובעת מכך שהתהליך הסטוכסטי של הילוך אקראי מתערבל במהירות אל ההתפלגות האחידה, וניתן לזהות משתנה רלוונטי כאשר ערכו מתהפך עבור דוגמא בה הוא משפיע על פלט החונטה. בנוסף אנו מראים כיצד ניתן להרחיב את אלגוריתם הלמידה למקרה בו מספר המשתנים הרלוונטים איננו ידוע מראש, וכן כיצד ניתן לחסום את סיבוכיות הלמידה בצורה משופרת תחת הנחות נוספות לגבי הרגישות של

הפונקציות במחלקה.

התוצאה המרכזית בעבודה זו היא הוכחה שהמחלקה RO-DNF, כלומר נוסחאות DNF בהן כל משתנה מופיע לכל היותר פעם אחת, ניתנת ללמידה יעילה במודל האונליין עם הילוך אקראי ומורה הסתברותי. תוצאה זו מפרידה בין מודל הלמידה על-ידי הילוך אקראי לבין המודלים אונליין ו-PAC, תחת ההשערה המקובלת בתחום של למידה חישובית לפיה לא ניתן ללמוד DNF עם ייצוג פולינומי במודלים הללו. אלגוריתם הלמידה איתו אנו לומדים ROM-DNF ולאחר מכן RO-DNF הוא ישיר למדי: מבנה הנתונים ההתחלתי מניח שהטרם בו מופיע כל משתנה מכיל גם את כל המשתנים האחרים, ובכל שאילתה אנו מנחשים את ערך הפונקציה כך שאם נטעה אז נוכל למחוק חלק מהמשתנים שהנחנו שמופיעים בטרמים. אם בכל טעות היינו מצליחים למחוק משתנים אז היינו מקבלים בכך למידה יעילה אפילו אם המורה היה עוין, מה שהיה גורר למידת DNF כללית על-פי הוכחת השקילות בין למידה עם הילוך אקראי ומורה עוין לבין אונליין כללי. ואכן, אלגוריתם הלמידה שלנו עלול להיתקל במקרים בו הוא מבצע טעויות בלי לקבל מידע חדש, ובמקרים אלה אנחנו מחליפים לסרוגין בין שני מצבים כדי לבחור באיזה ניחוש להשתמש כך שלא נבצע יותר מידי טעויות ללא תלות בפונקציה הנלמדת. עיקר הקושי הוא בהוכחה שסיבוכיות הלמידה של האלגוריתם היא פולינומית, שנובעת מכך שאנו חוסמים את היחס בין הדוגמאות שעלולות לגרום לטעות שלא מוסיפה מידע חדש לבין הדוגמאות שעלולות לגרום לטעות שכן מוסיפה מידע חדש בכל מצב ביניים אפשרי של האלגוריתם תוך כדי תהליך הלמידה.

בכך קיבלנו עדויות חזקות לכך שלמידה מדויקת במודל האונליין עם הילוך אקראי ומורה הסתברותי היא ממש קלה יותר מלמידת אונליין מדויקת כללית או למידת PAC כללית. לסיום אנו מוכיחים, תחת הנחות סבירות, שמודל האונליין עם הילוך אקראי ומורה הסתברותי איננו קל מידי, כלומר שלא כל מחלקה מוכרת של פונקציות בוליאניות ניתנת ללמידה במודל הזה. למעשה, אנו מוכיחים שלמידה יעילה במודל הזה של המחלקה read-3 DNF, כלומר של נוסחאות DNF בהן כל משתנה מופיע לכל היותר שלוש פעמים, גוררת למידה יעילה של DNF כללי עם ייצוג פולינומי במודל האונליין עם מורה אשר בוחר את הדוגמאות בהתפלגות אחידה, מה שבפרט גורר למידה יעילה של DNF כללי במודל ה-PAC תחת ההתפלגות האחידה, שהיא שאלה פתוחה חשובה. ההוכחה מסתמכת על סימולציה בה הדוגמאות שניתנות בהתפלגות אחידה מומרות בדוגמאות מורחבות בהן כל משתנה משוכפל מספר פעמים, וכדי לטעון שההתפלגות שמתקבלת תהיה בהסתברות גבוהה זהה להתפלגות שהיתה מתקבלת מהתהליך הסטוכסטי של הילוך אקראי כאשר מתבוננים רק בדוגמאות בהן למשתנים המשוכפלים יש את אותו ערך, אנו מסתמכים על כך שזמן הפגיעה בין שני צמתים בקוביה הבוליאנית הוא אקספוננציאלי במ-

ספר המשתנים.