Learning Binary Relations, Total Orders, and Read-Once Formulas

by

Sally Ann Goldman

S.M. EECS, Massachusetts Institute of Technology (1987)
Sc.B. Computer Science, Brown University (1984)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1990

Signature of Author Sally A. Goldman
Department of Electrical Engineering and Computer Science
July 16, 1990

Certified by Ronald L. Rivest
Ronald L. Rivest
Professor of Computer Science
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Abstract

We study several learning problems under various learning models. We first consider an on-line model in which the learner answers a sequence of yes/no questions with immediate feedback provided after each question. The learner strives to discover the correct classification rule while making as few mistakes as possible. The complexity of the learning task depends on the agent selecting the sequence of questions. Typically, it is assumed that an adversary selects this sequence. We present an extended mistake-bound model in which the sequence of questions is selected by a helpful teacher, by the learner, by an adversary, or at random.

Using this extended mistake-bound model, we study the problem of learning a relation between two sets of objects. If the relation has no structure, the learner cannot possibly make good predictions. We impose structure by restricting one set of objects to have relatively few "types". We describe efficient algorithms to learn a binary relation for the various selection methods. Complementing these results are lower bounds, often proving that the algorithms perform optimally.

Next we consider the problem of learning a total order on a set of elements. That is, we restrict the predicate of the relation to be a total order. Again both upper and lower bounds are provided for the different selection methods. Furthermore, we uncover an interesting relationship between learning theory and randomized approximation schemes.

Other questions arise from this generalized learning model. In the case that a teacher selects the sequence of questions, we consider the number of mistakes made by any learner that predicts according to a rule that agrees with all previous examples. Equivalently we ask: what is the minimum number of examples a teacher must reveal to uniquely identify the target concept? It is an interesting paradox that for many concept classes, the number of mistakes made with a helpful teacher may be worse than the number of mistakes made when the learner selects the sequence. In the case that the learner chooses the sequence of questions, we show that the number of mistakes can be significantly smaller than the number of queries needed.

Finally, we present a new technique for *exactly identifying* read-once formulas from random examples. The method is based on sampling the input-output behavior of the target formula on a probability distribution which is determined by the *fixed point* of the formula's *amplification function* (defined as the probability that a 1 is output by the formula when each input bit is 1 independently with probability p). We present efficient algorithms for exactly identifying families of read-once formulas (where each input appears at most once) over various bases. These include formulas of majority gates and a large subclass of formulas over the standard basis. We then apply these results to prove the existence of polynomial-length *universal identification sequences* for large classes of formulas.

Keywords: Machine Learning, Computational Learning Theory, Mistake-bounded Learning, On-line Learning, Binary Relations, Total Orders, Circuits, Read-once Formulas, Fully Polynomial Randomized Approximation Schemes, Amplification Functions, Teaching Dimension

Thesis supervisor: Ronald L. Rivest.
Title: Professor of Computer Science.

Acknowledgments

First and foremost, I would like to thank my thesis advisor, Ron Rivest, for all he has done for me during my graduate career. He's been a pleasure to work with and has done everything one could ask of an advisor. No matter how tight his schedule, he made time to listen to my ideas and to redirect me when I was at a dead end. Furthermore, when I gave him material to read, not only did he carefully check every technical aspect of the paper, but he also helped me to improve my writing style.

I would like to thank Mike Kearns and Rob Schapire for making the "learning ghetto" such an enjoyable and productive office. I've truly enjoyed working with them and have learned a lot from them. Portions of my thesis are joint with them and I thank them for allowing me to include this work here.

I thank Tom Cormen, Bruce Maggs, and Cliff Stein for being such good officemates during the three years we spent together. I'd especially like to thank Tom Cormen for his expert guidance with LaTeX and MacDraw.

The results in Chapter 4 were inspired by an "open problems session" led by Manfred Warmuth at Ron's weekly Machine Learning Reading Group meeting, where he proposed the basic idea of using an approximate halving algorithm based on approximate and probabilistic counting, and also suggested the problem of learning a total order. Also, the results described in the appendix are entirely due to Manfred. I thank Manfred for generously sharing his ideas.

Many thanks to Bill Aiello, Avrim Blum, Ken Goldman, Michelangelo Gringi, Tom Leighton, Nick Littlestone, Cindy Phillips, Bob Sloan, and Peter Winkler for many interesting and enlightening discussions. I thank Paris Kanellakis for introducing me to research and encouraging me to attend graduate school at MIT. Thanks to everyone in the Theory of Computation group for making MIT such an exciting place to spend my graduate career.

I'd also like to thank the members of my orals committee (Silvio Micali and Marvin Minsky), area exam committee (Alok Aggarwal, Peter Elias, David Shmoys, and Èva

Tardos), and thesis readers (Bob Berwick and Shafi Goldwasser) for all they have done for me.

Financial support was provided by an ONR Graduate Fellowship, by NSF grants DCR-8607494 and CCR-8914428, and by a grant from the Siemens Corporation.

I'd like to thank my siblings, grandparents and in-laws for being there when I needed them. I can't begin to repay my parents, Marilyn and Bob Goldwasser, for all that they have done for me during the last 27 years. I thank Mark for being such a wonderful son and allowing me plenty of time to work. Finally, my greatest thanks goes to my husband, Ken. He's done everything from reading drafts of papers and discussing research ideas to consoling me when I was down. And when things got tight I could always count on him to spend extra time with Mark. Without all his love, support and encouragement I could have never made it this far. To him I dedicate this thesis.

Contents

Chapter 1

Introduction

Our ultimate objective is to make programs that learn from their experience as effectively as humans do.
— John McCarthy, 1958 [48]

Building machines that learn from experience is an important research goal of artificial intelligence. The area of *computational learning theory* strives to define formal mathematical models of machine learning that enable rigorous analysis of the performance of learning algorithms. We are interested in designing learning algorithms that are efficient in their use of both time and data. Thus, our goal is to determine under what conditions a given learning problem is computationally feasible.

1.1 A Research Methodology

How can one apply the formalism of computational learning theory to a "real-life" learning problem? We suggest the following general framework.

1. Precisely describe the problem. Here, it is important to preserve the key features of the problem while simplifying the problem enough to be analyzed. Insights gained from studying the simplified problem may then be used to cope with the more general case.

2. Select an appropriate formal learning model. In selecting the learning model, some key questions to address are [55, 62]:

   - What is being learned?
   - How does the learner interact with the world?
   - What is the prior knowledge of the learner?
   - How is the learner's hypothesis represented?
   - What are the criteria for successful learning?

3. Design efficient learning algorithms. Typically, these algorithms learn from examples, and we are concerned with both the time to process each example and the total number of examples used.

4. Use the success criteria from the formal model to judge the performance of the learning algorithms.

If one proves no efficient algorithm can meet the success criteria, then one may want to further simplify the problem or modify the learning model. There are several options for modifying the learning model. One possibility is to enhance the learner's interaction with the environment; perhaps by allowing the learner to take a more active role, or by providing a teacher. One could provide the learner with additional prior knowledge, or allow a richer hypothesis class. Finally, the success criteria could be relaxed. Often several iterations of this process are needed to best model the given problem.

1.2 Thesis Overview

Most work in the area of computational learning theory has been focused on the problem of *concept learning*. For these learning problems there are a set of *instances* (or objects) and a single *target concept* that classifies each instance as a positive or negative instance. Let the *instance space* denote the set of all instances that the learner may see. The

learner's goal is to properly perform this classification task for each instance in the instance space. For example, if the instance space contains office furniture, the learner's task may be to find a rule that accurately distinguishes a chair from all other types of office furniture.

In many "real-life" learning problems, it is important to acquire information about a predicate relating elements of two sets. For example, one may wish to learn a "has-part" predicate relating a set of animals to a set of attributes. Observe that this problem of learning a binary relation can be viewed as a concept learning problem by letting the instances be all ordered pairs of objects from the two sets. However, these are fundamentally different problems. In concept learning there is a single set of objects and the learner's task is to classify these objects, whereas in learning a binary relation there are two sets of objects and the learner's task is to learn the predicate relating the two sets. Furthermore, the ways in which the problem may be structured are quite different when the true task is to learn a binary relation as opposed to a classification rule.

In Chapters 3 and 4 we consider the problem of learning binary relations. We represent a binary relation between two sets as an $n \times m$ matrix, where the row and column indices name the elements of the two sets. Each entry of the matrix gives the value of the predicate relating the corresponding pair of set elements. We study how the learner's performance depends on the order in which the learner must classify the objects. By considering when the presentation order is selected by the learner, by a helpful teacher, by an adversary, or at random.

If the learner is to have any hope of doing better than random guessing, there must be some structure in the relation. Furthermore, since there are so many ways to structure a binary relation, we give the learner prior knowledge about the nature of this structure. Not surprisingly, the learning task depends greatly on the prior knowledge provided. One way to impose structure is to restrict one set of objects to have relatively few "types." For example, a circus may contain many animals, but only a few different species. In Chapter 3 we study the case where the learner has "a priori" knowledge that there are a

limited number of object types. Namely, we restrict the matrix representing the relation to have at most k distinct row types. For each of the presentation orders mentioned above, we describe efficient learning algorithms for this problem. Complementing these results are information-theoretic lower bounds, often proving that the algorithms perform optimally.

In Chapter 4 we study the problem of learning a binary relation on a set where the predicate induces a total order on the set. That is, the learner has a priori knowledge that the relation forms a total order. For this problem we observe that the *halving algorithm* [6, 43] yields a good mistake bound against any query sequence. (The halving algorithm predicts according to the majority of the feasible concepts, and thus each mistake halves the number of concepts to consider.) This motivates a second goal of Chapter 4—developing efficient implementations of the halving algorithm. In doing so, we uncover an interesting application of randomized approximation schemes to computational learning theory. Namely, we use a randomized approximation scheme for counting the number of extensions of a partial order [16, 47] to build an algorithm for learning a total order under an adversary-selected query sequence. We contrast this result with an $n-1$ mistake bound when the learner or teacher selects the query sequence. Finally, we discuss how the halving algorithm may be used to construct efficient counting algorithms.

In studying the problem of learning a binary relation, we see that the learner's success depends greatly on the interaction between the learner and its environment. In particular, the complexity of the learner's task depends on the way in which the instances are selected. In Chapter 5, we study the relationship between the presentation order of the instances and the number of mistakes made by the learner in the domain of concept learning. When a teacher selects the presentation order, we consider the maximum number of mistakes made by any learner that predicts according to some concept that agrees with all previously seen examples. Thus we ask: what is the minimum number of examples a teacher must reveal to uniquely identify the target concept? We refer to this measure of the complexity of teaching any concept from the class as the *teaching*

dimension. We feel that this is a good model of a teacher since it corresponds to the best way to present the lesson so that any student who is "paying attention" will learn the lesson. We compare the teaching dimension to other well-studied measures of the complexity of a concept class. We compute upper and lower bounds for the teaching dimension of several concept classes and then present some more general results. Next we address the related question of how many mistakes the learner must make in the worst case when selecting the presentation order for the examples himself. We bound this quantity for several concept classes, showing that the number of mistakes can by asymptotically smaller than the number of queries needed.

Another important consideration in concept learning is the tradeoff between the criteria for successful learning and the regularity of the environment. In the ***distribution-free*** or *PAC* model introduced by Valiant [66], the learner receives random examples from an arbitrary distribution and must learn a rule that almost always gives a good approximation to the given concept. While some simple concept classes are efficiently learnable under these criteria, other classes have been shown not to be efficiently learnable*. While one may choose to provide the learner with additional prior knowledge or enhance the learner's interaction with its environment, another possibility is to place more structure on the environment. That is, a part of the distribution-free model that makes the learning task significantly more difficult is the requirement of learning under ***any*** distribution. In Chapter 6, we consider the effect of placing more structure on the environment by restricting the distribution from which the examples are drawn. We describe a new technique for *exactly identifying* certain classes of read-once formulas from random examples. (Furthermore, we prove that these classes of formulas are not efficiently learnable in the distribution-free model.) Although we fix the distribution, we are now able to learn a hypothesis that is equivalent to the target concept instead of learning just a good approximation to the target concept. The method is based on the observation of the input-output behavior of the target formula on a fixed probability distribution which

*Under complexity-theoretic or cryptographic assumptions.

is determined by the *fixed point* of the formula's *amplification function* (defined as the probability that a 1 is output when each input is 1 independently with probability p). By demonstrating that the formula's behavior is *unstable* in an appropriate sense on this distribution, we are able to infer all structural information about the formula (with high probability) by performing various statistical tests on easily sampled variants of the fixed distribution. Since we give positive results for concept classes that are not learnable in the distribution-free model (and thus by a result of Schapire [59] the learner can essentially perform no better than just randomly guessing), these results may be interpreted as demonstrating that while there are some distributions that, in a computationally bounded setting, reveal essentially *no* information about the target formula, there are natural distributions which reveal *all* information.

This thesis is organized as follows. In the next chapter we review existing learning models, and then present a new learning model. In Chapter 3, we study the problem of learning a binary relation when the learner has prior knowledge that there are a limited number of object types. In Chapter 4, we study the related problem in which the learner receives prior knowledge that the predicate forms a total order. Next, in Chapter 5, we study the relationship between the learner's interaction with the environment and the number of incorrect classifications made in the domain of concept learning. In Chapter 6, we present our algorithms for learning read-once formulas from random examples. Finally, in Chapter 7, we close with some concluding remarks.

Chapter 2

Formal Learning Models

The second step of the research methodology outlined in Chapter 1 is to select a formal learning model. Most learning models can be obtained by various combinations of the following two orthogonal features: the learner's interaction with the environment and the criteria for success. We begin by discussing some possibilities for these two features, and then review the most commonly used learning models. Finally, we discuss some interesting combination of these features that have not yet been considered.

Before describing these features, we describe the basic framework used in all learning models discussed here. For ease of exposition, we discuss these models in terms of concept learning. As we mentioned in Chapter 1, the problem of learning a binary relation can be viewed in this framework by letting the instances be all ordered pairs of objects from the two sets.

A *concept* c is a Boolean function on some domain of instances. A *concept class* C is a family of concepts. The learner's goal is to infer some unknown target concept chosen from some *known* concept class. That is, the learner knows *a priori* the concept class from which the target is chosen. Often C is decomposed into subclasses C_n according to some natural dimension measure n. That is, for each $n \geq 1$, let X_n denote a *learning domain*. Let $X = \bigcup_{n \geq 1} X_n$ denote the *instance space*, and $x \in X$ denote an *instance*. To illustrate these definitions, we consider the concept class of monomials. (A monomial is a

conjunction of literals, where each literal is either some Boolean variable or its negation.) For this concept class n is just the number of variables. Thus $X_n = \{0,1\}^n$, $|X_n| = 2^n$, and each $x \in X_n$ represents an assignment of 0 or 1 to each variable. For each $n \geq 1$, let $C_n \subseteq 2^{X_n}$ be a *family of concepts* over X_n. Let $C = \bigcup_{n \geq 1} C_n$ denote a *concept class* over X. For example if C_n contains monomials over n variables, then C is the class of all monomials. For $c \in C_n$ and $x \in X_n$, $c(x)$ denotes the outcome of evaluating c on input x. Given any concept $c \in C_n$, we say that x is a *positive instance* of c if $c(x) = 1$, and x is a *negative instance* of c if $c(x) = 0$. In our example, the target concept for the class of monomials over five variables might be $x_1\overline{x_4}x_5$. Then the instance "10001" is a positive instance and "00001" is a negative instance. Finally, the *hypothesis space* of algorithm A is simply the set of all hypotheses (or rules) h that A may output. (A hypothesis for C_n must make a prediction for each $x \in X_n$.)

As we briefly discuss later, the set of hypotheses which the learner may use is a significant factor. While this is an important issue, our results do not address it. Throughout this thesis we only require that the classification for each instance can be computed from the hypothesis h in time polynomial in n. Now that we have set up the basic framework, we are ready to discuss the two key features of a learning model: the learner's interaction with the environment and the criteria for successful learning.

2.1 Interaction with the Environment

In all learning models we consider, the environment presents the learner with instances from the instance space X_n. An instance may be either labeled or unlabeled. A *labeled instance* is an element x chosen from X_n along with a bit specifying whether x is a positive or negative instance of the target concept. An *unlabeled instance* is an element x chosen from X_n that is provided without a classification.

One important consideration is the way in which unlabeled and labeled instances are combined in the learning session. In the *batch model* the learner first receives labeled

instances that enable him to form a hypothesis for the target concept and then this hypothesis is used to classify unlabeled instances. In the *on-line model* the learner makes predictions for unlabeled instances from the start, modifying his hypothesis based on feedback provided by the environment. We now describe these two scenarios in more detail, and then consider another key feature of the learning model: the agent selecting the instances.

2.1.1 Batch Versus On-Line Learning

In this section we describe the two typical uses of labeled and unlabeled examples: the batch model and the on-line model.

Batch Learning

To motivate the batch learning model, we use the following example of Littlestone [42]. Suppose a state passes a law requiring that a deposit be placed on all mineral water bottles originally purchased in the state. The deposit is then refunded when the consumer returns the bottle to a recycling center. The mineral water bottles are specially marked to distinguish them from other kind of bottles. A company wants to build an automatic bottle receiving machine that will check returned bottles for the special mark and reject them if it is not found. The company decides to build a prototype machine that is capable of learning so that this machine can be trained to classify a given bottle as a mineral water bottle or some other type of bottle. For positive training examples, the company will give its employees bottles of mineral water to take home, drink, and return. To generate negative examples the employees will give the machine other kinds of bottles designed to mimic the mineral water bottles. For each bottle seen during the training period, the machine will be correctly told whether or not it is one of the specially marked bottles. After training the prototype, the company will copy the final classification rule produced by the training process and use it for the machines to be placed in the recycling centers.

A key property of the batch model is that there is a separation between the *training phase* and the *performance phase*. Formally, a batch learning algorithm for C is an algorithm that runs under the following scenario. A learning session consists of two phases: the training phase and the performance phase. In the training phase the learner is presented with a set of instances labeled according to the target concept $c \in C_n$. At the end of this phase the learner must output a hypothesis h that classifies each $x \in X_n$ as either a positive or negative instance. Then in the performance phase, the learner uses h to predict the classification of new unlabeled instances. Since the learner never finds out the true classification for the unlabeled instances, all learning occurs in the training phase.

On-Line Learning

We now give an example to motivate the on-line learning model. Suppose that when arriving at work (in Boston) you may either park in the street or park in a garage. In fact, between your office building and the garage there is a street on which you can always find a spot. On most days, street parking is preferable since you avoid paying the \$10 garage fee. Unfortunately, when parking on the street you risk being towed (\$50) due to street cleaning, snow emergency, special events, etc. When calling the city to find out when they tow, you are unable to get any reasonable guidance and decide the best thing to do is just learn from experience. There are many pieces of information that you might consider in making your prediction; e.g. the date, the day of the week, the weather. We make the following two assumptions: enough information is available to make good predictions if you know how to use it, and after you commit yourself to one choice or the other you learn of the right decision. In this example, the city has rules dictating when they tow; you just don't know them. If you park on the street at the end of the day you know if your car was towed; otherwise when walking to the garage you see if the street is clear (i.e. you learn if you would have been towed).

The on-line model is designed to study algorithms for learning to make accurate

predictions in circumstances such as these. Formally, an on-line learning algorithm for C is an algorithm that runs under the following scenario. A *learning session* consists of a set of *trials*. In each trial, the learner is given an unlabeled instance $x \in X_n$. The learner uses its current hypothesis to predict if x is a positive or negative instance of the target concept $c \in C_n$ and then the learner is told the correct classification of x. If the prediction was incorrect, the learner has made a *mistake*. Note that in this model there is no training phase. Instead, the learner receives *unlabeled instances* throughout the entire learning session. However, after each prediction the learner "discovers" the correct classification. This feedback can then be used by the learner to improve his hypothesis. A learner is *consistent* if, on every trial, there is some concept in C_n that agrees with the learner's prediction, as well as with all the labeled instances observed on the preceding trials.

2.1.2 Selection of the Instances

Another important factor of the learner's interaction with the environment is the method of instance selection. Observe that the method used to select the instances is independent of whether or not the instance is to be labeled; once the instance is selected, the label is just determined by the target concept.

Not surprisingly, some sequences of instances may guide the learner to a good hypothesis faster than other sequences. The most appropriate choice for the method of instance selection depends on the learning problem being modeled. It could be that instances are presented according to some natural distribution on the instance space. Another possibility is that an adversary chooses the instances to make the learning task as hard as possible. Or perhaps the learner is performing active experimentation and can select his own instances. Finally, it may be that a teacher is helping to guide the learner by selecting the "most informative" instance sequence. We now describe each of these possibilities in more detail.

Stochastic Selection

The first setting we discuss for the selection of the instances is *stochastic selection* in which the instance are drawn randomly according to some distribution D over the instance space. A primary motivation for this setting is to simulate what happens in Nature. Suppose the learner is sent on an African safari to build a rule to distinguish an elephant from other jungle animals. Whenever the learner sees an animal the guide tells him whether that animal is an elephant. There is some natural distribution on the animals that the learner will see on the safari. Since this distribution may be arbitrarily complex, it is desirable to model D as an *arbitrary* unknown distribution. We let *distribution-free stochastic selection* denote the case in which the instances are selected at random according to a fixed, unknown, arbitrary distribution.

In other cases, one may want to consider when the instances are randomly selected according to some *particular distribution*. For example, the distribution may be uniform on the instance space. Clearly anything learnable in the distribution-free setting will be learnable in this setting. In fact, one reason for fixing the distribution is to get positive results for classes where learning in the distribution-free setting is known to be infeasible. Or, in some cases, it simply might be that the naturally occurring distribution of the instances is known to be some particular distribution.

Adversary Selection

While the model of stochastic selection is appropriate for many situations, sometimes it is necessary to model the worst possible interaction that may occur. For example, in the parking example given to motivate the on-line model, it may be that the learner wants to know the maximum number of wrong decisions he could possibly make before knowing how to use the available information. Clearly the learner has no control over when the city will tow. Although one could assume that each combination of features occurs according to some probability distribution, this model will not answer our question about the worst-case number of mistakes made by the learner—it could be that the "hardest"

presentation order is not likely.

Thus, one often wants to let an "all-powerful" adversary select both the target concept and the presentation order for the instances. By all-powerful we mean that the adversary has unlimited computation time*, it knows the learner's algorithm, and it has access to any random bits used by the algorithm. Note that the adversary can create the target concept as the learning session proceeds. All we require of the adversary is that at the end of the learning session all instances have been labeled consistently with *some* target concept. Observe that this model is very much like that used in standard algorithm analysis for determining worst-case running time.

Learner Selection

In the two models discussed above, the learner's role is completely passive. Sometimes, one wants the learner to play a more active role by experimenting to determine the unknown target concept. We model this experimentation by allowing the learner to select the instances, where each instance must be selected in time polynomial in n. An instance selected by the learner is referred to as a *membership query*. That is, a membership query is a call to an oracle that on input x for any $x \in X_n$ classifies x as either a positive or negative instance according to the target concept $c \in C_n$.

Teacher Selection

Finally, in some cases one wants to model the situation in which a teacher helps guide the learner through a careful selection of the instances. For example in teaching subtraction, the teacher might first give the instances: "$3 - 2 = 1$" and "$11 - 9 = 2$" before giving "$172552 - 2374 = 170178$". That is, a teacher may first teach the learner the "easy" instances and then teach the harder ones.

A type of teacher that has been considered [2, 4] is one that knows the hypothesis h of the learner and always presents a labeled instance which is misclassified by h. Such a

*In fact the adversary may use uncomputable functions.

teacher is formalized by allowing the learner to make *equivalence queries*. An equivalence query is a call to an oracle that on input $c' \in C_n$ either replies that the target concept c is equivalent to c' or provides an instance $x \in X_n$ such that c and c' classify x differently. That is, the oracle either says that the conjectured concept is correct or provides a counterexample to it. Observe that this oracle is not a very "friendly" teacher in that the counterexample may provide as little information as possible. One could also consider such a scenario where the teacher gives the most informative counterexample. We require that the learner selects each equivalence query in time polynomial in n.

2.2 Success Criteria

Having discussed the learner's interaction with the environment, we now turn to the other key feature of the learning model: the criteria the learner must satisfy to be successful. Each of the criteria we discuss places requirements on the following three components: the accuracy of the hypothesis, the sample complexity of the learner's algorithm and the time complexity of the learner's algorithm. The difference between these criteria is the accuracy required of the learner's hypothesis. Before describing this first component in detail, we discuss the other two components. The *sample complexity* of a learning algorithm is the number of instances required by the learner in order to output the final hypothesis. For all criteria we require that when learning a target concept chosen from C_n the sample complexity is polynomial in n. The *time complexity* of a learning algorithm is the total time required by the learner in order to output the final hypothesis. (We assume that each example is drawn in unit time.) For all criteria we also require that the time complexity of the learning algorithm is polynomial in n[†]. In other words, the learner must make efficient use of both time and data.

Observe that due to the probabilistic nature of the stochastic selection models, for these models we must relax the requirements on the accuracy of the hypothesis to only

[†]Some researchers do not require that a learning algorithm is a polynomial-time algorithm.

hold with high probability. Likewise, when the learning algorithm is randomized, we only require that the learning algorithm succeeds with high probability. Formally, in the case in which instances are presented stochastically or the learning algorithm is randomized, the learner receives an additional input δ where $0 < \delta < 1$. In these situations the hypothesis is only required to have the desired accuracy with probability at least $1-\delta$, and the sample complexity and time complexity of the learning algorithm may be polynomial in $1/\delta$.

We now return to the first component of the success criteria: the accuracy of the hypothesis. Combined with the requirements of polynomial sample and time complexity, we get the following success criteria: good approximation, weak approximation, exact identification and mistake bounds. We now formally define each of these success criteria.

2.2.1 Good Approximation

Although it would be nice if the learner's hypothesis would classify *every* instance correctly, in many situations this is not possible and we may only require that the learner's hypothesis is a good approximation to the target concept. For example, when the instances are presented stochastically, there might be a very unlikely instance that is never presented. Furthermore, it could be that this instance could not possibly be correctly classified using insight gained from other instances. So we ask only that the learner properly classifies "most" instances.

We formalize the notion of a good approximation as follows. Let D be some probability distribution on the instance space. In the *good-approximation success criteria*, the learner is also given as input ϵ such that $0 < \epsilon < 1$. The learner's goal is to output a hypothesis h that has probability at most ϵ of disagreeing with the target concept c on a randomly drawn instance from D (thus, the hypothesis has *accuracy* at least $1 - \epsilon$). If such a learning algorithm A exists (that is, an algorithm A meeting the goal for any $n \geq 1$, any $c \in C_n$, any distribution D, and any ϵ), we say that A meets the good-approximation success criteria. In this setting we allow the sample and time complexity

of the learning algorithm to also be polynomial in $1/\epsilon$.

2.2.2 Weak Approximation

Another possibility, as opposed to asking the learner to output a hypothesis giving an arbitrarily good approximation to the target concept, is to just ask that the hypothesis perform slightly better than random guessing. This weaker success criteria is natural for applications in which the learner cannot obtain overwhelming accuracy in prediction, but may be able to obtain a significant advantage over guessing.

We formalize the notion of weak approximation as follows. As above, let D be some probability distribution on the instance space. The learner's goal is to output a hypothesis h that has probability at most $\frac{1}{2} - \frac{1}{p(n)}$, for some polynomial $p(n)$ (where $p(n) > 0$ for all n), of disagreeing with c on a randomly drawn instance from D. (Thus the hypothesis has accuracy at least $\frac{1}{2} + \frac{1}{p(n)}$.) If such a learning algorithm A exists (that is, an algorithm A meeting the goal for any $n \geq 1$, any $c \in C_n$, and any distribution D), we say that A meets the weak-approximation success criteria.

2.2.3 Exact Identification

For some settings, it is reasonable to ask that learner output a hypothesis that is equivalent to the target concept rather than just outputting an approximation to the target concept. Typically, this criterion is applied when the learner receives labeled examples either selected by the learner or by a teacher. We formalize this criterion as follows. The learner's goal is to output a hypothesis h that classifies all instances exactly as they are classified by the target concept. If such an algorithm exists (that is, an algorithm A that outputs a hypothesis that is equal to the target concept on all inputs for any $n \geq 1$, and any $c \in C_n$), we say that A meets the exact-identification success criteria.

2.2.4 Mistake Bounds

Another way to evaluate the performance of a learning algorithm is according to the mistakes made by the learner in the performance phase of the learning session. (We consider an on-line learning algorithm to always be in the performance phase.) We now describe two ways in which this notion has been formalized.

Probabilistic Mistake Bound: Let D be some probability distribution on the instance space. The *probabilistic mistake bound* is the probability that the learner's hypothesis disagrees with c on the t^{th} randomly drawn instance from D. Formally, given any $n \geq 1$ and any $c \in C_n$, the learner's goal is to output a hypothesis h such that the probability that h makes a mistake on the $t+1^{st}$ trial is at most $p(n)t^{-\beta}$ for some polynomial $p(n)$ and $0 < \beta$.

Absolute Mistake Bound: The *absolute mistake bound* is the worst-case total number of mistakes made when the learner must make predictions for any, possibly infinite, sequence of instances. (Even if the instance space is finite, repetitions may occur.) Formally, given any $n \geq 1$ and any $c \in C_n$, the learner's goal is to make at most $p(n)$ mistakes for some polynomial $p(n)$.

See Haussler et al. [28] for a discussion of the relationship between these mistake-bound success criteria and the good-approximation criteria. Observe that while we have given clear-cut notions of when a learning algorithm succeeds, a nice feature of the prediction models is that they provide a way to compare the performance of "good" algorithms. In particular, we can compare learning algorithms that achieve polynomial absolute mistake bounds according to the asymptotic value of their mistake bounds.

2.3 Review of Current Models

In this section, we review the commonly used learning models. For each model we describe how it combines the learner's interaction with the environment and the required success

criteria. Then in Section 2.4 we present our extended mistake-bound model.

2.3.1 Distribution-Free Learning Model

The distribution-free learning model introduced by Valiant [66] combines the batch setting with distribution-free stochastic instance selection. The success criteria used by this model is the good-approximation criteria (with high probability) where the distribution used to judge the learner's hypothesis is the same distribution from which the instances are drawn.

Putting it all together, we get the following model. The learner is given access to labeled (positive and negative) instances of the target concept, drawn randomly according to some unknown distribution D over X_n. The learner is also given as input ϵ and δ such that $0 < \epsilon, \delta < 1$. The learner's goal is to output with probability at least $1 - \delta$ a hypothesis h that has probability at most ϵ of disagreeing with c on a randomly drawn instance from D. In this model, a polynomial-time learning algorithm must have time and sample complexity that are polynomial in n, $1/\epsilon$ and $1/\delta$. If such a learning algorithm A exists (that is, an algorithm A meeting the goal for any $n \geq 1$, any $c \in C_n$, any distribution D, and any ϵ, δ), we say that C is *learnable in the distribution-free model*. This model is also known as "probably approximately correct" (PAC) learning model.

In his paper, Valiant [66] gives efficient learning algorithms for the class of monomials, k-CNF[‡] and k-DNF. Some of the other classes shown to be efficiently learnable in this model are k-decision lists [56], rectangles in n-dimensional space [10], intersection of n half planes in 2-dimensional space [10]. Recently, Helmbold, Sloan and Warmuth have shown that the nested difference of any concepts known to be PAC-learnable is also PAC-learnable [31]. For the concept classes mentioned above, the learning algorithms output a hypothesis from the given concept class. However, in some cases the learner must use a more expressive hypothesis space. For example, Pitt and Valiant [52] prove that k-term DNF is not efficiently learnable using a hypothesis from k-term DNF unless

[‡]Unless stated otherwise, assume k is a constant.

$\mathcal{RP} = \mathcal{NP}$. However, they give an efficient algorithm for learning k-term DNF using hypotheses from the class k-CNF. There are dual results for k-clause-CNF. Thus the choice of the hypothesis space may be significant in determining what concept classes are efficiently learnable.

Although there are many positive results for this learning model, there are also many negative results. As we mentioned above, there are some *representation dependent* hardness result. Recently, Kearns and Valiant [36] have proved *representation independent* hardness results for Boolean formulas, acyclic deterministic finite automata, and constant-depth threshold circuits. These hardness results are based on cryptographic assumptions. Recently Schapire [58] has shown pattern languages are not PAC-learnable, assuming $\mathcal{P}/\text{poly} \neq \mathcal{NP}/\text{poly}$. This result holds regardless of the representation used by the learning algorithm. One way to overcome these hardness results is to relax the restriction of learning under any distribution [7, 38, 39].

In this model, an important contribution in characterizing what concept classes are learnable was made by Blumer et al. [10]. We first need the following definitions. A finite set $Y \subseteq X$ is *shattered* by C if $\{c \cap Y \mid c \in C\} = 2^Y$. The *Vapnik-Chervonenkis dimension* of C, denoted $\text{VCD}(C)$, is defined to be the smallest d for which no set of $d+1$ points is shattered by C. Building on the work of Vapnik and Chervonenkis [67], Blumer et al. proved that any PAC-learning algorithm requires at least $\Omega(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{\text{VCD}(C)}{\epsilon})$ instances in the training phase. Furthermore, they proved that the general technique of finding a hypothesis consistent with a set of $\Theta(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{\text{VCD}(C)}{\epsilon}\ln\frac{1}{\epsilon})$ instances, when feasible, always results in a (possibly super-polynomial time) PAC-learning algorithm. Thus C is PAC-learnable (disregarding computation time) if and only if $\text{VCD}(C)$ is finite[§]. From this result we see that there are nearly tight bounds on the sample complexity required for PAC-learning. The results of Blumer et al. assume a model of *static sampling* where the number of examples in the training phase must be independent of the target concept. However, Linial, Mansour and Rivest [40] have shown that with *dynamic sampling*

[§]There are some technical restrictions on the concept classes for which this result applies.

(i.e. the number of examples may depend on the complexity of the target concept) various concept classes with *infinite* VC-dimension are learnable. Essentially, those concept classes with infinite VC-dimension that are learnable are those the can be written as a countable union $C = C_1 \cup C_2 \cup \ldots$ where for each concept subclass C_d, $\text{VCD}(C_d) \leq d$.

2.3.2 Weak-learning Model

Kearns and Valiant [36] introduced a weaker model of learnability than Valiant's original model. This model of *weak learnability* drops the requirement that the learner be able to achieve arbitrarily high accuracy; but rather need only output a hypothesis that performs slightly better (by a polynomial fraction) than random guessing. This model uses the batch setting with instances generated by the distribution-free stochastic setting. The difference from the PAC-model is that the success criteria used is the weak-approximation criteria (with high probability). Thus we sometimes refer to Valiant's original model as the *strong-learning* model.

Thus in the weak-learning model, the learner is given access to labeled (positive and negative) instances of the target concept, drawn randomly according to some unknown distribution D over X_n. The learner is also given as input δ such that $0 < \delta < 1$. The learner's goal is to output with probability at least $1-\delta$ a hypothesis h that has probability at most $\frac{1}{2} - \frac{1}{p(n)}$ (for some polynomial $p(n)$) of disagreeing with c on a randomly drawn instance from D. If such a learning algorithm A exists (that is, an algorithm A meeting the goal for any $n \geq 1$, any $c \in C_n$, any distribution D, and any δ), we say that C is *weakly learnable in the distribution-free model*. In this model, a polynomial-time learning algorithm must have time and sample complexity that are polynomial in n and $1/\delta$.

Recently, Schapire [59] has proved the surprising result that C can be weakly learned in polynomial time if and only if it can be PAC-learned in polynomial time. More precisely, he gives an efficient PAC-learning algorithm for C that uses a weak learning algorithm for C as a subroutine. Thus this apparently weak model is in fact equivalent to the stronger PAC-model under distribution-free instance selection. (If the distribution is

fixed, the notion of strong and weak approximation are not equivalent. Namely, Kearns and Valiant [37, 36] have shown that, under a uniform distribution on the instance space, monotone Boolean formulas are weakly, but not strongly, learnable.) While Schapire's result proves that the concept classes learnable in strong and weak learning models are the same, it does not address the question of sample complexity. See Goldman, Kearns, and Schapire [21] for a discussion of the sample complexity of weak learning.

2.3.3 Prediction Learning Models

For situations in which the accuracy on predicting all instances is important, Littlestone [42] presents an *on-line* (or *mistake-bound*) learning model. This model uses the on-line scenario with adversary selection. The success criteria used is the absolute mistake-bound criteria. In his thesis, Littlestone [42] describes many efficient learning algorithms working under this model. In particular, he gives an algorithm which learns the class of linearly separable Boolean functions with a small absolute mistake bound.

A natural question to ask is: how does this on-line model relate to the distribution-free model? Littlestone [42] describes a reduction for showing that any prediction algorithm making a polynomial number of mistakes can be converted to an algorithm that PAC-learns in the distribution-free model. This reduction is performed in two steps. First it is shown that the number of equivalence queries needed for exact identification is at most the absolute mistake bound plus one; each mistake provides a counterexample to an incorrect hypothesis. Then the following reduction of Angluin [4] is used to show that an exact-identification algorithm making a polynomial number of equivalence queries can be converted to an algorithm that learns in the distribution-free model. Replace the ith equivalence query by requesting $q_i = \left\lceil \frac{1}{\epsilon}(\ln \frac{1}{\delta} + i \ln 2) \right\rceil$ random labeled instances. If the current hypothesis misclassifies any of these q_i instances then return that instance as a counterexample. Otherwise, reply that the hypothesis is correct. So the probability that the ith simulated equivalence query will "okay" a hypothesis with error more than ϵ is at most $(1-\epsilon)^{q_i}$. Thus the probability that this technique will ever "okay" a hypothesis

with error more than ϵ is at most

$$\sum_{i=1}^{\infty}(1-\epsilon)^{q_i} \leq \sum_{i=1}^{\infty} e^{-\epsilon q_i} \leq \sum_{i=1}^{\infty} \frac{\delta}{2^i} \leq \delta.$$

A similar reduction is given by Haussler, Kearns, Littlestone, and Warmuth [28]. Over the Boolean domain $\{0,1\}^n$ it is known that if the learner is allowed unlimited computational resources then a concept class is PAC-learnable if and only if then there exists a prediction algorithm obtaining a polynomial mistake bound. However, Blum [9] has recently shown that, if one-way functions exist, there is a concept class over $\{0,1\}^n$ that is efficiently PAC-learnable, yet *any* efficient prediction algorithm makes an exponential number of mistakes.

Finally we note that Haussler, Littlestone, and Warmuth [29, 30] present a similar *probabilistic prediction* learning model. Like the on-line model, this model uses the on-line scenario with adversary selection. However, the probabilistic mistake-bound success criteria is used. They use this model to analyze learning algorithms for predicting $\{0,1\}$-valued functions over $\Re^n$.

2.3.4 Learning with Queries

We now discuss learning models that use queries to learn the unknown target concept. While there have been many types of queries proposed, we focus on membership queries and equivalence queries. For a more complete discussion on this topic, we refer the reader to Angluin's papers [1, 2, 3, 4]. Most work on learning with queries uses the exact-identification success criteria. That is, after seeing a polynomial number of labeled instances (as specified by the queries) the learner must output a hypothesis that correctly classifies all instances. Angluin, Hellerstein and Karpinski [5] give a polynomial-time algorithm that exactly identifies any monotone read-once formula from membership queries. If equivalence queries are also provided then a larger class of concepts can be exactly identified. For example, there are polynomial-time algorithms that achieve exact identification for the following concept classes: finite state automata [4], pattern lan-

guages [4], arbitrary read-once formulas [5], and read-once formula decision trees [24]. We note that Angluin's algorithm for inferring a finite state automaton critically relies on the use of a "reset". Recently, Rivest and Schapire [57] have extended Angluin's result to obtain a randomized algorithm that with high probability infers the given finite state automaton without the use of a reset. In other words, they have limited the use of arbitrary experimentation (membership queries) to allow the learner to perform only those experiments that are feasible in the current state.

A learning model that is quite similar to learning with membership and equivalence queries is *PAC-learning with membership queries* [4]. Here one uses the batch model with a combination of stochastic and learner selection. Like in the PAC-model, the success criteria is to output a good approximation to the target concept with high probability. Angluin [4] has shown that any learning algorithm using equivalence queries to achieve exact identification may be modified to learn in the PAC-model by simulating the equivalence queries by drawing random examples (as discussed above in Section 2.3.3). It follows from this result that any algorithm that achieves exact identification using membership and equivalence queries can be modified to obtain an algorithm that PAC-learns with membership queries.

2.4 Extended Mistake-Bound Model

As we saw in the previous section, the ways that a learner may interact with his environment and various success criteria have been combined in many ways yielding many interesting learning models. However, there are many other ways to combine these features. Not all combinations are reasonable: for example, one could imagine having a batch model with distribution-free stochastic presentation and require that a polynomial-time algorithm achieve exact identification (with high probability). However, this model is not interesting for any concept classes. Imagine a distribution in which some instance (that is not determined by the other instances) is given exponentially small weight. Since it is

unlikely that this instance will be seen in a polynomial-sized sample, no algorithm could achieve exact identification with high probability.

Nevertheless, there are many interesting combinations that have not been considered. We are particularly interested in an on-line model with the absolute mistake-bound criteria. In the next chapter we describe in some detail why this combination of features is appropriate for the problem of learning a binary relation. For now, we just briefly mention some of the reasons for choosing such a model. First of all, unlike the PAC-model, a mistake-bound model is appropriate for problems in which the instance space is polynomial-sized in the natural dimension measure. Just as running time is used in algorithm analysis, we can use mistake bounds to judge the performance of learning algorithms. However, we want to study the relation between the various presentation methods. Not surprisingly, the number of mistakes made by a prediction algorithm depends on the sequence of instances presented to the learner. Unlike previous work in which only adversary selection has been used with absolute mistake bounds, we will combine the absolute mistake-bound success criteria with other selection methods.

We now formally describe the resulting model which we call the *extended mistake-bound model*. The *query sequence* is a permutation $\pi = \langle x_1, x_2, \ldots, x_{|X_n|} \rangle$ of X_n where x_t is the instance presented to the learner at the t^{th} trial. We call the agent selecting the query sequence the *director*. We consider the following directors:

- **Learner**— The learner chooses π. To select x_t, the learner may use time polynomial in n, and all information obtained in the first $t-1$ trials. In this case we say that the learner is *self-directed*.

- **Helpful Teacher**— A teacher, who knows the target concept and wants to minimize the learner's mistakes, chooses π. To select x_t, the teacher uses knowledge of the target concept, $x_1, \ldots, x_{t-1}$, and the learner's predictions on $x_1, \ldots, x_{t-1}$. To avoid allowing the learner and teacher to have a coordinated strategy, in this scenario we consider the worst-case mistake bound over all consistent learners**. In

**Recall that a learner is consistent if, on every trial, there is some concept in C_n that agrees with the

this case we say the learner is *teacher-directed.*

- **Adversary**— The adversary who selected the target concept chooses π. This adversary, who tries to maximize the learner's mistakes, knows the learner's algorithm and has unlimited computing power. In this case we say the learner is *adversary-directed.*

- **Random**— In this model, π is selected randomly according to a uniform probability distribution on the permutations of X_n. Here the number of mistakes made by the learner for some target concept c in C_n is defined to be the expected number of mistakes over all possible query sequences. In this case we say the learner is *randomly-directed.*

We consider how a prediction algorithm's performance depends on the director. Namely, we let $\mathrm{MB}_Z(A, C_n)$ denote the worst case number of mistakes made by A for any target concept in C_n under any query sequence provided by Z. (When Z = adversary, $\mathrm{MB}_Z(A, C_n) = M_A(C_n)$ as defined by Littlestone [43].) We say that A is a *polynomial prediction algorithm* if A makes each prediction in time polynomial in n.

learner's prediction, as well as with the labeled instances observed on all the preceding trials.

Chapter 3

Learning Binary Relations

In this chapter we study the problem of learning a binary relation (represented as a matrix) when the learner has prior knowledge that there are a limited number of row types. We study this problem using the extended mistake-bound model introduced in Chapter 2.

Recall that a binary relation is defined between two sets of objects. Throughout this chapter, we assume that one set has cardinality n and the other has cardinality m. We also assume that for all possible pairings of objects, the predicate relating the two sets of variables is either true (1) or false (0). In this chapter we represent the relation as an $n \times m$ binary matrix, where an entry contains the value of the predicate for the corresponding elements. Since the predicate is binary-valued, all entries in this matrix are either 0 (false) or 1 (true). The *two-dimensional* structure arises from the fact that we are learning a binary relation.

For the sake of comparison, we now briefly mention some other representations. One could represent the relation as a table with two columns, where each entry in the first column is an item from the first set and each entry in the second column is an item from the second set. The rows of the table consist of the subset of the potential nm pairings for which the predicate is true. Another possibility is to represent the relation as a bipartite

This chapter describes joint research with Ron Rivest and Rob Schapire [22].

graph with n vertices in one vertex set and m vertices in the other set. An edge is placed between two vertices exactly when the predicate is true for the corresponding items.

If the learner is to have any hope of doing better than random guessing, there must be some structure in the relation. Furthermore, since there are so many ways to structure a binary relation, we give the learner prior knowledge about the nature of this structure. Not surprisingly, the learning task depends greatly on the prior knowledge provided. In this chapter we impose structure by restricting the matrix to have at most k distinct row types. (Two rows are of the same type if they agree in all columns.) Now the adversary cannot force nm mistakes since to do so may cause the created matrix to have more than k row types. We define a *k-binary-relation* to be a binary relation for which the corresponding matrix has at most k row types. This restriction is satisfied whenever there are only k types of objects in the set of n objects being considered in the relation. The learner receives *no* other knowledge about the predicate forming the relation.

For the concept class C_{nm} of k-binary-relations, the dimension measure is nm, the number of entries in the corresponding matrix, and $X_{nm} = \{1, \cdots, n\} \times \{1, \cdots, m\}$. An instance (i, j) is a positive instance if and only if the matrix entry in row i and column j is a 1. So in each trial the learner is repeatedly given an instance x from X_{nm} and asked to predict the corresponding matrix entry. After making its prediction, the learner is told the correct value of the matrix entry. The learner wishes to minimize the number of incorrect predictions it makes. Since we assume that the learner must eventually make a prediction for each matrix entry, the number of incorrect predictions depends on the size of the matrix.

For this concept class, we prove that any learning algorithm makes at least $(1-\beta)km + n\lfloor \lg(\beta k) \rfloor - (1-\beta)k\lfloor \lg(\beta k) \rfloor$ mistakes in the worst case for any fixed $0 < \beta \leq 1$ against any query sequence. So for $\beta = 1/2$, we get a lower bound of $\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$ on the number of mistakes made by any learner. If computational efficiency is not a concern, the halving algorithm [6, 43] makes at most $km + (n-k)\lg k$ mistakes against any query sequence. (The halving algorithm predicts according to the majority of the

feasible relations (or concepts), and thus each mistake halves the number of remaining relations.)

We present an efficient algorithm making at most $km+(n-k)\lceil \lg k \rceil$ mistakes with the learner as the director. We prove a tight mistake bound of $km + (n-k)(k-1)$ in the case that the helpful teacher is the director. For adversary-directed learning we present an efficient algorithm for $k = 2$ that makes at most $2m + n - 2$ mistakes, and for arbitrary k we present an efficient algorithm making at most $km + n\sqrt{(k-1)m}$ mistakes. We prove any algorithm makes at least $km+(n-k)\lfloor \lg k \rfloor$ mistakes when the adversary is the director, and use the existence of projective geometries to improve this lower bound to $\Omega(km + (n-k)\lfloor \lg k \rfloor + \min\{n\sqrt{m}, m\sqrt{n}\})$ for a large class of algorithms. Finally, we describe a technique to simplify the proof of expected mistake bounds when the query sequence is chosen at random, and use it to prove an $O(km + nk\sqrt{H})$ expected mistake bound for a simple algorithm. (Here H is the maximum Hamming distance between any two rows.)

In the next section we give a motivating example from the domain of testing for allergies. We use this example to motivate both the restriction that the matrix has k row types and the use of the extended mistake-bound model. We then present general upper and lower bounds on the number of mistakes made by the learner regardless of the director. Finally, we study the complexity of learning a k-binary-relation for self-directed, teacher-directed, adversary-directed and randomly-directed learning.

3.1 Motivation: Allergist Example

In this section we use the following example taken from the domain of allergy testing to motivate the problem of learning a k-binary relation.

Consider an allergist with a set of patients to be tested for a given set of allergens. Each patient is either *highly allergic*, *mildly allergic*, or *not allergic* to any given allergen. The allergist may use either an *epicutaneous* (scratch) test in which the patient is given a

	Epicutaneous (Scratch)	Intradermal (Under the Skin)
Not Allergic	negative	negative
Mildly Allergic	negative	weak positive
Highly Allergic	weak positive	strong positive

Figure 3-1: Summary of testing reactions for allergy testing example.

fairly low dose of the allergen, or an *intradermal* (under the skin) test in which the patient is given a larger dose of the allergen. The patient's reaction to the test is classified as *strong positive*, *weak positive* or *negative*. Figure 3-1 describes the reaction that occurs for each combination of allergy level and dosage level. Finally, we assume a strong positive reaction is extremely uncomfortable to the patient, but not dangerous.

What options does the allergist have in testing a patient for a given allergen? He could just perform the intradermal test (option 0). Another option (option 1) is to perform an epicutaneous test, and if it is not conclusive, then perform an intradermal test. (See Figure 3-2 for decision trees describing these two testing options.) Which testing option is best? If the patient has no allergy or a mild allergy to the given allergen, then testing option 0 is best, since the patient need not return for the second test. However, if the patient is highly allergic to the given allergen, then testing option 1 is best, since the patient does not experience a bad reaction. We assume the inconvenience of going to the allergist twice is approximately the same as having a bad reaction. That is, the allergist has no preference to error in a particular direction. While the allergist's final goal is to determine each patient's allergies, we consider the problem of learning the optimal testing option for each combination of patient and allergen.

The allergist interacts with the environment as follows. In each "trial" the allergist is asked to predict the best testing option for a given patient/allergen pair. He is then told the testing results, thus learning whether the patient is not allergic, mildly allergic or highly allergic to the given allergen. In other words, the allergist receives feedback as

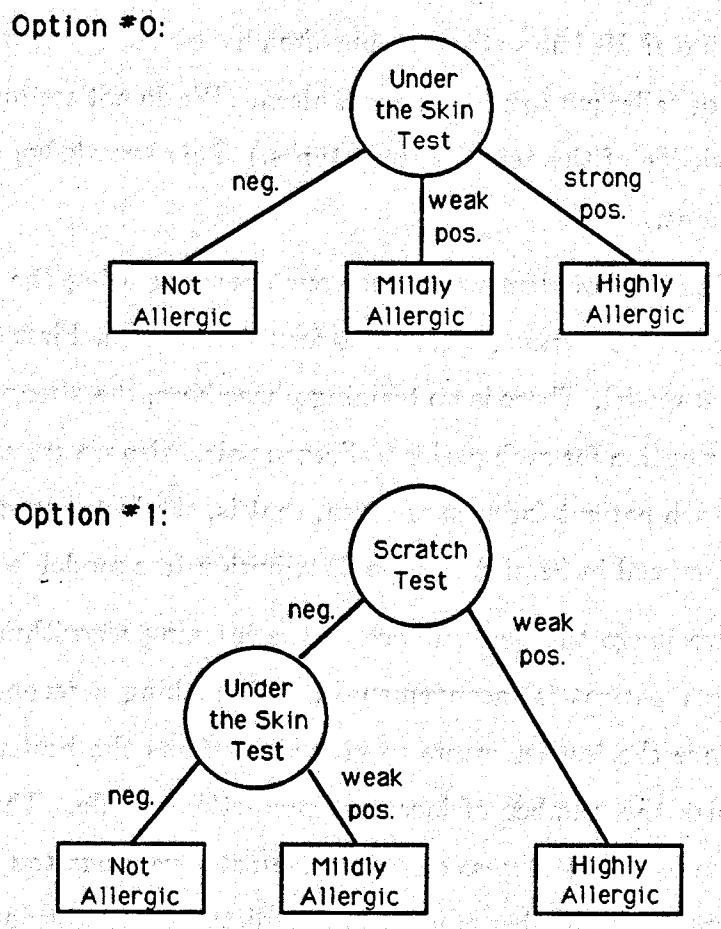


Figure 3-2: The testing options available to the allergist.

to the correct testing option. Note that we make no restrictions on how the hypothesis is represented as long as it can be evaluated in polynomial time. In other words, all we require is that given any patient/allergen pair, the allergist decides which test to perform in a "reasonable" amount of time.

How can the allergist possibly predict a patient's allergies? If the allergies of the patients are completely "random," then there is not much hope. What prior knowledge does the allergist have? He knows that people often have exactly the same set of allergies. So there are a set of "allergy types" that occur often. (We do not assume that the allergist has a priori knowledge of the actual allergy types.) This knowledge can help guide the allergist's predictions.

Having specified the problem we discuss our choice of using the extended mistake-bound model to evaluate learning algorithms for this problem. First of all, observe that we want an on-line model. There is no training phase here, the allergist wants to predict the correct testing option for each patient/allergen pair. Also we expect that the allergist has time to test each patient for each allergen, that is, the instance space is polynomial-sized. Thus as discussed in Section 2.4 the distribution-free model is not appropriate.

How should we judge the performance of the learning algorithm? For each wrong prediction made, a patient is inconvenienced with making a second trip or having a bad reaction. Since the learner wants to give all patients the best possible service, he strives to minimize the number of incorrect predictions made. Thus we want to use the absolute mistake-bound success criterion. Namely, we judge the performance of the learning algorithm by the number of incorrect predictions made during a learning session in which he must eventually test each patient for each allergen.

Up to now, the standard on-line model (using absolute mistake bounds) appears to be the appropriate model. We now discuss the selection of the instances. Since the allergist has no control over the target relation (i.e. the allergies of his patients), it makes sense to view the feedback as coming from an adversary. However, do we really want an adversary to select the presentation order for the instances? It could be that the allergist is working

for a cosmetic company and, due to restrictions of the Food and Drug Administration and the cosmetic company, the allergist is essentially told when to test each person for each allergen. In this case, it is appropriate to have an adversary select the presentation order. However, in the typical situation, the allergist can decide in what order to perform the testing so that he can make the best predictions possible. In this case, we want to allow the learner to select the presentation order. One could also imagine a situation in which an intern is being guided by an experienced allergist, and thus a teacher helps to select the presentation order. Finally, random selection of the presentation order may provide us with a better feeling for the behavior of an algorithm. Thus the learning model that is most appropriate for this example is the extended mistake-bound model.

3.2 General Mistake Bounds

In this section we begin our study of learning k-binary-relations by presenting general lower and upper bounds on the mistakes made by the learner regardless of the director.

Throughout this section, we use the following notation: We say an entry (i, j) of the matrix (M_{ij}) is *known* if the learner was previously presented that entry. We assume without loss of generality that the learner is never asked to predict the value of a known entry. We say rows i and i' are *consistent* (given the current state of knowledge) if $M_{ij} = M_{i'j}$ for all columns j in which both entries (i, j) and (i', j) are known.

We now look at general lower and upper bounds on the number of mistakes that apply for all directors. Clearly, any learning algorithm makes at least km mistakes for some matrix, regardless of the query sequence. The adversary can divide the rows into k groups and reply that the prediction was incorrect for the first column queried for each entry of each group. We generalize this approach to force mistakes for more than one row of each type.

Theorem 3.1 *For any $0 < \beta \leq 1$, any prediction algorithm makes at least $(1-\beta)km + n\lfloor \lg(\beta k) \rfloor - (1-\beta)k\lfloor \lg(\beta k) \rfloor$ mistakes regardless of the query sequence.*

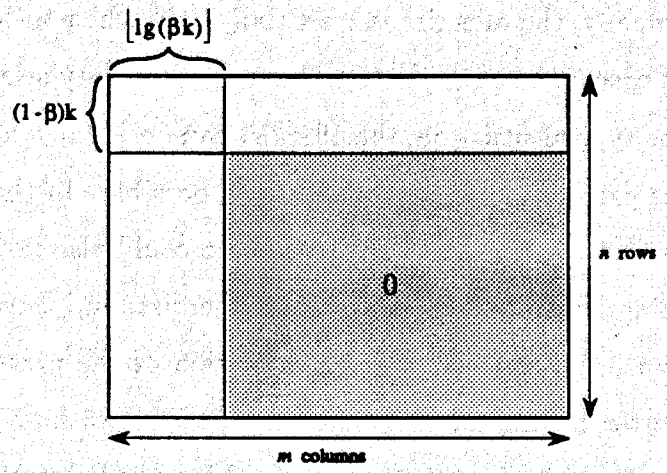


Figure 3-3: The final matrix created by the adversary in the proof of Theorem 3.1. All entries in the unmarked area will contain the bit not predicted by the learner. That is, a mistake is forced on each entry in the unmarked area. All entries in the marked area are 0.

Proof: The adversary selects its feedback for the learner's predictions as follows. For each entry in the first $\lfloor \lg(\beta k) \rfloor$ columns the adversary replies that the learner's response is incorrect. At most βk new row types are created by this action. Likewise, for each entry in the first $(1-\beta)k$ rows the adversary replies that the learner's response is incorrect. This creates at most $(1-\beta)k$ new row types. The adversary makes all remaining entries in the matrix 0. (See Figure 3-3.) The number of mistakes is at least the area of the unmarked region. Thus the adversary has forced at least $(1-\beta)km + n\lfloor \lg(\beta k) \rfloor - (1-\beta)k\lfloor \lg(\beta k) \rfloor$ mistakes while creating at most $\beta k + (1-\beta)k = k$ row types. ∎

By letting $\beta = \frac{1}{2}$ we obtain the following corollary.

Corollary 3.1 *Any algorithm makes at least* $\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$ *mistakes in the worst case regardless of the query sequence.*

If computational efficiency is not a concern, for all query sequences the halving algorithm [6, 43] provides a good mistake bound.

Observation 3.1 *The halving algorithm makes at most $km+n\lg k$ mistakes in the worst case.*

Proof: We use a simple counting argument on the size of the concept class C_{nm}. There are 2^{km} ways to select the k row types, and k^n ways to assign one of the k row types to each of the n rows. Thus $|C_{nm}| \leq 2^{km}k^n$. Littlestone [43] proves that the halving algorithm makes at most $\lg|C_{nm}|$ mistakes. Thus the number of mistakes made by the halving algorithm for this concept class is at most $\lg(2^{km}k^n) \leq km + n\lg k$. ∎

In the remainder of this section we study efficient prediction algorithms designed to perform well against each of the directors. For some directors, we are also able to prove information-theoretic lower bounds that are better than that of Theorem 3.1. In Section 3.3, we consider the case that the query sequence is selected by the learner. We study the helpful-teacher director in Section 3.4. In Section 3.5 we consider the case of an adversary as the director. Finally, in Section 3.6 we consider when the instances are drawn at random according to a uniform distribution on the instance space.

3.3 Self-directed Learning

In this section we present an efficient algorithm for learning the matrix for the case in which the learner is the director.

Theorem 3.2 *There exists a polynomial prediction algorithm that makes at most $km + (n-k)\lfloor\lg k\rfloor$ mistakes under self-directed learning.*

Proof: The query sequence selected simply specifies the entries of the matrix in row-major order. The learner begins assuming there is only one row type. Let $\hat{k}$ denote the learner's current estimate for k. So initially $\hat{k} = 1$. For the first row, the learner guesses each entry. (This row becomes the template for the first row type.) Next the learner assumes that the second row is the same as the first row. If he makes a mistake then the learner revises his estimate for $\hat{k}$ to be 2, guesses for the rest of the row, and uses that row as the template for the second row type. In general, to predict M_{ij}, the learner

predicts according to a majority vote of the recorded row templates that are consistent with row i (breaking ties arbitrarily). Thus, if a mistake is made, then at least half of the row types can be eliminated as the potential type of row i. If more than $\lfloor \lg \hat{k} \rfloor$ mistakes are made in a row, then a new row type has been found. In this case, $\hat{k}$ is incremented, the learner guesses for the rest of the row, and makes this row the template for row type $\hat{k}+1$.

How many mistakes are made by this algorithm? Clearly, at most m mistakes are made for the first row found of each of the k types. For the remaining $n-k$ rows, since $\hat{k} \leq k$, at most $\lfloor \lg k \rfloor$ mistakes are made. ■

Note that this algorithm need not know k a priori. Furthermore, it obtains the same mistake bound even if an adversary tells the learner which row to examine, and in what order to predict the columns, provided that the learner sees all of a row before going on to the next. We note that this upper bound is within a constant factor of the lower bound of Corollary 3.1. However, this problem becomes harder if the adversary can select the query sequence without restriction.

3.4 Teacher-directed Learning

In this section we present upper and lower bounds on the number of mistakes made under the helpful-teacher director. Recall that in this model, we consider the worst-case mistake bound over all consistent learners. Thus the question asked here is: what is the minimum number of matrix entries a teacher must reveal so that there is a unique completion of the matrix? That is, until there is a unique completion of the partial matrix, a mistake could be made on the next prediction.

We now prove an upper bound on the number of entries needed to uniquely define the target matrix.

Theorem 3.3 *The number of mistakes made with a helpful teacher as the director is at most* $km + (n-k)(k-1)$.

Proof: First, the teacher presents the learner with one row of each type. For each of the remaining $n-k$ rows the teacher presents an entry to distinguish the given row from each of the $k-1$ incorrect row types. After these $km+(n-k)(k-1)$ entries have been presented we claim that there is a unique matrix with at most k row types that is consistent with the partial matrix. Since all k distinct row types have been revealed in the first stage, all remaining rows must be the same as one of the first k rows presented. However, each of the remaining rows have been shown to be inconsistent with all but one of these k row templates. ■

Is Theorem 3.3 the best such result possible? Clearly the teacher must present a row of each type. But, in general, is it really necessary to present $k-1$ entries of the remaining rows to uniquely define the matrix? We now answer this question in the affirmative by presenting a matching lower bound.

Theorem 3.4 *The number of mistakes made with a helpful teacher as the director is at least* $\min\{nm, km+(n-k)(k-1)\}$.

Proof: The adversary selects the following matrix. The first row type consist of all zeros. For $2 \leq z \leq \min\{m+1, k\}$, row type z contains $z-2$ zeros, followed by a one, followed by $m-z+1$ zeros. The first k rows are each assigned to be a different one of the k row types. Each remaining row is assigned to be the first row type. (See Figure 3-4.) Until there is a unique completion of the partial matrix, by definition there exists a consistent learner that could make a mistake. Clearly if the learner has not seen each column of each row type, then the final matrix is not uniquely defined. This part of the argument accounts for km mistakes. When $m+1 \geq k$, for the remaining rows, unless all of the first $k-1$ columns are known, there is some row type besides the first row type that must be consistent with the given row. This argument accounts for $(n-k)(k-1)$ mistakes. Likewise, when $m+1 < k$, if any of the first m columns are not known then there is some row type besides the first row type that must be consistent with the given row. This accounts for $(n-k)m$ mistakes. Thus the total number of mistakes is at least $\min\{nm, km+(n-k)(k-1)\}$. ■

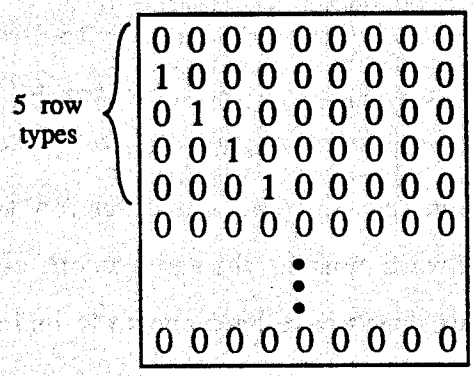


Figure 3-4: The matrix created by the adversary against the helpful-teacher director. In this example, there are 5 row types which appear in the first five rows of the matrix.

Due to the requirement that mistake bounds in the teacher-directed case apply to all consistent learners, we note that it is possible to get mistake bounds that are not as good as those obtained when the learner is self-directed. Recall that in the previous section, we proved a $km+(n-k)\lfloor \lg k \rfloor$ mistake bound under self-directed learning. This bound is better than that obtained with a teacher because the learner uses a majority vote among the known row types for making predictions. However, a consistent learner may use a *minority vote* and could thus make $km + (n-k)(k-1)$ mistakes.

3.5 Adversary-directed Learning

In this section we derive upper and lower bounds on the number of mistakes made when an adversary is the director. We first present an information-theoretic lower bound on the number of mistakes an adversary can force the learner to make. Next, we present an efficient prediction algorithm that achieves an optimal mistake bound if $k \leq 2$. Next we consider the related problem of computing the minimum number of row types needed to complete a partially known matrix. We then consider learning algorithms that work against an adversary for arbitrary k.

We now present an information-theoretic lower bound on the number of mistakes made by any prediction algorithm when the adversary selects the query sequence. We obtain this result by modifying the technique used in Theorem 3.1.

Theorem 3.5 *Any prediction algorithm makes at least* $\min\{nm, km + (n-k)\lfloor \lg k \rfloor\}$ *mistakes against an adversary-selected query sequence.*

Proof: The adversary starts by presenting all entries in the first $\lfloor \lg k \rfloor$ columns (or m columns if $m < \lfloor \lg k \rfloor$) and replying that each prediction is incorrect. If $m \geq \lfloor \lg k \rfloor$, this step causes the learner to make $n\lfloor \lg k \rfloor$ mistakes. Otherwise, this step causes the learner to make nm mistakes. Each row can now be classified as one of k row types. Next the adversary presents the remaining columns for one row of each type, again replying that each prediction is incorrect. For $m \geq \lfloor \lg k \rfloor$ this step causes the learner to make $k(m - \lfloor \lg k \rfloor)$ additional mistakes. For the remaining matrix entries, the adversary replies as dictated by the completed row of the same row type as the given row. So the number of mistakes made by the learner is at least $\min\{nm, n\lfloor \lg k \rfloor + km - k\lfloor \lg k \rfloor\} = \min\{nm, km + (n-k)\lfloor \lg k \rfloor\}$. ∎

3.5.1 Special Case: $k = 2$

We now consider efficient prediction algorithms for learning the matrix under an adversary-selected query sequence. (Recall that if efficiency is not a concern the halving algorithm makes at most $km + (n-k)\lg k$ mistakes.) In this section we consider the case that $k \leq 2$, and present an efficient prediction algorithm that performs optimally.

Theorem 3.6 *There exists a polynomial prediction algorithm that makes at most* $2m + n - 2$ *mistakes against an adversary-selected query sequence for* $k = 2$.

Proof: The algorithm uses a graph G whose vertices are the rows of the matrix and that initially has no edges. To predict M_{ij} the algorithm 2-colors the graph G, and then:

1. If no entry of column j is known, it guesses randomly.

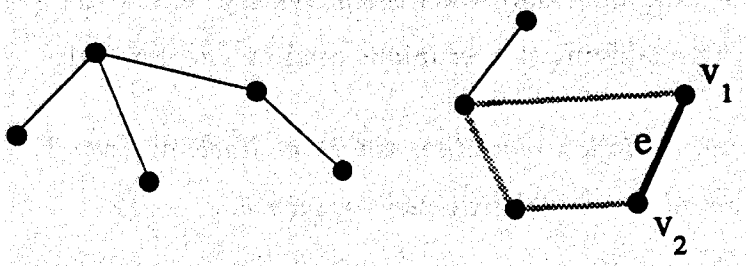


Figure 3-5: The situation occurring if a case 3 mistake does not reduce the number of connected components of G. The thick grey edges and the thick black edge show the cycle created in G. Let e (shown as a thick black edge) be the edge added to form the cycle.

2. Else if every known entry of column j is zero (respectively, one), it guesses zero (one).

3. Else it finds a row i' of the same color as i and known in column j, and guesses $M_{i'j}$.

Finally, after the prediction is made and the feedback received, the graph G is updated by adding an edge $\overline{ii'}$ to G for each row i' known in column j for which $M_{ij} \neq M_{i'j}$. Note that one of the above cases always applies. Also, since $k = 2$, it will always be possible to find a 2-coloring.

How many mistakes can this algorithm make? It is not hard to see that cases 1 and 2 each occur only once for every column, so there are at most m mistakes made in each of these cases. Furthermore, the first case 2 mistake adds at least one edge to G. We now argue that each case 3 mistake reduces the number of connected components of G by at least 1. We use a proof by contradiction. That is, assume that a case 3 mistake does *not* reduce the number of connected components. Thus it follows that the edge $e = \overline{v_1 v_2}$ added to G must form a cycle. (See Figure 3-5.) We now separately consider the cases that this cycle contains an odd number of edges or an even number of edges.

- **Case 1: Odd-length cycle.** Since G is assumed to be 2-colorable, this case cannot occur.

- **Case 2: Even-length cycle.** Before adding e, since v_1 and v_2 were connected by an odd number of edges, in any legal 2-coloring they must have been different colors. Since Step 3 of the algorithm picks nodes of the same color, an edge could have never been placed between v_1 and v_2. Thus we again have a contradiction.

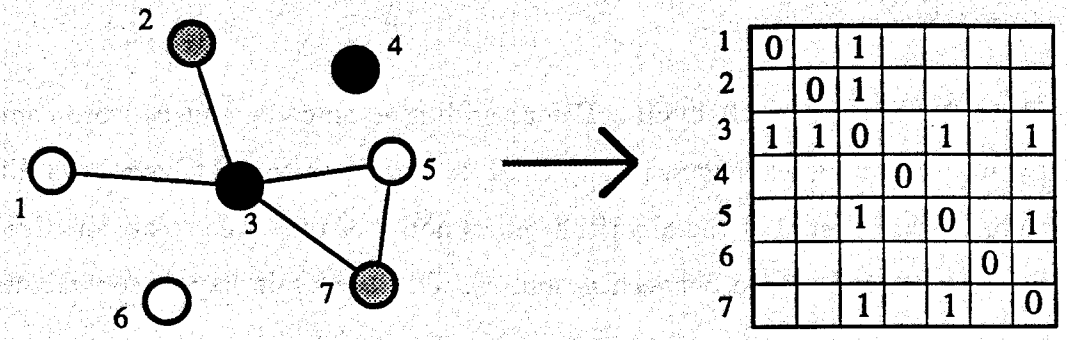
In both cases we reach a contradiction, and thus we have shown that after every case 3 mistake reduces the number of connected components of G. Thus after at most $n-2$ case 3 mistakes, G must be fully connected and thus there must be a unique 2-coloring* of G and no more mistakes can occur. Thus, the worst-case number of mistakes made by this algorithm is $2m+n-2$. ■

Note that for $k=2$ this upper bound matches the information-theoretic lower bound of Theorem 3.5. We also note that if there is only one row type then the algorithm given in Theorem 3.6 makes at most m mistakes, matching the information-theoretic lower bound.

An interesting theoretical question is to find a linear mistake bound for constant $k \geq 3$ when provided with a k-colorability oracle. However, such an approach would have to be greatly modified to yield a polynomial prediction algorithm since a polynomial-time k-colorability oracle exists only if $\mathcal{P} = \mathcal{NP}$. Furthermore, even good polynomial-time approximations to a k-colorability oracle are not known [8, 41].

The remainder of this section focuses on designing polynomial prediction algorithms for the case that the matrix has at least three row types. One approach that may seem promising is to make predictions as follows: Compute a matrix that is consistent with all known entries and that has the fewest possible row types. Then use this matrix to make the next prediction. We now show that even computing the minimum number of row types needed to complete a partially known matrix is $\mathcal{NP}$-complete. Formally, we

*Two 2-colorings under renaming of the colors are considered to be the same.



1	0		1				
2		0	1				
3	1	1	0		1		1
4				0			
5			1		0		1
6						0	
7			1		1		0

Figure 3-6: An example of the reduction used in Theorem 3.7. The graph G is the instance for the graph coloring problem. The partial matrix M is the instance for the matrix complexity problem. We note that there exists a matrix that is an completion of M that uses only three row types. The corresponding 3-coloring of G is demonstrated by the node colorings used in G.

define the *matrix k-complexity* problem as follows: given an $n \times m$ binary matrix M that is partially known, decide if there is some matrix with at most k row types that is consistent with M. The matrix k-complexity problem can be shown to be $\mathcal{NP}$-complete by a reduction from graph k-colorability for any fixed $k \geq 3$.

Theorem 3.7 *For fixed $k \geq 3$, the matrix k-complexity problem is $\mathcal{NP}$-complete.*

Proof: We use a reduction from graph k-colorability. Given an instance $G = (V, E)$ of graph k-colorability we transform it into an instance of the matrix k-complexity problem. Let $m = n = |V|$. For each edge $\{v_i, v_j\} \in E$, we add entries to the matrix so that row i and row j cannot be the same row type. Specifically, for each vertex v_i, we set $M_{ii} = 0$, and $M_{ji} = 1$ for each neighbor v_j of v_i. An example demonstrating this reduction is given in Figure 3-6.

We now show that there is some matrix of at most k row types that is consistent with this partial matrix if and only if G is k-colorable. We first argue that if there is a matrix M' consistent with M that has at most k row types then G is k-colorable.

By construction, if two rows are of the same type there cannot be an edge between the corresponding nodes. So just let the node color for each node be the type of the corresponding row in M'.

We now argue that if G is k-colorable, then there exists a matrix M' consistent with M that has at most k row types. By the construction of M, if a set of vertices are the same color in G then the corresponding rows are consistent with each other. Thus there exists a matrix with at most k row types that is consistent with M. ∎

3.5.2 Row-filter Algorithms

In this section we study the performance of a whole class of algorithms designed to learn a matrix with arbitrary complexity k when an adversary selects the query sequence. We say that an algorithm A is a *row-filter algorithm* if A makes its prediction for M_{ij} strictly as a function of j and all entries in the set I of rows consistent with row i and defined in column j. That is, A's prediction is $f(I, j)$ where f is some (possibly probabilistic) function. So, to make a prediction for M_{ij}, a row-filter algorithm considers all rows that could be the same type as row i and whose value for column j is known, and uses these rows in any way one could imagine to make a prediction. For example it could take a majority vote on the entries in column j of all rows that are consistent with row i. Or, of the rows defined in column j, it could select the row that has the most known values in common with row i and predict according to its entry in column j. We have found that many of the prediction algorithms we considered are row-filter algorithms.

Consider the simple row-filter algorithm, *ConsMajorityPredict*, in which $f(I, j)$ computes the majority vote of the entries in column j of the rows in I. (Guess randomly in the case of a tie.) Note that *ConsMajorityPredict* only takes time linear in the number of known entries of the matrix to make a prediction. We now give an upper bound on the number of mistakes made by *ConsMajorityPredict*.

Theorem 3.8 *The algorithm ConsMajorityPredict makes at most* $km + n\sqrt{(k-1)m}$ *mistakes against an adversary-selected query sequence.*

Proof: For all i, let $d(i)$ be the number of rows consistent with row i. We define the *potential* of a partially known matrix to be $\Phi = \sum_{i=1}^{n} d(i)$. We first consider how much the potential function can change over the entire learning session.

Lemma 3.1 *The potential function Φ decreases by at most $\frac{k-1}{k}n^2$ during the learning session.*

Proof: Initially, for all i, $d(i) = n$. So $\Phi_{\text{init}} = n^2$. Let $C(z)$ be the number of rows of type z for $1 \leq z \leq k$. By definition, $\Phi_{\text{final}} = \sum_{z=1}^{k} C(z)^2$. Thus our goal is to minimize $\sum_{z=1}^{k} C(z)^2$ under the constraint that $\sum_{z=1}^{k} C(z) = n$. Using the method of Lagrange multipliers we obtain that Φ_{final} is minimized when for all z, $C(z) = n/k$. Thus $\Phi_{\text{final}} \geq (n/k)^2 k = n^2/k$. So $\Delta\Phi = \Phi_{\text{init}} - \Phi_{\text{final}} \leq n^2 - \frac{n^2}{k} = \frac{k-1}{k}n^2$. ∎

Now that the total decrease in Φ over the learning session is bounded, we need to determine how many mistakes can be made without Φ decreasing by more than $\frac{k-1}{k}n^2$. We begin by noting that Φ is strictly decreasing. Once two rows are found to be inconsistent, they remain inconsistent. So for each i, $d(i)$ is strictly decreasing, and thus Φ is strictly decreasing. So to bound the number of mistakes made by *ConsMajorityPredict* we must compute a lower bound on the amount Φ is decreased by each mistake. Intuitively, one expects Φ to decrease by larger amounts as more of the matrix is seen. We formalize this intuition in the next two lemmas. For a given row type z, let $B(j, z)$ denote the set of matrix entries that are in column j of a row of type z.

Lemma 3.2 *The r^{th} mistake made when predicting an entry in $B(j, z)$. decreases Φ by at least $2(r-1)$.*

Proof: Suppose that this mistake occurs in predicting entry (i, j) where row i is of type z. Consider all the rows of type z. Since $r-1$ mistakes have occurred in column j, at least $r-1$ entries of $B(j, z)$ are known. Since *ConsMajorityPredict* is a row-filter algorithm these rows must be in I. Furthermore, *ConsMajorityPredict* uses a majority voting scheme, and thus if a mistake occurs there must be at least $r-1$ entries in I (and

thus consistent with row i) that differ in column j with row i. Thus if a mistake is made, row i is found to be inconsistent with at least $r-1$ rows it was thought to be consistent with. When two previously consistent rows are found to be inconsistent, Φ decreases by two. Thus the total decrease in Φ caused by the r^{th} mistake made when predicting an entry in $B(j,z)$ is at least $2(r-1)$. ■

From Lemma 3.2, we see that the more entries known in $B(j,z)$, the greater the decrease in Φ for future mistakes on such entries. So, intuitively it appears that the adversary can maximize the number of mistakes made by the learner by balancing the number of entries seen in $B(j,z)$ for all j and z. We prove that this intuition is correct and apply it to obtain a lower bound on the amount Φ must have decreased after the learner has made μ mistakes.

Lemma 3.3 *After μ mistakes are made, the total decrease in Φ is at least $km\left(\frac{\mu}{km}-1\right)^2$.*

Proof: From Lemma 3.2, after the r^{th} made in predicting an entry from $B(j,z)$, the total decrease in Φ from its initial value is at least $\sum_{x=1}^{r} 2(x-1) \geq (r-1)^2$. Let $W(j,z)$ be the number of mistakes made in column j of rows of type z. The total decrease in Φ is at least

$$D = \sum_{j=1}^{m}\sum_{z=1}^{k}(W(j,z)-1)^2$$

subject to the constraint $\sum_{j=1}^{m}\sum_{z=1}^{k} W(j,z) = \mu$.

Using the method of Lagrange multipliers, we obtain that d is minimized when $W(j,z) = \frac{\mu}{km}$ for all j and z. So the total decrease in Φ is at least

$$\sum_{j=1}^{m}\sum_{z=1}^{k}\left(\frac{\mu}{km}-1\right)^2 = km\left(\frac{\mu}{km}-1\right)^2.$$

■

We now complete the proof of the theorem. Combining Lemma 3.1 and Lemma 3.3 along with the observation that Φ is strictly non-increasing, we have shown that

$$km\left(\frac{\mu}{km}-1\right)^2 \leq \frac{k-1}{k}n^2.$$

This implies that $\mu \leq km + n\sqrt{(k-1)m}$. ∎

We note that by using the simpler argument that each mistake, except for the first mistake in each column of each row type, decreases Φ by at least 2, we obtain a $km + \frac{k-1}{2k}n^2$ mistake bound for *any* row-filter algorithm. Also, Manfred Warmuth [68] has independently given an algorithm, based on the weighted majority algorithm of Littlestone and Warmuth [44], that makes at most $O(km + n\sqrt{m \lg k})$ mistakes in the worst case. Warmuth's algorithm builds a complete graph of n vertices where row i corresponds to vertex v_i and all edges get an initial weight of 1. To predict a value for (i,j) the learner takes a weighted majority of all active neighbors of v_i (v_k is *active* if M_{kj} is known). After receiving feedback, the learner sets the weight on the edge from v_i to v_k to be 0 if $M_{kj} \neq M_{ij}$. Finally, if a mistake occurs the learner doubles the weight of (v_i, v_k) if $M_{kj} = M_{ij}$ (i.e. the edges to neighbors that predicted correctly). We note that this algorithm is not a row-filter algorithm. See Appendix A for details of this algorithm and its analysis.

Does *ConsMajorityPredict* give the best performance possible by a row-filter algorithm? We now present an information-theoretic lower bound on the number of mistakes an adversary can force against any row-filter algorithm.

Theorem 3.9 *Let p be a prime and let $m = (p^2 + p + 1)$. Any row-filter algorithm for learning a $2n \times m$ matrix with $m \geq n$ and $k \geq 2$ makes at least $n(p+1) = \Omega(n\sqrt{m})$ mistakes when the adversary selects the query sequence.*

Proof: We assume that the adversary knows the learner's algorithm and has access to any random bits he uses. (One can prove a similar lower bound on the expected mistake bound when the adversary cannot access the random bits.)

Our proof depends upon the existence of a *projective geometry* Γ on m points and lines [13]. That is, there exists a set of m points and a set of m lines such that each line contains exactly $p+1$ points and each point is at the intersection of exactly $p+1$ lines. Furthermore, any pair of lines intersects at exactly one point, and any two points define exactly one line. Figure 3-7 shows a matrix representation of such a geometry; an "x" in entry (i,j) indicates that point j is on line i. We use the first n lines of Γ.

X	X		X			
	X	X		X		
		X	X		X	
			X	X		X
X				X	X	
	X				X	X
X		X				X

Figure 3-7: A projective geometry for $p = 2$, $m = 7$.

The matrix M consists of two row types: the odd rows are filled with ones and the even rows with zeros. Two rows of M are assigned to each line of Γ. (See Figure 3-8). We now prove that the adversary can force a mistake for each entry of Γ. The adversary's query sequence maintains the condition that an entry (i, j) is not revealed unless line $\lceil i/2 \rceil$ of Γ contains point j. In particular, the adversary will begin by presenting one entry of the matrix for each entry of Γ. We prove that for each entry of Γ the learner must predict the same value for the two corresponding entries of the matrix. Thus the adversary forces a mistake for the $n(p+1) = \Omega(n\sqrt{m})$ entries of Γ. The remaining entries of the matrix are then presented in any order.

Let I be the set of rows that may be used by the row-filter algorithm when predicting entry $(2i, j)$. Let I' be the set of rows that may be used by the row-filter algorithm when predicting entry $(2i-1, j)$. We prove by contradiction that $I = I'$. If $I \neq I'$ then it must be the case that there is some row r that is defined in column j and consistent with row $2i$, yet inconsistent with row $2i - 1$. By definition of the adversary's query sequence it must be the case that lines $\lceil r/2 \rceil$ and $\lceil (2i-1)/2 \rceil = i$ of Γ contain point j. Furthermore, since $(2i - 1, j)$ is being queried, that entry is not known. Thus rows r and $2i - 1$ must both be known in some other column j' since they are known to be inconsistent. Thus since only entries in Γ are shown, it follows that lines $\lceil r/2 \rceil$ and i of Γ also contain point j' for $j' \neq j$. So, this implies that lines $\lceil r/2 \rceil$ and i of Γ must intersect at two points

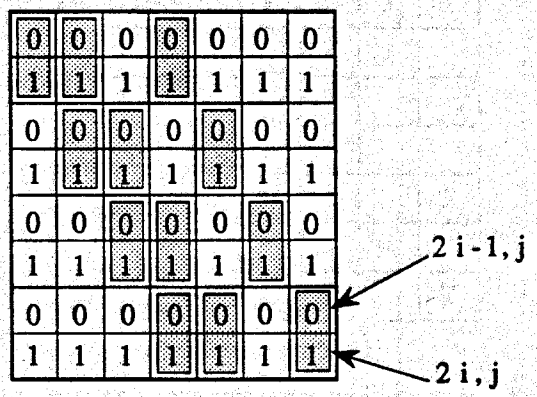


Figure 3-8: The matrix created by the adversary in the proof of Theorem 3.9. The shaded regions correspond to the entries in the first n lines of Γ. The learner is forced to make a mistake on one of the entries in each shaded rectangle.

giving a contradiction. Thus $I = I'$ and so $f(I,j) = f(I',j)$ for entry $(2i,j)$ and entry $(2i-1,j)$. Since the rows differ in each column the adversary can force a mistake on one of these entries. Since the adversary has access to the random bits of the learner, he can compute $f(I,j)$ just before making his query, and ask the learner to predict the entry for which the mistake will be made. This procedure is repeated for the pair of entries corresponding to each element of Γ. ∎

We use a similar argument to get an $\Omega(m\sqrt{n})$ bound for $m < n$. Combined with the lower bound of Theorem 3.5 and Theorem 3.9 we obtain a $\Omega(km + (n-k)\lfloor \lg k \rfloor + \min\{n\sqrt{m}, m\sqrt{n}\})$ lower bound on the number of mistakes made by a row-filter algorithm.

Corollary 3.2 *Any row-filter algorithm makes* $\Omega(km+(n-k)\lfloor \lg k \rfloor + \min\{n\sqrt{m}, m\sqrt{n}\})$ *mistakes against an adversary-selected query sequence.*

Comparing this lower bound to the upper bound proven for *ConsMajorityPredict*, we see that for fixed k the mistake bound of *ConsMajorityPredict* is within a constant factor of optimal.

Given this lower bound, one may question the $2m+n-2$ upper bound for $k = 2$ given

in Theorem 3.6. However, the algorithm described is not a row-filter algorithm. Also compared to our results for the learner-selected query sequence, it appears that allowing the learner to select the query sequence is quite helpful.

3.6 Randomly-directed Learning

In this section we consider the case that the learner is presented at each step with one of the remaining entries of the matrix selected uniformly and independently at random. We present a prediction algorithm that makes $O(km + nk\sqrt{H})$ mistakes on average where H is the maximum Hamming distance between any two rows of the matrix. (The Hamming distance between two rows is the number of entries on which they disagree.) We note that when $H = \Omega(\frac{m}{k})$ the result of Theorem 3.8 supersedes this result. A key result of this section is a proof relating two different probabilistic models for analyzing the mistake bounds under a random presentation. We first consider a simple probabilistic model in which the requirement that t matrix entries are known is simulated by assuming that each entry of the matrix is seen independently with probability $\frac{t}{nm}$. We then prove that any upper bound obtained on the number of mistakes under this simple probabilistic model holds under the true model (to within a constant factor) in which we have exactly t entries known. This result is extremely useful since in the true model the dependencies among the probabilities that matrix entries are known makes the analysis significantly more difficult.

We define the algorithm *RandomConsistentPredict* to be the row-filter algorithm where the learner makes his prediction for M_{ij} by choosing one row i' of I uniformly at random and predicting the value $M_{i'j}$. (If I is empty then *RandomConsistentPredict* makes a random guess.)

Theorem 3.10 *Let H be the maximum Hamming distance between any two rows of M. Then the expected number of mistakes made by RandomConsistentPredict is $O(k(n\sqrt{H} + m))$.*

Proof: Let U_t be the probability that the prediction rule makes a mistake on the $(t+1)$st step. That is, U_t is the chance that a prediction error occurs on the next randomly selected entry given that exactly t other randomly chosen entries are already known. Clearly, the expected number of mistakes is $\sum_{t=0}^{S-1} U_t$, where $S = nm$. Our goal is to find an upper bound for this sum.

The condition that exactly t entries are known makes the computation of U_t rather messy since the probability of having seen some entry of the matrix is *not* independent of knowing the others. Instead, we compute the probability V_t of a mistake under the simpler assumption that each entry of the matrix has been seen with probability t/S, independent of the rest of the matrix. We first compute an upper bound for the sum $\sum_{t=0}^{S-1} V_t$, and then show that this sum is within a constant factor of $\sum_{t=0}^{S-1} U_t$.

Lemma 3.4 $\sum_{t=0}^{S-1} V_t = O(km + nk\sqrt{H})$.

Proof: Fix t, and let $p = t/S$. Also, let $d(i)$ be the number of rows of the same type as row i.

By definition, V_t is the probability of a mistake occurring when a randomly selected unknown entry is presented, given that all other entries are known with probability p. Since each entry (i,j) is presented next with probability $1/S$, it follows that

$$V_t = \frac{1}{S} \sum_{i,j} R_{ij}$$

where R_{ij} is the probability of a mistake occurring, given that entry (i,j) is unknown and presented next.

If $d(i) = 1$, then we use the trivial bound $R_{ij} \leq 1$.

Otherwise, let I_{ij} be the random variable describing the set of rows consistent with row i and known in column j. Then R_{ij} is the probability that either I_{ij} is empty, or that I_{ij} is not empty and a randomly selected row i' from I_{ij} is such that $M_{ij} \neq M_{i'j}$.

The chance that $I_{ij} = \emptyset$ is at most the chance that I_{ij} contains no row of the same type as i. This latter probability is just the chance that none of the $d(i) - 1$ rows of the same type as i have been seen in column j, which is easily computed to be $(1-p)^{d(i)-1}$.

If $I_{ij} \neq \emptyset$, then a row from I_{ij} is randomly selected. Let ξ be a random variable describing the uniformly random choice of one of the n rows of the matrix. (Note that ξ is chosen from among *all* n rows.) Then the chance of error is

$$\begin{aligned}\Pr[M_{\xi j} \neq M_{ij} \mid \xi \in I_{ij}] &= \frac{\Pr[M_{\xi j} \neq M_{ij} \wedge \xi \in I_{ij}]}{\Pr[\xi \in I_{ij}]} \\ &= \frac{\sum_{i' \neq i, M_{i'j} \neq M_{ij}} \Pr[i' \in I_{ij}]}{\sum_{i' \neq i} \Pr[i' \in I_{ij}]}\end{aligned}$$

Note that if i and i' are the same type then $\Pr[i' \in I_{ij}] = p$, the chance that $(i'j)$ is known. Thus, the denominator is lower bounded by $p(d(i)-1)$.

If $M_{ij} \neq M_{i'j}$, then $\Pr[i' \in I_{ij}]$ is the chance that (i', j) is known and that i and i' are consistent. Entry (i', j) is known with probability p, and i and i' are consistent if either (i, j') or (i', j') is unknown for each column $j' \neq j$ in which i and i' differ. If $h(i, i')$ is the Hamming distance between the rows, then this probability is computed to be $(1-p^2)^{h(i,i')-1}$.

Combining these facts, we have:

$$\begin{aligned}V_t &\leq \frac{1}{S}\sum_i\sum_j (1-p)^{d(i)-1} + \frac{1}{S}\sum_{d(i)>1}\sum_j \frac{\sum_{i' \neq i, M_{i'j} \neq M_{ij}} p(1-p^2)^{h(i,i')-1}}{p(d(i)-1)} \\ &= \frac{1}{n}\sum_i (1-p)^{d(i)-1} + \frac{1}{S}\sum_{d(i)>1}\sum_{i' \neq i} \frac{h(i,i')(1-p^2)^{h(i,i')-1}}{d(i)-1}\end{aligned}$$

Recall that our goal is to upper bound the sum $\sum_{t=0}^{S-1} V_t$. Applying the above upper bound for V_t we get

$$\sum_{t=0}^{S-1} V_t \leq \sum_{t=0}^{S-1}\left(\frac{1}{n}\sum_i (1-(t/S))^{d(i)-1}\right) + \sum_{t=0}^{S-1}\left(\frac{1}{S}\sum_{d(i)>1}\sum_{i' \neq i}\frac{h(i,i')}{d(i)-1}(1-(t/S)^2)^{h(i,i')-1}\right). \tag{3.1}$$

We now evaluate the first part of the above expression. We begin by noting that

$$\sum_{t=0}^{S-1}\left(\frac{1}{n}\sum_{i=1}^{n}(1-(t/S))^{d(i)-1}\right) \leq \frac{1}{n}\sum_{i=1}^{n}\left(1 + \int_0^S \left(1-\frac{t}{S}\right)^{d(i)-1} dt\right).$$

Since

$$\int_0^S \left(1-\frac{t}{S}\right)^{d(i)-1} dt = \int_0^S \left(\frac{S-t}{S}\right)^{d(i)-1} dt = \int_0^S \left(\frac{t}{S}\right)^{d(i)-1} dt = \frac{S}{d(i)},$$

we see that

$$\frac{1}{n}\sum_{i=1}^{n}\left(1+\int_0^S\left(1-\frac{t}{S}\right)^{d(i)-1}dt\right) = \frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{S}{d(i)}\right)$$
$$= 1+m\sum_{i=1}^{n}\frac{1}{d(i)} = km+1. \qquad (3.2)$$

We obtain the last step of the equality by rewriting the summation to go over all the row types: there are $d(r)$ terms for rows of type r and thus each row type contributes 1 to the summation.

We are now ready to evaluate the second part of Expression 3.1. To complete the proof of the theorem we must show that

$$\sum_{t=0}^{S-1}\left(\frac{1}{S}\sum_{d(i)>1}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}(1-(t/S)^2)^{h(i,i')-1}\right) = O(nk\sqrt{H}).$$

We begin by deriving an upper bound for the second part of Expression (3.1) of

$$\frac{1}{S}\sum_{d(i)>1}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}\left(1+\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt\right).$$

If $h(i,i')=1$ then this integral is trivially evaluated to be S. Otherwise, applying the inequality $e^x \geq 1+x$ we get

$$\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt \leq \int_0^S \exp\left\{-\left(\frac{t}{S}\right)^2(h(i,i')-1)\right\}dt. \qquad (3.3)$$

A standard integral table [23] gives

$$\int_0^\infty \exp\left\{-\left(\frac{t}{S}\right)^2(h(i,i')-1)\right\}dt = \frac{S\sqrt{\pi}}{2\sqrt{h(i,i')-1}}. \qquad (3.4)$$

Combining these bounds, we have

$$\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt \leq \frac{S\sqrt{\pi}}{\sqrt{2h(i,i')}} \qquad (3.5)$$

for $h(i,i') \geq 1$. Therefore, the second part of Expression 3.1 can be upper bounded by

$$\frac{1}{S}\sum_{d(i)>1}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}\left(1+\int_0^S\left(1-(t/S)^2\right)^{h(i,i')-1}dt\right)$$

$$\leq \frac{1}{S}\sum_{i}\sum_{i'\neq i}\frac{2h(i,i')}{d(i)}\left(1+\frac{S\sqrt{\pi}}{\sqrt{2h(i,i')}}\right)$$
$$\leq \frac{2m}{S}\sum_{i}\sum_{i'\neq i}\frac{1}{d(i)}+\sqrt{2\pi}\sum_{i}\sum_{i'\neq i}\frac{\sqrt{h(i,i')}}{d(i)} = O(nk\sqrt{H})$$

This implies the desired bound. ∎

To complete the theorem, we prove the main result of this section, namely, that the upper bound obtained under this simple probabilistic model holds (to within a constant factor) for the true model. In other words, to compute an upper bound on the number of mistakes made by a prediction algorithm when the instances are selected according to a uniform distribution on the instance space, one can replace the requirement that exactly t matrix entries are known by the requirement that each matrix entry is known with probability $\frac{t}{nm}$.

Lemma 3.5 $\sum_{t=0}^{S-1} U_t = O\left(\sum_{t=0}^{S-1} V_t\right)$.

Proof: We first note that

$$V_t = \sum_{r=0}^{S-1}\binom{S}{r}\left(\frac{t}{S}\right)^r\left(1-\frac{t}{S}\right)^{S-r}U_r.$$

To see this, observe that for each r, where r is the number of known entries, we need just multiply U_r by the probability that exactly r entries are known assuming each entry is known with probability of t/S. Therefore,

$$\begin{aligned}\sum_{t=0}^{S-1} V_t &= \sum_{t=0}^{S-1}\sum_{r=0}^{S-1} U_r\binom{S}{r}\left(\frac{t}{S}\right)^r\left(1-\frac{t}{S}\right)^{S-r} \\ &= \sum_{r=0}^{S-1} U_r\left[\sum_{t=0}^{S-1}\binom{S}{r}\left(\frac{t}{S}\right)^r\left(1-\frac{t}{S}\right)^{S-r}\right].\end{aligned} \tag{3.6}$$

Thus, to prove the lemma, it suffices to show that the inner summation is bounded below by a positive constant. By symmetry, assume that $r \leq S/2$ and let $y = S - r$. Stirling's approximation implies that

$$\binom{S}{r} = \Theta\left(\frac{S^S}{r^r y^y}\sqrt{\frac{S}{ry}}\right).$$

Applying this formula to the desired summation we obtain that

$$\begin{aligned}\sum_{t=0}^{S}\binom{S}{r}\left(\frac{t}{S}\right)^r\left(1-\frac{t}{S}\right)^{S-r} &= \Theta\left(\sqrt{\frac{S}{ry}}\sum_{t=0}^{S}\left(\frac{t}{r}\right)^r\left(\frac{S-t}{y}\right)^y\right)\\ &= \Omega\left(\sqrt{\frac{S}{ry}}\sum_{x=1}^{\sqrt{ry/S}}\left(\frac{r+x}{r}\right)^r\left(\frac{y-x}{y}\right)^y\right).\end{aligned}$$

The last step above follows by letting $x = t-r$ and reducing the limits of the summation. To complete the proof that the inner summation of Equation 3.7 is bounded below by a positive constant we need just prove that

$$\left(\frac{r+x}{r}\right)^r\left(\frac{y-x}{y}\right)^y = \Omega(1)$$

for all $1 \le x \le \sqrt{ry/S}$

Using the inequality $1+x \le e^x$, it can be shown that for $1+y > 0$, $1+y \ge e^{\frac{y}{1+y}}$. We apply this observation to get that

$$\begin{aligned}\left(\frac{r+x}{r}\right)^r\left(\frac{y-x}{y}\right)^y &= \left(1+\frac{x}{r}\right)^r\left(1-\frac{x}{y}\right)^y\\ &\ge \exp\left\{\frac{x}{1+\frac{x}{r}}-\frac{x}{1-\frac{x}{y}}\right\} = \exp\left\{\frac{rx}{r+x}-\frac{yx}{y-x}\right\}\\ &= \exp\left\{\frac{-x^2(r+y)}{(r+x)(y-x)}\right\} = \exp\left\{\frac{-Sx^2}{(r+x)(y-x)}\right\}.\end{aligned}$$

Since $x \le \sqrt{ry/S}$, it follows that $Sx^2 \le ry$. Applying this observation to the above inequality it follows that

$$\begin{aligned}\left(\frac{r+x}{r}\right)^r\left(\frac{y-x}{y}\right)^y &\ge \exp\left\{\frac{-ry}{(r+x)(y-x)}\right\}\\ &= \exp\left\{\frac{-ry}{ry+(y-r)x-x^2}\right\}\\ &\ge \exp\left\{\frac{-ry}{ry-ry/S}\right\} = \exp\left\{\frac{-1}{1-\frac{1}{S}}\right\}.\end{aligned}$$

Finally we note that for $S \ge 2$, $e^{\frac{-1}{1-1/S}} \ge e^{-2}$. This completes the proof of the lemma. ■

Clearly Lemma 3.4 and Lemma 3.5 together imply that $\sum_{t=0}^{S-1} U_t = O(km + nk\sqrt{H})$, giving the desired result. ■

This completes our discussion of learning k-binary-relations.

Director	Lower Bound	Upper Bound	Notes
Learner	$\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$	$km + (n-k)\lfloor \lg k \rfloor$	
Teacher	$km + (n-k)(k-1)$	$km + (n-k)(k-1)$	
Adversary	$km + (n-k)\lfloor \lg k \rfloor$	$O(km + n\sqrt{m \lg k})$ †	
Adversary	$2m + n - 2$	$2m + n - 2$	$k = 2$
Adversary	$\Omega(km + (n-k)\lg k + \min\{n\sqrt{m}, m\sqrt{n}\})$	$km + n\sqrt{(k-1)m}$	row-filter algorithm
Random	$\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$	$O(km + nk\sqrt{H})$	avg. case, row-filter alg.

Table 3.1: Summary of our results for learning a k-binary-relation.

3.7 Conclusions and Open Problems

In this chapter we have studied the the problem of learning a binary relation between two sets of objects under the extended mistake-bound model. Our specific results are summarized in Table 3.1. In this table all lower bounds are information-theoretic bounds and all upper bounds are for polynomial-time learning algorithms.

From observing Table 3.1 one can see that several of the above bounds are tight and several others are asymptotically tight. However, there is a gap in the bound for the random and adversary (except $k \leq 2$) directors. Note that the bounds for row-filter algorithms are asymptotically tight for k constant. Clearly, if we want asymptotically tight bounds that include a dependence on k we must incorporate k into the projective geometry lower bound. (Currently, the relation created by the adversary has only two row types.)

Recall that the results of this chapter were motivated by the problem of learning an arbitrary binary relation. Since an unstructured relation is hopeless to learn, we forced some structure by providing the learner with prior knowledge that there are only k row types. As demonstrated by the allergist example, for some situations such structure is

†Due to Manfred Warmuth. See Appendix A for details on his algorithm and its analysis.

quite natural. However, as we see in the next chapter, there are other interesting ways in which a relation can be structured.

Chapter 4

Learning Total Orders

In this chapter we continue our study of learning binary relations under the extended mistake-bound model. Here we structure the relation by giving the learner prior knowledge that the relation is a total order. (For example the predicate may be "<".) One can view this problem as that of learning a total order on a set of n objects where an instance corresponds to comparing which of two objects is greater in the target total order. Thus this problem is like comparison-based sorting except for two key differences: we vary the agent selecting the order in which comparisons are made (in sorting the learner does the selection) and we charge the learner only for all *incorrectly predicted* comparisons.

Before describing our results, we motivate this section with the following example. There are n basketball teams that are competing in a round-robin tournament. That is, each team will play all other teams exactly once. Furthermore, we make the (admittedly simplistic) assumption that there is a ranking of the teams such that a team wins its match if and only if its opponent is ranked below it. The learner wants to place a \$10 bet on each game: if he bets on the winning team he wins \$10 and if he bets on the losing team he loses \$10. Of course, the goal of the learner is to win as many bets as possible.

We formalize the problem of learning a total order as follows. The instance space $X_n = \{1, \ldots, n\} \times \{1, \ldots, n\}$. An instance (i, j) in X_n is in the target concept if and

This chapter describes joint research with Ron Rivest [22].

only if object i precedes object j in the corresponding total order.

If computation time is not a concern, then the halving algorithm makes only $O(n \lg n)$ mistakes. However, we are interested in efficient algorithms and thus our goal is to design an efficient version of the halving algorithm. In the next section we discuss the relation between the halving algorithm and approximate counting. Then we show how to use an approximate counting scheme to efficiently implement a randomized version of the approximate halving algorithm, and apply this result to the problem of learning a total order on a set of n elements. Finally, we discuss how a majority algorithm can be used to implement a counting algorithm.

4.1 The Halving Algorithm and Approximate Counting

In this section we review the halving algorithm and approximate counting schemes. We first cover the halving algorithm [6, 43]. Let $\mathcal{V}$ denote the set of concepts in C_n that are consistent with the feedback from *all previous queries*. Given an instance x in X_n, for each concept in $\mathcal{V}$ the halving algorithm computes the prediction of that concept for x and predicts according to the majority. Finally, all concepts in $\mathcal{V}$ that are inconsistent with the correct classification are deleted. Littlestone [43] shows that this algorithm makes at most $\lg |C_n|$ mistakes. Now suppose the prediction algorithm predicts according to the majority of concepts in set $\mathcal{V}'$, the set of all concepts in C_n consistent with all *incorrectly* predicted instances. Littlestone [43] also proves that this *space-efficient halving algorithm* makes at most $\lg |C_n|$ mistakes. So for any prediction algorithm A that only remembers its mistakes, the number of instances stored by A is bounded by $\mathrm{MB}_Z(A, C_n)$.

We define an *approximate halving algorithm* to be the following generalization of the halving algorithm. Given instance x in X_n an approximate halving algorithm predicts in agreement with at least $\varphi|\mathcal{V}|$ of the concepts in $\mathcal{V}$ for some constant $0 < \varphi \leq 1/2$.

Theorem 4.1 *The approximate halving algorithm makes at most* $\log_{(1-\varphi)^{-1}} |C_n|$ *mistakes for learning* C_n.

Proof: Each time a mistake is made, the number of concepts that remain in $\mathcal{V}$ are reduced by a factor of at least $1-\varphi$. Thus after at most $\log_{(1-\varphi)^{-1}} |C_n|$ mistakes there is only one consistent concept left in C_n. ∎

We note that the above result holds also for the space-efficient version of the approximate halving algorithm.

When given an instance $x \in X_n$, one way to predict as dictated by the halving algorithm is to count the number of concepts in $\mathcal{V}$ for which $c(x) = 0$ and for which $c(x) = 1$ and then predict with the majority. As we shall see, by extending this idea we can implement the approximate halving algorithm using an approximate counting scheme.

We now introduce the notion of an approximate counting scheme for counting the number of elements in a finite set $\mathcal{S}$. Let x be a description of a set $\mathcal{S}_x$ in some natural encoding. An *exact counting scheme* on input x outputs $|\mathcal{S}_x|$ with probability 1. Such a scheme is polynomial-if it runs in time polynomial in $|x|$. Sometimes exact counting can be done in polynomial time; however, many counting problems are $\#\mathcal{P}$-complete and thus assumed to be intractable. (For a discussion of the class $\#\mathcal{P}$ see Valiant [64].) For many $\#\mathcal{P}$-complete problems good approximations are possible [33, 60, 63]. A *randomized approximation scheme*, R, for a counting problem satisfies the following condition for all $\epsilon, \delta > 0$:

$$\Pr\left[\frac{|\mathcal{S}_x|}{(1+\epsilon)} \leq R(x,\epsilon,\delta) \leq |\mathcal{S}_x|(1+\epsilon)\right] \geq 1-\delta$$

where $R(x,\epsilon,\delta)$ is R's estimate on input x, ϵ, and δ. In other words, with high probability, R estimates $|\mathcal{S}_x|$ within a factor of $1+\epsilon$. Such a scheme is *fully polynomial* if it runs in time polynomial in $|x|, \frac{1}{\epsilon}$, and $\lg \frac{1}{\delta}$. For further discussion see Sinclair [60].

We now review work on counting the number of linear extensions of a partial order. That is, given a partial order on a set of n elements, the goal is to compute the number of total orders that are linear extensions of the given partial order. We discuss the relation-

ship between this problem and that of computing the volume of a convex polyhedron. (For more details on this subject, see Section 2.4 of Lovász [45].) Given a convex set S and an element a of $\Re^n$, a ***weak separation oracle***

1. Asserts that $a \in S$, or

2. Asserts that $a \notin S$ and supplies a reason why. In particular for closed convex sets in $\Re^n$, if $a \notin S$ then there exists a hyperplane separating a from S. So if $a \notin S$, the oracle responds with such a separating hyperplane as the reason why $a \notin S$.

We now discuss how to reduce the problem of counting the number of extensions of a partial order on n elements to that of computing the volume of a convex n-dimensional polyhedron given by a separation oracle. The polyhedron built in the reduction will be a subset of the $[0,1]^n$ (i.e. the unit hypercube in $\Re^n$) where each dimension corresponds to one of the n elements. Observe that any inequality $x_i > x_j$ defines a halfspace in $[0,1]^n$. Let $\Delta(t)$ denote the polyhedron obtained by taking the *intersection* of the halfspaces given by the inequalities of the partial order t. (See Figure 4-1 for an example with $n = 3$.) For any pair of total orders t_1 and t_2, the polyhedra $\Delta(t_1)$ and $\Delta(t_2)$ are simplices that only intersect in a face (zero volume): a pair of elements say x_i and x_j that are ordered differently in t_1 and t_2 (such a pair must exist) define a hyperplane $x_i = x_j$ that separates $\Delta(t_1)$ and $\Delta(t_2)$. Let T_n be the set of all $n!$ total orders on n elements. Then

$$[0,1]^n = \bigcup_{t \in T_n} \Delta(t). \tag{4.1}$$

In other words, the union of the polyhedra associated with all total orders yields the unit hypercube. We have already seen that polyhedra associated with the $t \in T_n$ are disjoint. To see that they cover all of $[0,1]^n$ observe that any point $y \in [0,1]^n$ defines some total order t, and clearly $y \in \Delta(t)$. Let P be a partial order on a set of n elements. From equation (4.1) and the observation that the volumes of the polyhedra formed by each total order is equal, it follows that the volume of the polyhedron defined by any total

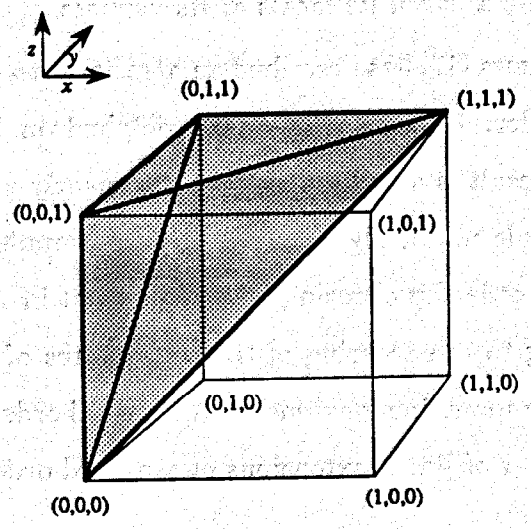


Figure 4-1: The polyhedron formed by the total order $z > y > x$.

order is $1/n!$. Thus it follows that for any partial order P

$$\frac{\text{number of extensions of } P}{n!} = \text{volume of } \Delta(P). \tag{4.2}$$

Rewriting equation (4.2), we obtain that

$$\text{number of extensions of } P = n! \cdot (\text{volume of } \Delta(P)). \tag{4.3}$$

Finally, we note that the weak separation oracle is easy to implement for any partial order. Given inputs a and S, it just checks each inequality of the partial order to see if a is in the convex polyhedron S. If a does not satisfy some inequality then reply that $a \notin S$ and return that inequality as the separating hyperplane. Otherwise, if a satisfies all inequalities, reply that $a \in S$.

Dyer, Frieze and Kannan [16] give a fully-polynomial randomized approximation scheme (*fpras*) to approximate the volume of a polyhedron given a separation oracle. From equation (4.3) we see that this fpras for estimating the volume of a polyhedron can be easily applied to estimate the number of extensions of a partial order. We note

that Dyer and Frieze [17] proved that it is $\#\mathcal{P}$-hard to exactly compute the volume of a polyhedron given either by a list of its facets or its vertices.

Independently, Matthews [47] has described an algorithm to generate a random linear extension of a partial order. Consider the convex polyhedron K defined by the partial order. Matthew's main result is a technique to sample nearly uniformly from K. Given such a procedure to sample uniformly from K, one can sample uniformly from the set of extensions of a partial order by choosing a random point in K and then selecting the total order corresponding to the ordering of the coordinates of the selected point. The procedure to generate a random linear extension of a partial order is then used repeatedly to approximate the number of linear extensions of a partial order.

4.2 Application to Learning

We begin this section by studying the problem of learning a total order under teacher-directed and self-directed learning. Then we show how to use a fpras to implement a randomized version of the approximate halving algorithm, and apply this result for the problem of learning a total order on a set of n elements.

Under the teacher-selected query sequence we obtain an $n-1$ mistake bound. The teacher can uniquely specify the target total order by giving the $n-1$ instances that correspond to consecutive elements in the target total order. Since $n-1$ instances are needed to uniquely specify a total order, we get a matching lower bound. Winkler [69] has shown that under the learner-selected query sequence , one can also obtain an $n-1$ mistake bound. To achieve this bound the learner uses an insertion sort, as described for instance by Cormen, Leiserson, and Rivest [15], where for each new element the learner guesses it is smaller than each of the ordered elements (starting with the largest) until a mistake is made. When a mistake occurs this new element is properly positioned in the chain. Thus at most $n-1$ mistakes will be made by the learner. Furthermore, the learner can be forced to make at least $n-1$ mistakes. The adversary gives feedback using the

following simple strategy: the first time an object is involved in a comparison, reply that the learner's prediction is wrong. In doing so, the adversary creates a set of *chains* where a chain is a total order on a subset of the elements. If c chains are created by this process then the learner has made $n-c$ mistakes. Since all these chains must be combined to get a total order, the adversary can force $c-1$ additional mistakes by always replying that a mistake occurs the first time that elements from two different chains are compared. (It is not hard to see that the above steps can be interleaved.) Thus the adversary can force $n-1$ mistakes.

Next we consider the case that an adversary selects the query sequence . We first prove an $\Omega(n \lg n)$ lower bound on the number of mistakes made by any prediction algorithm. We use the following result of Kahn and Saks [34]: Given any partial order P that is not a total order there exists an incomparable pair of elements x_i,x_j such that

$$\frac{3}{11} \leq \frac{\text{number of extensions of } P \text{ with } x_i \leq x_j}{\text{number of extensions of } P} \leq \frac{8}{11}.$$

So the adversary can always pick a pair of elements so that regardless of the learner's prediction, the adversary can force a mistake while only eliminating a constant fraction of the remaining total orders.

Finally, we present a polynomial prediction algorithm making $n \lg n + (\lg e) \lg n$ mistakes with very high probability. We first show how to use an exact counting algorithm R, for counting the number of concepts in C_n consistent with a given set of examples, to implement the halving algorithm.

Lemma 4.1 *Given a polynomial-time algorithm R to exactly count the number of concepts in C consistent with a given set E of examples, one can efficiently implement the halving algorithm for C.*

Proof: We show how to use R to efficiently make the predictions required by the halving algorithm. To make a prediction for an instance x in X_n the following procedure is used: Construct E^- from E by appending x as a negative example to E. Use the counting algorithm R to count the number of concepts $C^- \in \mathcal{V}$ that are consistent with E^-. Next

construct E^+ from E by appending x as a positive example to E. As before, use R to count the number of concepts $C^+ \in \mathcal{V}$ that are consistent with E^+. Finally if $|C^-| \geq |C^+|$ then predict that x is a negative example; otherwise predict that x is a positive example.

Clearly a prediction is made in polynomial time, since it just requires calling R twice. It is also clear that each prediction is made according to the majority of concepts in $\mathcal{V}$. ■

We modify this basic technique to use a fpras instead of the exact counting algorithm to obtain an efficient implementation of a randomized version of the approximate halving algorithm.

Theorem 4.2 *Let R be a* fpras *for counting the number of concepts in C_n consistent with a given set E of examples. If $|X_n|$ is polynomial in n, one can produce a prediction algorithm that for any $\delta > 0$ runs in time polynomial in n and $\lg \frac{1}{\delta}$ and makes at most $\lg |C_n| \left(1 + \frac{\lg e}{n}\right)$ mistakes with probability at least $1 - \delta$.*

Proof: The prediction algorithm implements the procedure described in Lemma 4.1 with the exact counting algorithm replaced by the fpras $R(n, \frac{1}{n}, \frac{\delta}{2|X_n|})$. Consider the prediction for an instance $x \in X_n$. Let $\mathcal{V}$ be the set of concepts that are consistent with all previous instances. Let r^+ (respectively r^-) be the number of concepts in $\mathcal{V}$ for which x is a positive (negative) instance. Let $\hat{r}^+$ (respectively $\hat{r}^-$) be the estimate output by R for r^+ (r^-). Since R is a fpras, with probability at least $1 - \frac{\delta}{|X_n|}$

$$\frac{r^-}{1+\epsilon} \leq \hat{r}^- \leq (1+\epsilon) r^- \quad \text{and} \quad \frac{r^+}{1+\epsilon} \leq \hat{r}^+ \leq (1+\epsilon) r^+$$

where $\epsilon = 1/n$. Without loss of generality, assume that the algorithm predicts that x is a negative instance, and thus $\hat{r}^- \geq \hat{r}^+$. Combining the above inequalities and the observation that $r^- + r^+ = |\mathcal{V}|$, we obtain that $r^- \geq \frac{|\mathcal{V}|}{1+(1+\epsilon)^2}$.

We define an *appropriate* prediction to be a prediction that agrees with *at least* $\frac{|\mathcal{V}|}{1+(1+\epsilon)^2}$ of the concepts in $\mathcal{V}$. To analyze the mistake bound for this algorithm, suppose that each prediction is appropriate. For a single prediction to be appropriate, both calls to the fpras R must output a count that is within a factor of $1+\epsilon$ of the true count. So any

given prediction is appropriate with probability at least $1 - \frac{\delta}{|X_n|}$, and thus the probability that all predictions are appropriate is at least

$$1 - |X_n| \left(\frac{\delta}{|X_n|} \right) = 1 - \delta.$$

Clearly if all predictions are appropriate then the above procedure is in fact an implementation of the approximate halving algorithm with $\varphi = \frac{1}{1+(1+\epsilon)^2}$ and thus by Theorem 4.1 at most $\log_{(1-\varphi)^{-1}} |C_n|$ mistakes are made. Substituting ϵ with its value of $\frac{1}{n}$ and simplifying the expression we obtain that with probability at least $1 - \delta$,

$$\# \text{ mistakes} \leq \frac{\lg |C_n|}{\lg \frac{1}{1-\varphi}} = \frac{\lg |C_n|}{\lg \left(1 + \frac{n^2}{n^2+2n+1}\right)}. \tag{4.4}$$

Since $\frac{n^2}{n^2+2n+1} \geq 1 - \frac{2}{n}$,

$$\begin{aligned} \frac{1}{\lg \left(1 + \frac{n^2}{n^2+2n+1}\right)} &\leq \frac{1}{\lg \left(1 + 1 - \frac{2}{n}\right)} \\ &= \frac{1}{1 + \lg \left(1 - \frac{1}{n}\right)} \\ &= 1 - \frac{\lg \left(1 - \frac{1}{n}\right)}{1 + \lg \left(1 - \frac{1}{n}\right)} \end{aligned}$$

Applying the inequalities $\lg \left(1 - \frac{1}{n}\right) \geq \frac{-\lg e}{n-1}$ and $1 + \lg \left(1 - \frac{1}{n}\right) \leq 1 - \frac{\lg e}{n}$ it follows that

$$\begin{aligned} \frac{\lg \left(1 - \frac{1}{n}\right)}{1 + \lg \left(1 - \frac{1}{n}\right)} &\geq \frac{\frac{-\lg e}{n-1}}{1 - \frac{\lg e}{n}} \\ &= \frac{-\lg e}{n - 1 - \frac{n-1}{n} \lg e} \\ &\geq \frac{-\lg e}{n} \end{aligned}$$

Finally, applying these inequalities to equation (4.4) yields that

$$\# \text{ mistakes} \leq \frac{\lg |C_n|}{\lg \left(1 + \frac{n^2}{n^2+2n+1}\right)} \leq \lg |C_n| \left(1 + \frac{\lg e}{n}\right).$$

■

Note that we could modify the above proof by not requiring that all predictions be appropriate. In particular if we allow γ predictions not to be appropriate then we get a mistake bound of $\lg |C_n| \left(1 + \frac{\lg e}{n}\right) + \gamma$.

We now apply this result to obtain the main result of this section. Namely, we describe a randomized polynomial prediction algorithm for learning a total order in the case that the adversary selects the query sequence.

Theorem 4.3 *There exists a prediction algorithm A for learning total orders such that on input δ (for all $\delta > 0$), and for any query sequence provided by the adversary, A runs in time polynomial in n and $\lg \frac{1}{\delta}$ and makes at most $n \lg n + (\lg e) \lg n$ mistakes with probability at least $1 - \delta$.*

Proof Sketch: We apply the results of Theorem 4.2 using the fpras for counting the number of extensions of a partial order given independently by Dyer, Frieze and Kannan [16], and by Matthews [47]. We know that with probability at least $1-\delta$, the number of mistakes is at most $\lg |C_n| \left(1 + \frac{\lg e}{n}\right)$. Since $|C_n| = n!$ the desired result is obtained. ■

We note that the probability that A makes more than $n \lg n + (\lg e) \lg n$ mistakes does not depend on the query sequence selected by the adversary. The probability is taken over the coin flips of the randomized approximation scheme.

Thus, as in learning a k-binary-relation using a row-filter algorithm, we see that a learner can do asymptotically better with self-directed learning versus adversary-directed learning. Furthermore, while the self-directed learning algorithm is deterministic, the adversary-directed algorithm is randomized. To accommodate such randomized prediction algorithms in our extended mistake-bound model we provide the learner with an input δ and allow the algorithm to exceed the "worst-case" mistake bound with a probability δ.

4.3 Majority Algorithms vs Counting Algorithms

In the last section we saw how a counting algorithm could be used to implement the halving algorithm. In this section, we consider the conditions under which the halving algorithm can be used to implement a counting algorithm. That is, we further explore the relationship between counting schemes and the halving algorithm.

Let $\mathcal{W}$ be a set of elements for which some subset $\mathcal{S}$ of the elements are distinguished. We use the function g that maps an element of $\mathcal{W}$ to $\{0,1\}$ to represent which of the elements of $\mathcal{W}$ are distinguished. Specifically, for $w \in \mathcal{W}$, $g(w) = 1$ if and only if w is a distinguished element. Let Σ^* be an alphabet used to describe some subset of $\mathcal{W}$. Let f be a function from $\Sigma^* \rightarrow 2^{\mathcal{W}}$ that maps $\sigma \in \Sigma^*$ to the subset of $\mathcal{W}$ that it describes. We denote $f(\sigma)$ by $\mathcal{V}_\sigma$. Let $t_\sigma = |\mathcal{V}_\sigma|$, and let d_σ be the number of distinguished elements in $\mathcal{V}_\sigma$. Formally, we have $d_\sigma = |\{w \in \mathcal{V}_\sigma : g(w) = 1\}|$. Finally, $u_\sigma = |\mathcal{V}_\sigma| - d_\sigma$. That is, u_σ is the number of undistinguished elements in $\mathcal{V}_\sigma$. So $d_\sigma + u_\sigma = t_\sigma$. Throughout this section we assume that there is a procedure SIZE(σ) that on input σ returns t_σ in unit time.

A *majority algorithm* takes as input σ and outputs a bit that is 1 if $d_\sigma \geq u_\sigma$, and 0 if $d_\sigma < u_\sigma$. That is,

$$\text{MAJORITY}(\sigma) = \begin{cases} 1 & \text{if } d_\sigma \geq u_\sigma \\ 0 & \text{if } d_\sigma < u_\sigma \end{cases}$$

Thus a majority algorithm solves a decision problem. On the other hand, a *counting algorithm* must output d_σ.

In Lemma 4.1 we used a counting algorithm to implement a majority algorithm. (There $\mathcal{W} = C_n$ and an element of $\mathcal{W}$ is distinguished if and only if it is consistent with the given set of examples.) In this section we discuss how a majority algorithm can be used to implement a counting algorithm. The results of this section are preliminary. Although we describe two techniques to convert a majority algorithm to a counting algorithm, we do not have applications for these techniques that yield any previously unknown results. We apply the first technique to an example problem; however, it is

trivial to construct a counting algorithm for this problem.

4.3.1 First Approach

In this section we describe our first approach for using a majority algorithm to implement a counting algorithm. We then show that our algorithm can be used to convert an algorithm that determines whether a majority of k-CNF formulas classifies a given instance as positive to one that computes the number of k-CNF formulas that classify the given instance as positive.

We now describe our first approach. Here we recursively apply MAJORITY, at each step reducing the larger of d_σ and u_σ by a factor of at least two. To use this approach, in addition to the MAJORITY oracle, a CONSTRAIN oracle must be provided. The specification for the CONSTRAIN oracle is as follows:

$$\text{CONSTRAIN}(\sigma) = \begin{cases} \text{new–object}(d_\sigma - \lceil t_\sigma/2 \rceil, u_\sigma) & \text{if } d_\sigma \geq u_\sigma \text{ (i.e. MAJORITY}(\sigma) = 1) \\ \text{new–object}(d_\sigma, u_\sigma - \lceil t_\sigma/2 \rceil) & \text{if } d_\sigma < u_\sigma \text{ (i.e. MAJORITY}(\sigma) = 0) \end{cases}$$

where new–object(d, u) creates a word $\sigma \in \Sigma^*$ such that $d_\sigma = d$ and $u_\sigma = u$. So the oracle CONSTRAIN just reduces the larger of d_σ and u_σ by a factor of $\lceil t_\sigma/2 \rceil$. The result of the CONSTRAIN oracle is illustrated in Figure 4-2.

Theorem 4.4 *One can construct, from a* MAJORITY, SIZE *and* CONSTRAIN *oracle, a counting algorithm that on input σ uses at most $\lg(t_\sigma)$ calls to each oracle.*

Proof: We construct a recursive counting procedure that takes input σ, and uses CONSTRAIN to reduce the larger of d_σ and u_σ. The initial call should be *Exact–Count1*(σ).

Exact–Count1(σ)
1 **if** SIZE$(\sigma) = 0$
2 **then return** 0
3 **else return** $\lceil t_\sigma/2 \rceil$MAJORITY(σ) + *Exact–Count1*(CONSTRAIN(σ))

We first argue that this procedure is correct. Each time CONSTRAIN is called, we know that d_σ is reduced by exactly $\lceil t_\sigma/2 \rceil$MAJORITY(σ). So the total decrease in d_σ from

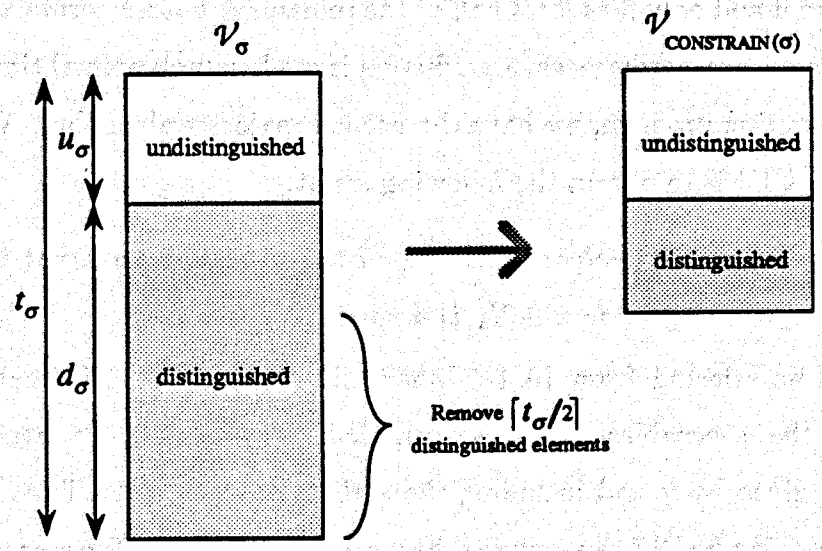


Figure 4-2: An illustration of the CONSTRAIN oracle when the majority of the elements in $\mathcal{V}_\sigma$ are distinguished.

its original value is accumulated in line 3 of the procedure. Finally, when $t_\sigma = 0$ then clearly $d_\sigma = 0$; thus the base case is correct.

Since at each step t_σ is reduced to $\lfloor t_\sigma/2 \rfloor$, at most $\lg(t_\sigma)$ recursive calls are made. Furthermore, MAJORITY, SIZE, and CONSTRAIN are called only once for each recursive call. ■

In terms of our original goal of converting a majority algorithm into a counting algorithm we have the following corollary of Theorem 4.4.

Corollary 4.1 *Let $\mathcal{M}$ be a majority algorithm for $\mathcal{W}$ using Σ^*. Let $T_{\mathcal{M}}(\sigma)$ be the running time of $\mathcal{M}$ on input σ. Furthermore, suppose that* CONSTRAIN *can be implemented in time $T_C(\sigma)$ on input σ. Then there exists an exact counting algorithm that runs in time at most $O((T_{\mathcal{M}}(\sigma) + T_C(\sigma)) \lg t_\sigma)$.*

We now apply this conversion to the following problem. Given a boolean vector $x = \{0,1\}^n$, compute the number of k-CNF formulas over n variables for which x is a positive example. As Angluin notes [4], Valiant's algorithm for PAC-learning k-CNF

implements the halving algorithm. That is, given an instance x, it replies that x is a positive instance if and only if at least half of the remaining k-CNF formulas classify x as a positive example. Furthermore, each prediction is made in polynomial time. Therefore, for the given counting problem, we have the needed majority algorithm. We now apply Corollary 4.1 to k-CNF to obtain the following result.

Lemma 4.2 *There exists a polynomial time algorithm to exactly count the number of k-CNF formulas for which some $x \in X_n$ is a positive instance.*

Proof: Let σ be selected from $\{0,1\}^n$ where the interpretation is that σ gives the assignments to the n variables. So $\mathcal{V}_\sigma$ is computed by evaluating the target formula on the assignment given by σ and including those that evaluate to 1. Thus Valiant's [66] algorithm for learning k-CNF can serve as the majority algorithm. We now prove that the CONSTRAIN oracle can also be implemented in polynomial time. As Angluin [4] notes, if Valiant's algorithm predicts 0, then there exists some clause τ in the learner's hypothesis that evaluates to 0. So by removing τ the requirements of CONSTRAIN are satisfied. ■

We note that this result is easily obtained without using Corollary 4.1. By studying the recursive structure of the counting algorithm, we obtain the following counting algorithm for k-CNF: Let T be the number of possible clauses of size k or less that are true for instance x. Since the target formula can contain any subset of these clauses (but none of the clauses that are negative for x), the number of k-CNF formulas that predict that x is a positive instance is 2^T.

4.3.2 Second Approach

In this section we describe our second approach for using a majority algorithm to implement a counting algorithm. To motivate this approach, we consider how the first approach might fail. The algorithm *Exact-Count1* can be used only if one can remove elements of $\mathcal{V}_\sigma$ in a controlled manner. However, it may be the case that one cannot remove elements from $\mathcal{V}_\sigma$ as desired, but can create some σ' such that $\mathcal{V}_{\sigma'} \supset \mathcal{V}_\sigma$ and furthermore, all elements of $\mathcal{V}_{\sigma'} - \mathcal{V}_\sigma$ are distinguished (or undistinguished). Our second

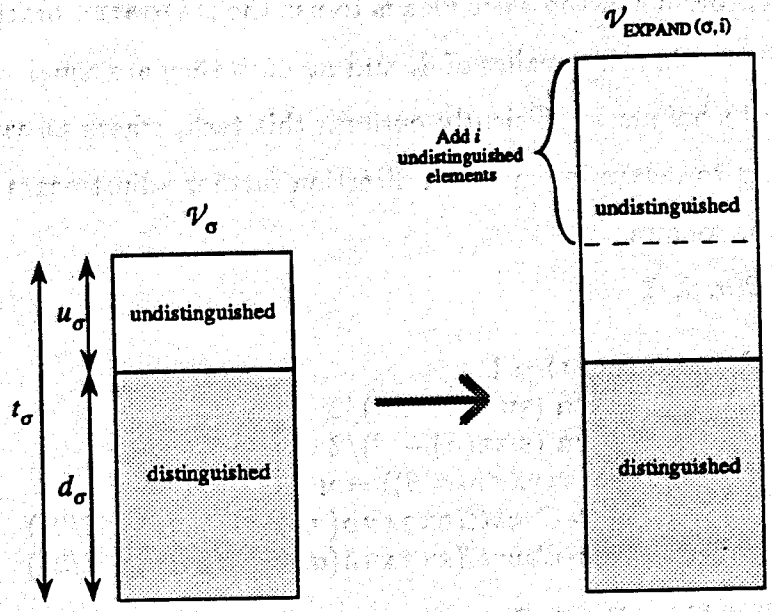


Figure 4-3: An illustration of the EXPAND oracle for the case that a majority of $\mathcal{V}_\sigma$ is distinguished.

approach is based on these ideas; instead of adjusting the size of d_σ and u_σ by reducing their sizes, we achieve the same effect by appropriately increasing the size of the smaller one.

To use this approach, in addition to the MAJORITY oracle, an EXPAND oracle must be provided instead of the CONSTRAIN oracle. The specification of the EXPAND oracle is as follows:

$$\text{EXPAND}(\sigma, i) = \begin{cases} \text{new-object}(d_\sigma, u_\sigma + i) & \text{if } d_\sigma \geq u_\sigma \\ \text{new-object}(d_\sigma + i, u_\sigma) & \text{if } d_\sigma < u_\sigma \end{cases}$$

So the oracle EXPAND just adds i elements to the smaller of d_σ and u_σ. The result of the EXPAND oracle is illustrated in Figure 4-3.

We now describe the details of our second approach to convert a majority algorithm to a counting algorithm.

Theorem 4.5 *One can construct from a* MAJORITY, SIZE, *and* EXPAND *oracle, a counting algorithm that on input σ uses at most $\lg(t_\sigma)$ calls to each oracle.*

Proof: As in Theorem 4.4, the basic idea is to use the MAJORITY oracle to compute d_σ. We do this by increasing the smaller of d_σ and u_σ until they are equal. As before, we use a binary search technique to efficiently perform this task, where at each step we use a call to MAJORITY to determine in which direction further adjustments should be made. The details are as follows.

Exact-Count2(σ, β, i)
1 **if** $i = 0$
2 **then if** MAJORITY$(\sigma) = 1$
3 **then return** $(\text{SIZE}(\sigma) + \beta)/2$
4 **else return** $(\text{SIZE}(\sigma) - \beta)/2$
5 **else if** MAJORITY(EXPAND$(\sigma, \beta)) =$ MAJORITY(σ)
6 **then** *Exact-Count2*(EXPAND$(\sigma, \beta), \beta + \lceil i/2 \rceil, \lfloor i/2 \rfloor)$
7 **else** *Exact-Count2*(EXPAND$(\sigma, \beta), \beta - \lceil i/2 \rceil, \lfloor i/2 \rfloor)$

The initial call should be *Exact-Count2* $(\sigma, 0, t_\sigma)$. The variable β is the current estimate of the number of elements that need to be added to the small side to make both sides equal, and i is the size of the next adjustment to be made to β.

We first argue that the procedure is correct. Let β_l, i_l denote the input values of β and i to the lth call of *Exact-Count2*. Let $\sigma_l = \text{EXPAND}(\sigma, \beta_l)$. Finally, we use d_l (respectively u_l) to denote d_{σ_l} (respectively u_{σ_l}). Without loss of generality, assume that $d_\sigma > u_\sigma$. So for all l,

- $d_l \doteq d_0 = d_\sigma$,
- $u_l = u_0 + \beta_l = u_\sigma + \beta_l$, and
- $i_l = \lfloor i_{l-1}/2 \rfloor$.

We claim that when the input $i = 0$ in *Exact-Count2*, $d_l = u_l$. We prove this using the following lemma.

Lemma 4.3 *For all l, $|d_{\sigma_l} - u_{\sigma_l}| \leq i_l$.*

Proof: We use an inductive proof on l. Clearly, the base case, $|d_\sigma - u_\sigma| \leq t_\sigma$, holds.

We now prove the inductive step. Assume inductively that $|d_l - u_l| = |d_\sigma - u_\sigma - \beta_l| \leq i_l$. Our goal is to show that

$$|d_\sigma - u_\sigma - \beta_{l+1}| \leq i_{l+1} = \lfloor i_l/2 \rfloor.$$

We separately consider when $d_l > u_l$, $d_l < u_l$, and $d_l = u_l$.

- **Case 1:** $d_l > u_l$. In this case, $\beta_{l+1} = \beta_l + \lceil i_l/2 \rceil$. So we must show that

$$-\lfloor i_l/2 \rfloor \leq d_\sigma - u_\sigma - \beta_l - \lceil i_l/2 \rceil \leq \lfloor i_l/2 \rfloor.$$

  That is, we must show that:

$$\lceil i_l/2 \rceil - \lfloor i_l/2 \rfloor \leq d_\sigma - u_\sigma - \beta_l \leq \lfloor i_l/2 \rfloor + \lceil i_l/2 \rceil = i_l.$$

  The inequality $d_\sigma - u_\sigma - \beta_l \leq i_l$ follows immediately from the inductive hypothesis. For the other inequality note that $\lceil i_l/2 \rceil - \lfloor i_l/2 \rfloor \leq 1$. Furthermore, since $d_l > u_l$, $d_\sigma - u_\sigma - \beta_l = d_l - u_l > 1$.

- **Case 2:** $d_l < u_l$. In this case, $\beta_{l+1} = \beta_l - \lceil i_l/2 \rceil$. So we must show that

$$-\lfloor i_l/2 \rfloor \leq d_\sigma - u_\sigma + \beta_l - \lceil i_l/2 \rceil \leq \lfloor i_l/2 \rfloor.$$

  The proof for these inequalities is similar to that of Case 1.

- **Case 3:** $d_l = u_l$. For this case we first use an inductive proof to show that $d_l + u_l \equiv i_l \pmod 2$. The base case follows from the fact that $i_0 = d_0 + u_0$. For the inductive step, observe that $i_l - i_{l+1} = |(d_l + u_l) - (d_{l+1} + u_{l+1})|$. Since $d_l = u_l$, it must be that i_l divisible by two, and thus $\lfloor i_l/2 \rfloor = \lceil i_l/2 \rceil$. The inductive hypothesis immediately follows from here.

This completes the proof of the lemma. ∎

Since the recursion continues until $i_l = 0$ it follows from Lemma 4.3 that when $i = 0$ in *Exact-Count2*, $d_l = u_l$. Thus we know that for β the final value of β_l, we have that $d_\sigma = u_\sigma + \beta$. Applying the equality that $d_\sigma + u_\sigma = t_\sigma$ and solving for d_σ, we see that

Director	Lower Bound	Upper Bound	Notes
Teacher	$n-1$	$n-1$	
Learner	$n-1$	$n-1$ *	
Adversary	$\Omega(n \lg n)$	$n \lg n + (\lg e) \lg n$	randomized algorithm

Table 4.1: Summary of our results for learning a total order.

$d_\sigma = (t_\sigma + \beta)/2$. (Since $d_l = u_l$, it follows that $t_\sigma + \beta$ is even.) The case that $d_\sigma < u_\sigma$ is handled similarly. Thus we have shown that the output from *Exact–Count2* is correct.

Since at each step the increment size i is reduced to $\lfloor i/2 \rfloor$, at most $\lg(t_\sigma)$ recursive calls are made. Furthermore each oracle is called at most once during each recursive call. ■

In terms of our original goal of converting a majority algorithm into a counting algorithm we have the following corollary of Theorem 4.5.

Corollary 4.2 *Let $\mathcal{M}$ be a majority algorithm for $\mathcal{W}$ under Σ^*. Let $T_{\mathcal{M}}(\sigma)$ be the running time of $\mathcal{M}$ on input σ. Furthermore, suppose that* EXPAND *can be implemented in time $T_C(\sigma)$ on input of size at most t_σ. Then there exists an exact counting algorithm that runs in time at most $O((T_{\mathcal{M}}(\sigma) + T_C(\sigma)) \lg t_\sigma)$.*

4.4 Conclusions and Open Problems

We have studied the the problem of learning a binary relation between a set and itself under the extended mistake-bound model. We have presented general techniques to help develop efficient versions of the halving algorithm. In particular, we have shown how a fpras can be used to efficiently implement a randomized version of the approximate halving algorithm. Our specific results are summarized in Table 4.1. In this table all lower bounds are information theoretic and all upper bounds are for polynomial-time learn-

*Due to Peter Winkler.

ing algorithms. Also the results listed are for deterministic algorithms unless otherwise stated.

From observing Table 4.1 one can see that all the above bounds are tight or asymptotically tight. Although the fpras for approximating the number of extensions of a partial order is a polynomial-time algorithm, the exponent on n is somewhat large and the algorithm is quite complicated. Thus an interesting problem is to find a "practical" prediction algorithm for the problem of learning a total order. Another interesting direction of research is to explore other ways of modeling the structure in a binary relation. Finally, we hope to find other applications of fully polynomial randomized approximation schemes to computational learning theory.

Chapter 5

The Power of Teacher-directed and Self-directed Learning

Not surprisingly, in studying the problems of learning binary relations and total orders, we saw that the complexity of the learning task depends greatly on the director (i.e. the agent selecting the presentation order of the instances). Thus, to study these problems we used the extended mistake-bound model introduced in Chapter 2. For the more typical domain of concept learning, in all previous work using an on-line model with the absolute mistake-bound success criteria, an adversary has been the director.

In this chapter we address some of the questions raised by the extended mistake-bound model. Namely, in an on-line model, what is the power of a teacher or the learner selecting the instances when the learner is judged by the number of mistakes he makes? For teacher-directed learning we consider the worst-case mistake bound of any learner that predicts according to some concept that agrees with all previously seen instances. Informally, we say the *teaching dimension* of a concept class is the minimum number of instances a teacher must reveal to uniquely identify *any* target concept chosen from the class.

We show that this new dimension measure is fundamentally different from the Vapnik-

This chapter describes joint research with Mike Kearns.

Chervonenkis dimension [10, 40, 67] and the dimension measure of Natarajan [51]. We also describe a straightforward relation between the teaching dimension and the number of membership queries needed for exact identification. Next, we give tight bounds on the teaching dimension for the class of monomials and then extend this result to more complicated concept classes such as monotone k-term DNF and k-term μ-DNF. We also compute bounds on the teaching dimension for the class of monotone rank-1 decision trees and orthogonal rectangles in $\{0,1,\cdots,n-1\}^d$. Finally, we build a concept class C_n for which the teaching dimension is $|C_n|-1$ and describe concept classes for which a significantly better bound is possible. As an example, for concept classes closed under exclusive-or, we prove that the teaching dimension is at most logarithmic in the size of the concept class.

In the second part of this chapter we study self-directed learning when it is applied to the domain of concept learning. That is, when the learner chooses the instance sequence, how many incorrect predictions are made before the target concept is uniquely specified? We give tight bounds on this measure for the class of monomials, k-term DNF formula, and orthogonal rectangles in $\{0,1,\cdots,n-1\}^d$. These bounds demonstrate that the number of mistakes can be asymptotically less than the number of queries. We then show that for concept classes for which the frontier of the rule space in Mitchell's version space algorithm [49] is always of size 1, the learner can make at most a single mistake. Finally we explore the following interesting paradox raised by our results: for many concept classes, the number of mistakes made with teacher-directed learning may be worse that the number of mistakes made with self-directed learning.

5.1 The Teaching Dimension

In this section, we study the power of having a helpful teacher as the director in the extended mistake-bound model. We begin by formally defining the *teaching dimension*. Let an *instance sequence* denote a sequence of *unlabeled* instances, and an *example se-*

quence denote a sequence of *labeled instances*. For concept class C_n and target concept $c \in C_n$, we define a *teaching sequence* T as an example sequence for C_n that uniquely specifies c. That is, all concepts in C_n except for c disagree with the classification of some instance in T. We define the *teaching dimension* $\mathrm{TD}(C_n)$ of a concept class C_n as follows:

$$\mathrm{TD}(C_n) = \max_{c \in C_n} \left(\min_{\tau \in T(c)} |\tau| \right)$$

where $T(c)$ is the set of all teaching sequences for c. In other words, the teaching dimension of a concept class is the minimum number of examples a teacher must reveal to uniquely identify *any* concept in the class.

Observe that the teaching dimension of a concept class is equal to the optimal mistake bound under the helpful-teacher director of the extended mistake-bound model. To see this correspondence, recall that in teacher-directed learning, we consider the worst-case mistake bound over all consistent learners. Clearly $\mathrm{TD}(C_n)$ is a lower bound for the optimal mistake bound: unless the unique target concept has been identified a mistake could be made on the next prediction. Furthermore, $\mathrm{TD}(C_n)$ is an upper bound for the optimal mistake bound since no mistake could be made after the target concept is uniquely identified.

In the next section we compare the teaching dimension to other dimension measures that have been used to characterize learnable concept classes. Then we compute the teaching dimension for several well-studied concept classes, and give general results that help to compute the teaching dimension for other concept classes.

5.1.1 Comparison to Other Dimension Measures

In this section we compare the teaching dimension to two other dimension measures that have been used to describe the complexity of concept classes.

concept	instances							
c_{target}	+	+	+	+	⋯	+	+	+
c_1	−	+	+	+	⋯	+	+	+
c_2	+	−	+	+	⋯	+	+	+
c_3	+	+	−	+	⋯	+	+	+
⋮								
$c_{\lvert X_n\rvert-1}$	+	+	+	+	⋯	+	−	+
$c_{\lvert X_n\rvert}$	+	+	+	+	⋯	+	+	−

Figure 5-1: A concept class C_n for which $\text{TD}(C_n) = |C_n| - 1$.

Vapnik-Chervonenkis Dimension

We now show that the teaching dimension is fundamentally different from the well-studied Vapnik-Chervonenkis dimension. (See Section 2.3.1 for the definition of the Vapnik-Chervonenkis dimension.) As we discussed in Chapter 2 the VC-dimension has been very important in computational learning theory: The results of Blumer et al. [10] prove that the VC-dimension characterizes what is PAC-learnable (disregarding computation time) using static sampling.

We now compare the teaching dimension to the VC-dimension. We begin by showing that neither of these dimension measures dominates the other: in some cases the $\text{TD}(C_n) > \text{VCD}(C_n)$ and in other cases $\text{TD}(C_n) < \text{VCD}(C_n)$.

Lemma 5.1 *There is a concept class C_n for which $\text{TD}(C_n) = |C_n| - 1$ and $\text{VCD}(C_n) = 1$.*

Proof: Consider the concept class C_n in which the target concept c_{target} classifies all instances as positive. In addition, for each $x \in X_n$ there is a concept $c \in C_n$ such that c classifies all instances but x as positive. (See Figure 5-1.) No concept in C_n has two negative instances, so clearly no set of two points is shattered. Since any singleton set is shattered, $\text{VCD}(C_n) = 1$. Finally, since each instance distinguishes only one of the concepts $c_1, \ldots, c_{|X_n|}$ from c_{target}, the $\text{TD}(C_n) = |C_n| - 1$. ■

	x_0	x_1	x_2	$\cdots$	x_{n-2}	x_{n-1}	x_n	x_{n+1}	$\cdots$	$x_{n+\lg n-1}$
c_0	+	−	−	$\cdots$	−	−	−	−	$\cdots$	−
c_1	−	+	−	$\cdots$	−	−	+	−	$\cdots$	−
c_2	−	−	+	$\cdots$	−	−	−	+	$\cdots$	−
$\vdots$										
c_{n-2}	−	−	−	$\cdots$	+	−	+	+	$\cdots$	−
c_{n-1}	−	−	−	$\cdots$	−	+	+	+	$\cdots$	+

Figure 5-2: A concept class C_n for which $\text{VCD}(C_n) > \text{TD}(C_n)$.

Thus the VC-dimension can be arbitrarily smaller than the teaching dimension. Furthermore, we note that the concept class used in the proof of Lemma 5.1 has the largest possible teaching dimension.

Observation 5.1 *For any concept class C, $\text{TD}(C) \leq |C| - 1$.*

Proof: Each concept in C must differ from all other concepts for at least one instance. Thus for each concept that is not the target concept there must be an example that it and the target concept classify differently. ■

We now show that the teaching dimension may be smaller than the VC-dimension.

Lemma 5.2 *There is a concept class C_n for which $\text{TD}(C_n) < \text{VCD}(C_n)$.*

Proof: Let $C_n = \{c_0, c_2, \ldots, c_{n-1}\}$ be a concept class over the instance space $X_n = \{x_0, x_1, \ldots, x_{n+\lg n-1}\}$ where $n = 2^k$ for some constant k. The concept c_i classifies instance x_i as positive and the rest of $x_0, \ldots, x_{n-1}$ as negative. The remaining $\lg n$ instances for c_i are classified according to the binary representation of i. (See Figure 5-2.) Clearly $\{x_n, \ldots, x_{n+\lg n-1}\}$ is a shattered set and thus $\text{VCD}(C_n) = \lg n = \lg |C_n|$. However, $\text{TD}(C_n) = 1$ since instance x_i uniquely defines concept c_i. ■

Since for all finite C_n, $\text{VCD}(C_n) \leq \lg |C_n|$ and $\text{TD}(C_n) \geq 1$ it follows that $\text{VCD}(C_n) \leq \lg |C_n| \cdot \text{TD}(C_n)$ and thus the concept class of Lemma 5.2 provides the maximum factor by which the VC-dimension can exceed the teaching dimension for a finite concept class.

Combined with Lemma 5.1 we have a complete characterization of how the teaching dimension relates to the VC-dimension.

We now uncover another key difference between the VC-dimension and the teaching dimension: the potential effect of removing a single concept from the concept class. Let C_n be a concept class with $\text{VCD}(C_n) = d$, and let $C'_n = C_n - \{f\}$ for some $f \in C_n$. Regardless of the choice of f, clearly $\text{VCD}(C'_n) \geq d - 1$. In contrast to this we have the following result.

Theorem 5.1 *Let C_n be a concept class for which $\text{TD}(C_n) \geq |C_n| - k$. Then there exists an $f \in C_n$ such that for $C'_n = C_n - \{f\}$, $\text{TD}(C'_n) \leq k$.*

Proof: Let f be a concept from C_n that requires a teaching sequence of length $\text{TD}(C_n)$. Let T be an *optimal* teaching sequence for f (i.e. $|T| = \text{TD}(C_n)$). We let $C'_n = C_n - \{f\}$, and prove that $\text{TD}(C'_n) \leq k$. To achieve this goal we must show that for each concept $c \in C'_n$ there exists a teaching sequence for c of length at most k. Without loss of generality, let $f' \in C'_n$ be the target concept that requires the longest teaching sequence. We now prove that there is a teaching sequence for f' of length at most k.

The intuition for the remainder of this proof is as follows. Since T is an optimal teaching sequence, for each instance $x_i \in T$ there must be a concept $f_i \in C'_n$ such that $f(x_i) \neq f_i(x_i)$. Furthermore, $f_i \neq f_j$ for $i \neq j$. However, it may be that a $x_i \in T$ distinguishes f from many concepts in C'_n. We say that all concepts in $C'_n - \{f_i\}$ that x_i distinguishes from f are eliminated as the possible target *"for free"*. Intuitively, since the teaching dimension is large, few concepts can be eliminated "for free". We then use this observation to show that $TD(C'_n)$ is small.

We now formalize this intuition. Let $x_* \in T$ be an instance that f and f' classify differently. (Since T is a teaching sequence such an x_* must exist.) We now define a set F of concepts that are distinguished from f "for free". For ease of exposition we shall also create a set S that will contain $C_n - F - \{f\}$.

We build the sets F and S as follows. Initially let $S = \{f'\}$ and let $F = \{c \in C'_n - \{f'\} \mid c(x_*) = f'(x_*)\}$. That is F initially contains the concepts that are also

distinguished from f by x_* and thus eliminated "for free". Then for each $x \in T - \{x_*\}$ of all concepts in $C'_n - T - S$ that disagree with f on x place one of these concepts (choose arbitrarily) in S and place the rest in F. Since T is a teaching sequence at the end of this process $|S| = \text{TD}(C_n)$ and $C_n = T \cup S \cup \{f\}$. Combining these observations with the assumption that $\text{TD}(C_n) \geq |C_n| - k$ we get that $|T| = |C_n| - \text{TD}(C_n) - 1 \leq k - 1$. That is, at most $k - 1$ concepts are eliminated "for free".

We now generate a teaching sequence for f' of length at most k. By the definition of F and S any concept in $C'_n - \{f'\}$ that classifies x_* as f' classifies it must be in F. That is, *all* concepts in S are distinguished from f' by x_*. Furthermore, since $|F| \leq k - 1$ at most $k - 1$ additional instances are needed to distinguish f' from the concepts in F. Finally since $C'_n = F \cup S$ it follows that there is a teaching sequence for f' of length at most $1 + (k - 1) = k$. This completes the proof of the theorem. ∎

So when $k = 1$, Theorem 5.1 implies that for any concept class C_n for which $\text{TD}(C_n) = |C_n| - 1$, there exists a concept whose removal causes the teaching dimension of the remaining class to be reduced to 1. We briefly mention an interesting consequence of this result. Although it appears that for a concept class C_n with $\text{VCD}(C_n) = |C_n| - 1$ there is little the teacher can do, this result suggest the following strategy: first teach some concept f' in $C_n - \{f\}$ and then (if possible) list the instances that f and f' classify differently.

While we have shown that the teaching dimension and the VC-dimension are fundamentally different, there are some relations between them. We now derive an upper bound for the teaching dimension that is based on the VC-dimension.

Theorem 5.2 *For any concept class C_n,*

$$\text{TD}(C_n) \leq \text{VCD}(C_n) + |C_n| - 2^{\text{VCD}(C_n)}.$$

Proof: Let $x_1, \ldots, x_d$ be a shattered set of size d for $d = \text{VCD}(C_n)$. By the definition of a shattered set, in an example sequence consisting of these d instances, all but one of a set of 2^d concepts are eliminated. Thus after placing $x_1, \ldots, x_d$ in the teaching sequence,

there are at most $|C_n| - 2^d + 1$ concepts remaining. Finally, we use the naive algorithm of Observation 5.1 to eliminate the remaining functions using at most $|C_n| - 2^d$ additional examples. ■

Natarajan's Dimension Measure

In this section we compare the teaching dimension to the following dimension measure defined by Natarajan [51] for concept classes of Boolean functions:

$$\text{ND}(C_n) = \min_d \left\{ \begin{array}{l} \text{For all } c \in C_n, \text{ there exists a labeled sample } S_c \text{ of } |S_c| \leq d \\ \text{such that } c \text{ is consistent with } S_c \text{ and for all } c' \in C_n \text{ that} \\ \text{are consistent with } S_c, c \subseteq c' \end{array} \right\}.$$

Natarajan shows that this dimension measure characterizes the class of Boolean functions that are learnable with one-sided error from polynomially many examples. He also gives the following result relating this dimension measure to the VC-dimension.

Theorem [Natarajan] *For concept classes C_n chosen from the domain of Boolean functions over n variables,* $\text{ND}(C_n) \leq \text{VCD}(C_n) \leq n \cdot \text{ND}(C_n)$.

Note that the definition of Natarajan's dimension measure is similar to the teaching dimension except that if there is more than one concept consistent with the given set of examples, the learner is required to pick the most specific one. Using this correspondence we obtain the following result.

Lemma 5.3 *For concept classes C_n chosen from the domain of Boolean functions,* $\text{TD}(C_n) \geq \text{ND}(C_n)$.

Proof: Suppose there exists a concept class for which $\text{TD}(C_n) < \text{ND}(C_n)$. By the definition of the teaching dimension, there exists a sequence of $\text{TD}(C_n)$ examples that uniquely specify the target concept from all concepts in C_n. Let this sequence of $\text{TD}(C_n)$ examples be the labeled sample S_c used in the definition of $\text{ND}(C_n)$. Since S_c uniquely

specifies $c \in C_n$ there are no $c' \in C_n$ for $c' \neq c$ that are consistent with S_c. This gives a contradiction, thus proving that $\text{TD}(C_n) \geq \text{ND}(C_n)$. ■

Combining Lemma 5.3 with the theorem of Natarajan gives the following result.

Corollary 5.1 *For concept classes C_n chosen from the domain of Boolean functions over n variables, $\text{VCD}(C_n) \leq n \cdot \text{TD}(C_n)$.*

5.1.2 Computing the Teaching Dimension

In this section we compute the teaching dimension for several concept classes, and provide general techniques to aid in computing the teaching dimension for other concept classes.

We begin by noting that given a concept class C and target concept $c \in C$ the problem of finding an optimal teaching sequence is equivalent to finding an optimal set covering. (See Garey and Johnson [19] for the definition of the set covering problem.) To see these are equivalent problems observe that one can view the instances as the objects in the set cover problem, and the concepts as the sets. Finally, an object is in a set if and only if the concept associated with the set disagrees with the classification of the target concept on the instance associated with the object. Clearly an optimal teaching sequence directly corresponds to an optimal set covering.

While it is $\mathcal{NP}$-complete to compute an optimal set covering [19], Chvatal [14] proves that the greedy algorithm (which is a polynomial-time algorithm) computes a cover that is within a logarithmic factor of optimal. While this problem appears to be well understood, there is one very important distinction—in the set covering problem no assumptions are made about the structure of the sets, whereas for the problem of computing an optimal teaching sequence there is assumed to by a short (polynomial-sized) representation of the objects contained in each set. Thus we suggest the following question: What is the complexity of the set covering problem when there is structure among the objects contained in each set?

Although the problem of computing the optimal teaching sequence for teaching a given concept is interesting, for the remainder of this chapter we focus on computing the

teaching dimension for various concept classes. We start by describing a straightforward relation between the teaching dimension and the number of membership queries needed to achieve exact identification.

Observation 5.2 *The number of membership queries needed to exactly identify any given $c \in C_n$ is at least* $\text{TD}(C_n)$.

Proof: Suppose $\text{TD}(C_n)$ is greater than the number of membership queries needed for exact identification. By the definition of exact identification, the sequence of membership queries used must be a teaching sequence that is shorter than the claimed shortest teaching sequence. This gives a contradiction. ∎

Thus an algorithm that achieves exact identification using membership queries provides an upper bound on the teaching dimension.

We now compute the teaching dimension for several concept classes. Recall that in Chapter 3 we proved that for the concept class of k-binary-relations (where the corresponding matrix has n rows and m columns), the teaching dimension is $km+(n-k)(k-1)$. In Chapter 4 we showed that for the problem of learning a total order on a set of n objects, the teaching dimension is $n-1$. In the remainder of this section we study the teaching dimension for: monotone monomials, arbitrary monomials, monotone rank-1 decision trees, orthogonal rectangles in $\{0,1,\cdots,n-1\}^d$, monotone k-term DNF, and k-term μ-DNF.

Monomials

We now compute the teaching dimension for the class of monomials. For the case in which the director is an adversary, Littlestone [43] describes an algorithm that makes at most $2r \lg n + 2$ mistakes for learning monotone monomials, where n is the number of variables and r is the number of relevant variables. He then gives a transformation to the class of arbitrary monomials that makes only one additional mistake. We now prove a tight bound on the teaching dimension for the class of monotone monomials, and then generalize this result for arbitrary monomials.

Theorem 5.3 *For the concept class C_n of monotone monomials over n variables*

$$\text{TD}(C_n) = \min(r+1, n)$$

where r is the number of relevant variables.

Proof: We begin by exhibiting a teaching sequence of length $\min(r+1, n)$. First present a positive example in which all variables in the target monomial are 1 and the rest are 0. For example, if there are five variables and the target is $v_2v_3v_5$ then present "01101,+". This example ensures that no irrelevant variables are in the target monomial. (Any example in which a relevant variable is 0 must be negative.) If all variables are in the monomial then this positive example can be eliminated.

Next present r negative examples to prove that the r relevant variables are in the monomial. To achieve this goal, take the positive example and negate each relevant variable, one at at time. For the example above, the remainder of the teaching sequence is "00101,–", "01001,–", and "01100,–". So for each relevant variable v_r there is a positive and negative example that differ only in the value of v_r thus proving that v_r is relevant. Thus this sequence is a teaching sequence.

We now prove that no shorter sequence of examples suffices. If any variable is not in the monomial, a positive example is required to rule out the monomial containing all variables. We now show that r negative examples are required. What information is revealed by a single negative example? At best, it proves that at least one relevant variable, from those that are 0, must be in the target. Suppose a set of $r-1$ negative examples (and any number of positive examples) proved that all r relevant variables must be in the target. We construct a monomial, missing a relevant variable, that is consistent with this example sequence: for each negative example select one of the relevant variables that is 0 and place it in the monomial. This procedure clearly creates a consistent monotone monomial with at most $r-1$ literals. ■

We now extend these ideas to give a tight bound on the teaching dimension for the class of arbitrary monomials. The key modification is that the positive examples not only

prove which variables are relevant, but they also provide the sign of the relevant variables. (By the sign of a variable we simply mean whether or not the variable is negated.)

Theorem 5.4 *For the concept class C_n of monomials over n variables*

$$\text{TD}(C_n) = \min(r+2, n+1)$$

where r is the number of relevant variables.

Proof: First we exhibit a teaching sequence of length $\min(r+2, n+1)$. We present two positive examples, in each make all literals in the target monomial true and reverse the setting of all irrelevant variables. For example, if there are five variables and the target monomial is $\overline{v}_1 v_2 v_5$ then present "01001,+" and "01111,+". Next r negative examples are used to prove that each remaining literal is in the monomial: take the first positive example and negate each relevant variable, one at a time. For the example above, the remainder of the teaching sequence is "11001, –", "00001,–", and "01000,–".

We now prove that the above example sequence is a teaching sequence for the target monomial. We use the following facts.

Fact 5.1 *Let C_n be the class of monomials and let $c \in C_n$ be the target concept. If some variable v is 0 in a positive example then v cannot be in c. Likewise, if v is 1 in a positive example then $\overline{v}$ cannot be in c.*

Fact 5.2 *Let C_n be the class of monomial, and let $c \in C_n$ be the target concept. Let $x^+ \in X_n$ be a positive example and $x^- \in X_n$ be a negative example. If x^+ and x^- are identical except that some variable v is 1 in x^+ and 0 in x^-, then v must appear in c. Likewise, if x^+ and x^- are identical except that some variable v is 0 in x^+ and 1 in x^-, then $\overline{v}$ must appear in c.*

We first prove that no irrelevant variables are in a monomial consistent with the positive examples. Since each irrelevant variables is set to both 0 and 1 in a positive example, it follows from Fact 5.1 than each of these variables could not appear in any consistent

monomial. (Each relevant variable has the same value in both positive instances.) From Fact 5.1 it also follows that the positive examples provide the signs of the relevant variables. (If all variables are in the monomial, then only one positive example revealing the sign of each variable is needed.)

We now show that any monomial consistent with the negative examples must contain all relevant variables. For each relevant variable v_r the teaching sequence contains a positive and negative example that differ only in the assignment to v_r; so by Fact 5.2, v_r must be relevant. Thus the above example sequence is in fact a teaching sequence.

We now prove that no shorter sequence suffices. If any variable, say v_i is irrelevant then at least two positive examples are required in a valid teaching sequence since the teacher must prove that both v_i and $\overline{v_i}$ are not in the target monomial. Finally, the argument used in Theorem 5.3 proves that r negative examples are needed. ■

Observe that Theorems 5.3 and 5.4 can easily be modified to give the dual result for the classes of monotone and arbitrary 1-DNF formulas.

Monotone Rank-1 Decision Trees

Next we consider the concept class of monotone rank-1 decision trees. Let $V_n = \{v_1, v_2, \ldots, v_n\}$ be a set of n Boolean variables. Let the instance space $X_n = \{0,1\}^n$. The class C_n of *monotone rank-1 decision trees* [18] is defined as follows. A concept $c \in C_n$ is a list $L = \langle (y_1, b_1), \ldots (y_\ell, b_\ell) \rangle$ where each $y_i \in V_n$ and each $b_i \in \{0,1\}$. For a instance $x \in X_n$, we define $L(x)$, the output from L on input x, as follows: $L(x) = b_j$ where $1 \leq j \leq \ell$ is the least value such that y_j is 1 in x; $L(x) = 0$ if there is no such j. Let $b(v_i)$ denote the bit associated with v_i. One may think of a rank-1 decision tree as an extended "**if—then—elseif—... else**" rule. We note that this class is equivalent to 1-decision lists as defined by Rivest [56]. See Figure 5-3 for an example monotone rank-1 decision tree on $v_1, \ldots, v_7$. Some positive instances for this concept are "0100111" and "0010000", and some negative instances are "0000101" and "0100000".

We now give an asymptotically tight bound on the teaching dimension for the class

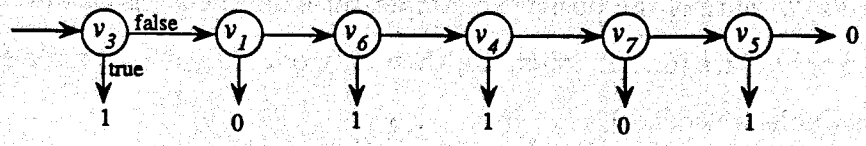


Figure 5-3: A diagram showing the following monotone rank-1 decision tree: $\langle (v_3, 1), (v_1, 0), (v_6, 1), (v_4, 1), (v_7, 0), (v_5, 1) \rangle$.

of monotone rank-1 decision trees.

Theorem 5.5 *For the concept class C_n of monotone rank-1 decision trees:*

$$\text{TD}(C_n) \leq 2n - 1.$$

Proof: We construct a teaching sequence of length at most $2n - 1$. For each variable v_i (imagine that all irrelevant variables are at the end of the list with an associated bit of 0), we first teach $b(v_i)$ and then we teach the ordering of the nodes.

To teach $b(v_i)$ present the instance in which v_i is 1 and all other variables are 0. Then $b(v_i)$ is 1 if and only if $x = (v_1 v_2 \cdots v_n)$ is a positive example. Thus using n examples, we teach the bit associated with each variable. Next we teach the ordering of the nodes. Observe that for consecutive nodes v_i and v_j for which $b(v_i) = b(v_j)$, reversing the order of these nodes produces an equivalent concept. Thus, the learner can order them arbitrarily. For each $1 \leq i \leq n - 1$ we present the example in which all variables are 0 except for v_i and all v_j where $j > i$ and $b(v_j) \neq b(v_i)$. (Figure 5-4 shows the teaching sequence for the target concept of Figure 5-3.) Observe that the ith example from this second part of the teaching sequence proves that v_i precedes all nodes v_j for which $j > i$ and $b(v_i) \neq b(v_j)$. This ordering information is sufficient to reconstruct the ordering of the nodes. ■

We now show that the upper bound of Theorem 5.5 is asymptotically tight by proving the the teaching dimension of monotone rank-1 decision trees is at least n. Unless the learner knows $b(v_i)$ for all i, he could not possibly know the target concept. However, to

1 0 0 0 0 0 0 , −
0 1 0 0 0 0 0 , −
0 0 1 0 0 0 0 , +
0 0 0 1 0 0 0 , +
0 0 0 0 1 0 0 , +
0 0 0 0 0 1 0 , +
0 0 0 0 0 0 1 , −

1 1 1 0 0 0 1 , +
1 0 0 1 1 1 0 , −
0 1 0 0 0 1 1 , +
0 1 0 1 0 0 1 , +
0 0 0 0 1 0 1 , −
0 1 0 0 1 0 0 , +

Figure 5-4: Teaching sequence for the target concept shown in Figure 5-3.

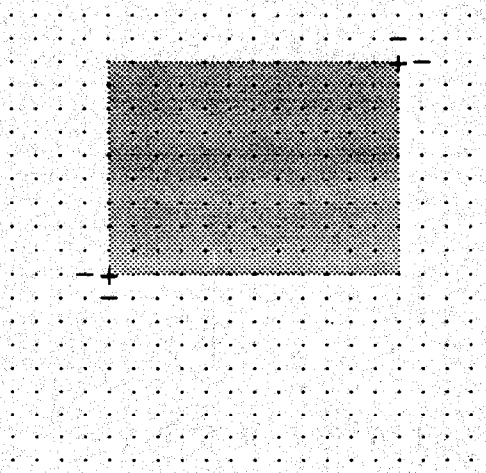


Figure 5-5: The teaching sequence for a concept selected from the class of orthogonal rectangles in $\{0,1,\cdots,n-1\}^d$ for $n = 20$ and $d = 2$.

teach $b(v_i)$ (for $1 \leq i \leq n$), n examples are needed: any single example only teaches b_j for the smallest j for which y_j is 1.

Orthogonal Rectangles in $\{0,1,\cdots,n-1\}^d$

We now consider the concept class, BOX_n^d, of orthogonal rectangles in $\{0,1,\cdots,n-1\}^d$. Maass and Turan [46] have shown a mistake bound of $\theta(d \lg n)$ for learning BOX_n^d when an adversary selects the query sequence.

Theorem 5.6 *For the concept class C_d of orthogonal rectangles in $\{0,1,\cdots,n-1\}^d$:*

$$\text{TD}(C_d) = 2 + 2d.$$

Proof: We build the following teaching sequence T. Select any two opposing corners of the box, and show those points as positive instances. Now for each of these points show the following d negative instances: for each dimension, give the neighboring point (unless the given point is on the border of the space $\{0,1,\cdots,n-1\}^d$ in the given dimension) just outside the box in that dimension as a negative instance. (See Figure 5-5 for an example teaching sequence.) Clearly the target concept is consistent with T, thus to prove that

T is a teaching sequence we need only show that it is the only concept consistent with T. Let B be the target box and suppose there is some other box B' that is consistent with T. Since $B \neq B'$ either B' makes a false positive or false negative error. However, the two positive examples ensure that $B' \supseteq B$ and the negative examples ensure that $B \supseteq B'$. Thus such a B' could not exist.

We now show that no sequence T' of less than $2 + 2d$ examples could be a teaching sequence. Suppose $n \geq 4$ and let the target concept be a box defined by opposing corners 1^d and $(n-2)^d$. We first argue that T' must contain two positive points—if T' contained a single positive point then the box containing only that point would be consistent with T'. Thus any teaching sequence must contain at least two positive points. Finally to prevent a hypothesis B' from making a false positive error, there must be a negative example that eliminates any hypothesis that moves each face out by even one unit. Clearly a single point can only serve this purpose for one face. Since a d-dimensional box has $2d$ faces, $2d$ negative examples are needed. ■

Monotone k-term DNF Formulas

We now describe how bounds on the teaching dimension for simple concept classes can be used to derive bounds on the teaching dimension for more complex classes. We begin by using the result of Theorem 5.3 to upper bound the teaching dimension for monotone k-term DNF formulas. We note that it is crucial to this result that the learner knows k. While the teacher can force the learner to create new terms, there is no way for the teacher to enforce an upper bound on the number of terms in the learner's formula.

Lemma 5.4 *For the class C_n of monotone k-term DNF formulas,*

$$\text{TD}(C_n) \leq \ell + k$$

where ℓ is the number of literals in the target formula.

Proof: Let $f = t_1 \vee t_2 \vee \cdots \vee t_k$ be the target formula, and let $f(x) : x \in X_n \rightarrow \{+, -\}$ denote the value of f on input x. We assume, without loss of generality, that f is

reduced meaning that f is not equivalent to any formula obtained by removing one of its terms. Thus for all i and j, $t_i \not\supset t_j$ (where $t_i \supset t_j$ denotes that for any instance x, $t_j(x) = + \Rightarrow t_i(x) = +$). The approach we use is to independently teach each term of the target formula.

For all i we build the teaching sequence T_i for term t_i (as if t_i was a concept from the class of monotone monomials) as described in Theorem 5.3. As an example consider the case in which there are five variables and the target is $v_2v_3v_5 \vee v_1v_3$. For the term $v_2v_3v_5$ the teaching sequence is "01101,+", "00101,–", "01001,–" and "01100,–". Observe that the other term v_1v_3 contains a variable, namely v_1, that is not in $v_2v_3v_5$. Since v_1 is 0 for any instance x in the above teaching sequence, x is positive if and only if $v_2v_3v_5$ is true on input x. Likewise the variable v_2 is in $v_2v_3v_5$, but not in v_1v_3. As we shall see, the teaching sequences for $v_2v_3v_5$ and v_1v_3 when combined form a teaching sequence for $v_2v_3v_5 \vee v_1v_3$.

We now use the ideas from the above example to prove that $T = T_1T_2 \cdots T_k$ is a teaching sequence for f. The key property we use is:

$$\text{for any } x \in T_i, f(x) = +\text{if and only if } t_i(x) = +. \tag{5.1}$$

To prove that property (5.1) holds we prove that for all $x \in T_i$, all terms except for t_i are negative on on x. Recall that all variables not in t_i are 0 in every $x \in T_i$. Thus we need just prove that each term of f, except for t_i, must contain *some* variable that is not in t_i. Suppose for term t_j, no such variable exists. Then t_j would contain a subset of the variables in t_i. However, this violates the assumption that f is reduced. Thus property (5.1) holds.

We now use property (5.1) to prove that T is a teaching sequence for f. We first show that f is consistent with T. From property (5.1) we know that f is positive on $x \in T_i$ if and only if $t_i(x) = +$—and since T_i is a teaching sequence for t_i it follows that $f(x) = +$ if and only if x is positive. Thus f is consistent with T.

We now use a proof by contradiction to show that T uniquely specifies f. For monotone monomials $g_1, g_2, \ldots, g_k$ suppose $g = g_1 \vee \cdots \vee g_k$ is consistent with T yet $g \neq f$. Then

there must exist some term t_i from f that is not equal to any term in g. Without loss of generality suppose that $g_j(x) = +$ for a positive point $x \in T_i$. (Some term in g must be true since g is assumed to be consistent with T_i.) By property (5.1), this implies that g_j can only contain those variables that are in t_i. Finally, since g_j must reply correctly on all negative points in t_i it follows that $g_j = t_i$ giving the desired contradiction.

To complete the proof of the theorem, we need just compute this size of T. From Theorem 5.3 we know that for each i, $|T_i| \leq r + 1$ where r is the number of literals in t_i. Thus it follows that $|T| \leq \ell + k$ where ℓ is the number of literals in f. ∎

We note that by using teaching sequence from monotone 1-DNF formulas as the "building blocks" we can prove a dual result for monotone k-term CNF formulas .

k-term μ-DNF Formulas

We now extend the idea of Lemma 5.4 to use teaching sequences for monomials to build a good teaching sequence for a k-term μ-DNF formula.

Lemma 5.5 *For the class C_n of k-term μ-DNF formulas,*

$$\text{TD}(C_n) \leq n + 2k.$$

Proof: As in Lemma 5.4 we assume that the target formula $f = t_1 \vee t_2 \vee \cdots \vee t_k$ is reduced. Once again, the key idea here is to independently teach each term of f. For each term of f we begin by letting T_i be the teaching sequence for t_i (as if t_i was a concept from the class of monomials) as described in Theorem 5.4. Then we modify $T = T_1 \cdots T_k$ as follows. For each singleton term t_i (e.g. $t_i = v_j$ or $t_i = \overline{v_j}$) modify all examples in $T - T_i$ by setting v_j so that t_i is false. For each remaining term t_i modify T so that at least one literal from term t_i is false in all examples in $T - T_i$. (Since f is a μ-formula and all remaining terms contain at least two literals, this goal is easily achieved.)

We first prove that T is a teaching sequence for f. It is easy to see that for any instance $x \in T_i$, all terms in the formula, except for t_i, are false on input x. The literal in any singleton term t_i is only true for elements in T_i. For any other term t_j, for any

instance $x \in T - T_j$ at least one literal from t_j is false and thus t_j is false. What remains to be proven is that T_i $(1 \leq i \leq k)$ is a teaching sequence for t_i. However, if one individually considers each T_i this may not be true. The key observation is to first consider the portion of the teaching sequence associated with the singleton terms. It follows from the proof of Theorem 5.4 that each singleton is a term in the formula. Finally, since each variable can only appear in one term, the portion of the teaching sequence associated with each remaining term is a teaching sequence for that term. Thus the technique of Theorem 5.4 can be used to prove that T is a teaching sequence for f. ■

Classes Closed Under XOR

We now discuss a situation in which one can generate a teaching sequence that has length logarithmic in the size of the concept class. For $c_1, c_2 \in C$ we define $c = c_1 \text{XOR } c_2$ as follows: for each instance $x \in X_n$, $c(x)$ is the exclusive-or of $c_1(x)$ and $c_2(x)$. We say a concept class C is *closed under* XOR if the concept c obtained by taking the bitwise exclusive-or of any pair of concepts $c_i, c_j \in C$ (for all i, j) is also in C.

Theorem 5.7 *If C_n is closed under* XOR *then there exists a teaching sequence of size at most* $\lfloor \lg(|C_n| - 1) \rfloor + 1$.

Proof: We construct a teaching sequence using the following algorithm.

Build-teaching-sequence (f_{target}, C_n)
1 Repeat until f_{target} is uniquely determined
2 Find instance, x, for which f_{target} disagrees with a non-eliminated function from C_n
3 Use x as the next example

We now show that each instance selected by *Build-teaching-sequence* removes at least half of the non-eliminated functions from $C_n - \{f_{\text{target}}\}$. Consider the example x added in step 2 of *Build-teaching-sequence*. Let τ contain the examples that have already been presented to the learner, and let $\mathcal{V}$ contain the concepts from $C_n - \{f_{\text{target}}\}$ that are consistent with τ. We now show that at least half of the concepts in $\mathcal{V}$ must disagree with x. Suppose that x is a positive example. If all the concepts in $\mathcal{V}$ predict that x is

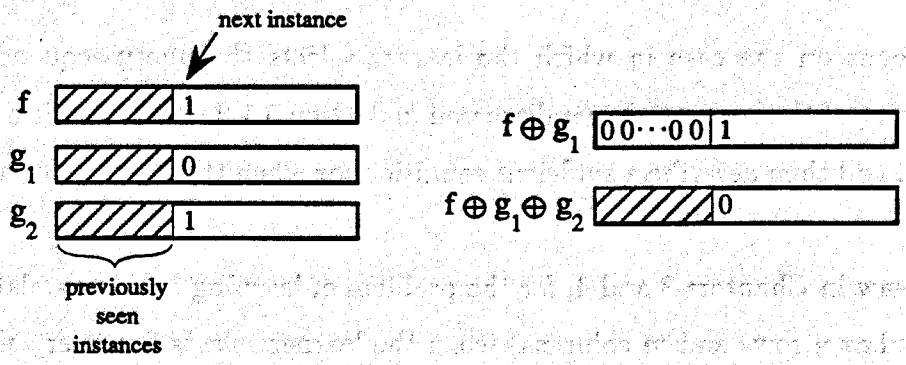


Figure 5-6: Result of combining with XOR.

negative, then x eliminates all remaining concepts in $\mathcal{V}$ besides the target. Otherwise, there exists some set $\mathcal{V}_x^+$ of concepts that predict x is positive. (Let $\mathcal{V}_x^- = V - \mathcal{V}_x^+$ be the concepts from V that predict x is negative.) By the choice of x, it must be that $|\mathcal{V}_x^-| \geq 1$—let g_1 be a concept in $\mathcal{V}_x^-$. We now use the fact that C_n is closed under XOR to prove that for each concept $g_2 \in \mathcal{V}_x^+$, there is a one-to-one mapping to a concept in $\mathcal{V}_x^-$. First consider the result of taking $f \oplus g_1$. Since all elements in $\mathcal{V}$ are consistent with instances in τ, for these instances $f \oplus g_1$ is 0. For the instance x, $f \oplus g_1$ is 1. Now consider taking $f \oplus g_1 \oplus g_2$ for $g_2 \in \mathcal{V}_x^+$. Since the instances in τ are 0 in $f \oplus g_1$, in $f \oplus g_1 \oplus g_2$ the instances in τ are the same as in f. For x, $f \oplus g_1 \oplus g_2$ is 0. Finally, since all concepts in $\mathcal{V}_x^+$ must disagree with each other on some instance in $X_n - \tau - \{x\}$, the concept $f \oplus g_1 \oplus g_2 \in \mathcal{V}_x^-$ forms a one-to-one mapping with the concepts $g_2 \in \mathcal{V}_x^+$. (See Figure 5-6.)

Repeating this same argument when x is a negative instance, we conclude that on each instance at least half of the concepts in $\mathcal{V}$ are eliminated. Since *Build-teaching-sequence* removes at least half of the non-eliminated concepts with each example, after at most $\lfloor \lg(|C_n| - 1) \rfloor$ examples $\mathcal{V}$ contains at most one concept from $C_n - \{f\}$. Finally, one additional example is used to distinguish this remaining concept from the target concept. Thus the length of the teaching sequence is at most $\lfloor \lg(|C_n| - 1) \rfloor + 1$. ■

5.2 Self-directed Learning

We now focus on the case in which the learner selects the query sequence. We first consider some of the concept classes discussed in Section 5.1.2 when studying the teaching dimension, and then describe a sufficient condition for when the learner can make a single mistake.

As we saw in Chapters 3 and 4, for the problem of learning k-binary-relations (where the matrix has n rows and m columns) when the learner selects the query sequence, the number of mistakes is $\theta(km + (n-k)\lg k)$, and for the problem of learning a total order in the case that the learner selects the query sequence, there is a tight mistake bound of $n-1$. We now consider the mistake bounds under self-directed learning for the classes of monotone monomials, monomials, monotone k-DNF formulas, and orthogonal rectangles in $\{0,1,\cdots,n-1\}^d$.

5.2.1 Monomials

We first give an algorithm to learn monotone monomials and we then modify it to learn arbitrary monomials.

Theorem 5.8 *There exists a learning algorithm for learning monotone monomials that makes at most one mistake under self-directed learning.*

Proof: The learner uses the following query sequence, always predicting that the instances are negative, and stopping when the first mistake occurs. First consider the instance in which all variables are 0. Next consider the n instances in which a single variable is 1. Then consider the $\binom{n}{2}$ instances in which two variables are 1, and so on. (See Figure 5-7.)

Let c be the target monomial and let c' be the monomial consisting of the variables that are 1 in the incorrectly predicted instance x. We now prove that $c' = c$. Clearly any variable in c must also be in c'—if variable v is in c then v must be 1 in any positive example. Suppose that some irrelevant variable v is 1 in the incorrectly predicted

$$
\begin{array}{cccccccccc}
0 & 0 & 0 & \cdots & 0 & 0 & 0 & , & - \\
1 & 0 & 0 & \cdots & 0 & 0 & 0 & , & - \\
0 & 1 & 0 & \cdots & 0 & 0 & 0 & , & - \\
 & & & \vdots & & & & & \\
0 & 0 & 0 & \cdots & 0 & 1 & 0 & , & - \\
0 & 0 & 0 & \cdots & 0 & 0 & 1 & , & - \\
1 & 1 & 0 & \cdots & 0 & 0 & 0 & , & - \\
1 & 0 & 1 & \cdots & 0 & 0 & 0 & , & - \\
 & & & \vdots & & & & & \\
0 & 0 & 0 & \cdots & 1 & 0 & 1 & , & - \\
0 & 0 & 0 & \cdots & 0 & 1 & 1 & , & - \\
1 & 1 & 1 & \cdots & 0 & 0 & 0 & , & -
\end{array}
$$

Figure 5-7: Learner-selected query sequence for learning the monotone monomial $x_1x_2x_3$. Note that the last instance is the first one incorrectly predicted by the learner.

instance, yet it is not in c. Consider the positive instance x' which is the same as x except that v is 0. Clearly x' must precede x in the query sequence defined above. Since the learner reaches the instance x it follows that x' must be a negative instance, giving the desired contradiction. ∎

We now modify these ideas to handle the situation in which some variables in the monomial may be negated.

Theorem 5.9 *There exists a learning algorithm for learning arbitrary monomials that makes at most two mistakes under self-directed learning.*

Proof: The algorithm used here is a simple modification of the algorithm for learning monotone monomials. Suppose that the learner knew the sign of each variable. Then the learner can use the algorithm for learning monotone monomials where setting a variable to 0 (respectively 1) is interpreted as setting the variable so that the corresponding literal is false (respectively true).

Finally, we use a standard trick [43] to learn the sign of each variable making only one mistake. For arbitrarily chosen instances, make the prediction that the instance is a negative instance. Let x be the positive instance obtained on the first mistake. The sign of each relevant variable is given by its assignment in x. ■

5.2.2 Monotone k-term DNF formulas

In this section we consider learning monotone k-term DNF formulas under self-directed learning. We obtain our algorithm for learning monotone k-term DNF formulas by extending the algorithm of Theorem 5.8 to handle the conjunction of k monotone monomials.

Theorem 5.10 *There exists a learning algorithm for the class of monotone k-term DNF formulas that makes at most k mistakes under self-directed learning.*

Proof Sketch: The algorithm used here is a modification of that described in Theorem 5.8. The query sequence selected is like the one shown in Figure 5-7 except that an instance x is predicted as positive if the monomial corresponding to any incorrectly predicted instance predicts that x is positive. Using the same technique as in the proof of Theorem 5.8, one can show that the target formula is just the disjunction of the monomials corresponding to the incorrectly predicted instances. ■

5.2.3 Orthogonal Rectangles in $\{0,1,\cdots,n-1\}^d$

Finally, in this section we consider the concept class BOX_n^d of orthogonal rectangles in $\{0,1,\cdots,n-1\}^d$.

Theorem 5.11 *For the concept class of BOX_n^d, there exists an algorithm that makes at most two mistakes under self-directed learning.*

Proof: Select two opposing corners of the space $\{0,1,\cdots,n-1\}^d$. Let L be the line through these two opposing corners. Our teaching sequence finds each corner of the

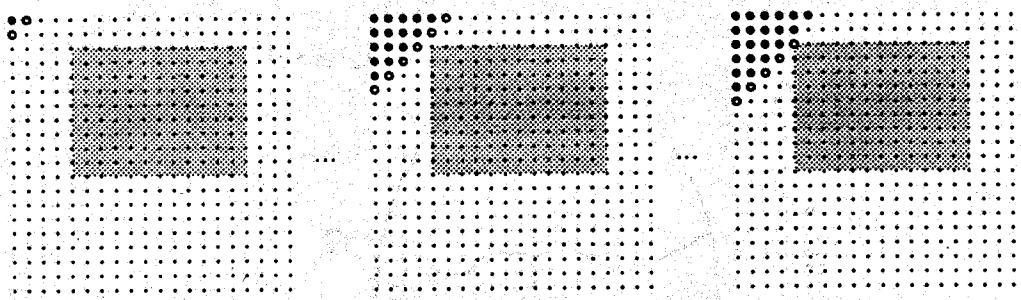


Figure 5-8: The query sequence for learning a concept from BOX_n^2. The filled circles in the figure represents correctly predicted negative instances. The unfilled circles represent the instances on the frontier. Finally, note that in the last figure the querying stops when a mistake is made predicting that the corner of the box is a negative instance.

target box by approaching it with a hyperplane perpendicular to L. That is, for each opposing corner present the following set of instances, predicting that each is negative.

Learn-box-corner (corner pt)
1 Query the corner point, predicting it is negative
2 **while** no mistake has occurred
3 **for** points p in the set of seen points
4 Query unseen points reachable from p by taking one step in any dimension measure

So basically, dovetailing is used to find each corner. See Figure 5-8 for an example of the query sequence for learning a box in two-dimensional space.

At the first mistake, we claim that a corner point of the box has been found. Since the target box is approached with a hyperplane perpendicular to the axis between the two corner points of the space, the first point of the box queried must be the corner point. Finally, we argue that the learner has seen a negative example (predicted correctly) corresponding to the negative instances in the teaching sequence for BOX_n^d. Note for each corner that these points are along the previous position of the hyperplane. Finally, the second corner is found in the same manner. Since a teaching sequence has been seen, the learner knows the exact location of the box. ■

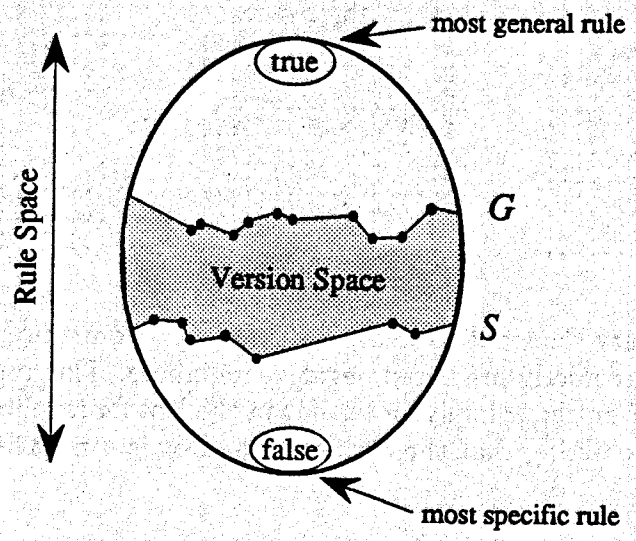


Figure 5-9: The rule space.

5.2.4 Mitchell's Version Space Algorithm

We have demonstrated that the number of mistakes made under self-directed learning may be quite small. Is there some characterization for the situations in which the learner can perform so well? As a partial answer to this question, we describe a relation between this work and Mitchell's version space algorithm.

We begin by describing Mitchell's version space algorithm. The *rule space* (hypothesis space) is the set of all rules available to the learner. The *version space* with respect to a given sample is the set of *all* rules that are consistent with the sample (i.e. example sequence). The set G is the list of the *most general* rules in the version space. The set S is the list of *most specific* rules in the version space. (See Figure 5-9.) The main ideas of the version space algorithm are as follows. Initially let G be the formula that is always true and let S be the formula that is always false. Then for each example, both G and S are appropriately updated. See Haussler [26, 27] for a discussion of Mitchell's version space algorithm viewed under the distribution-free learning model.

We now give the following relation between the version space algorithm and situations

in which the learner can make a single mistake.

Theorem 5.12 *If for all presentation orders of the instances $|S| = 1$ in Mitchell's version space algorithm, then there exists a self-directed learning algorithm that makes at most one mistake.*

Proof: The learning algorithm is as follows. For each rule r in the rule space let $i(r)$ be the instance that is positive if and only if the target is r or a generalization of r. Note that since $|S| = 1$ such an instance must exist for each rule in the rule space. The algorithm goes through the rules from most general to most specific (i.e. go through the layers of the lattice from top to bottom) and selects the instance $i(r)$ with a prediction that it is a negative instance. We claim that when the first mistake is made, the target rule is the single rule in S. Clearly, when the first mistake is made it is known that the target is r or a generalization of r. However, since the rules are considered from most general to most specific, all rules that are generalizations of r must already be eliminated from the version space. Thus the target rule must be r. ∎

As an application note that Bundy et al. [12] have noted that for the class of monotone monomials* the set S never contains more than one hypothesis.

We now extend this result to concept classes that are made of of the disjunction of rules from a space for which $|S| = 1$.

Theorem 5.13 *Let C_n be a concept class for which $|S| = 1$ regardless of the presentation order for the instances. Then a concept consisting of $c_1 \vee c_2 \vee \ldots \vee c_k$ for $c_1, \ldots, c_k \in C_n$ can be learned with at most k mistakes under a learner selected query sequence.*

Proof Sketch: The algorithm used here is a modification of the algorithm given in the proof of Theorem 5.12. However, the instance $i(r)$ is predicted to be positive if it is a specialization of a rule for which a mistake was made on a previous predictions. Using the same technique as in the proof of Theorem 5.12, one can show that the target formula

*Actually, they show this claim for the more general class of pure conjunctive concepts over a finite set of tree-structured attributes.

is just the disjunction of the rules corresponding to the incorrectly predicted instances. ■

Applying the result of Bundy et al. [12] we get the result of Theorem 5.10.

Finally, all the results we have described relating the number of mistakes made under a learner selected query sequence to Mitchell's version space algorithm can be extended to give the dual results for when $|G| = 1$. Namely, if for C_n the set G always contains a single element then there exists a self-directed learning algorithm that makes at most one mistake. Furthermore, there exists a self-directed learning algorithm that can learn $c_1 \wedge c_2 \wedge \cdots \wedge c_k$ for $c_1, \ldots, c_k \in C_n$ making at most k mistakes.

5.3 Conclusions and Open Problems

In this chapter we have studied some of the interesting questions related to the extended mistake-bound model. In particular we consider the power of the teacher or learner selecting the instances when the learner's performance is judged on the number of incorrect predictions it makes. See Table 5.1 for a summary of the specific results. (The lower bounds for monotone k-term DNF and k-term μ-DNF are obtained by just letting $k = 1$ and using the bounds for monomials.) Observe that for these concept classes, the teaching sequence selected is highly dependent on the target concept. In Section 6.6 of Chapter 6 we describe a situation in which the instances in the teaching sequence are *independent* of the target concept.

Along with this specific results, we have described results relating the teaching dimension to the Vapnik-Chervonenkis dimension and given some general results for both the case when the teacher or learner selects the instances.

Finally we explore the following interesting paradox raised by our results: for many concept classes, the number of mistakes made under teacher-directed learning may be worse that the number of mistakes made with self-directed learning. Intuitively, the "bright" student may learn the lesson quicker working on his own rather than listening

Concept Class	Director	Lower Bound	Upper Bound
Monotone Monomials	Teacher	$\min(r+1,n)$	$\min(r+1,n)$
	Learner	1	1
Monomials (r relevant vars.)	Teacher	$\min(r+2,n+1)$	$\min(r+2,n+1)$
	Learner	2	2
Monotone k-term DNF (ℓ literals)	Teacher	$\ell+1$	$\ell+k$
	Learner	1	k
k-term μ-DNF	Teacher	n	$n+2k$
Monotone Rank-1 Decision Trees	Teacher	n	$2n-1$
Orthogonal Rectangles in $\{0,1,\cdots,n-1\}^d$	Teacher	$2+2d$	$2+2d$
	Learner	2	2

Table 5.1: Summary of results on teacher-directed and self-directed learning for concept learning.

to the lecture designed for the slower student. This phenomenon occurs because of the restriction that the mistake bound in teacher-directed learning apply for all consistent learners; thus it is possible to get better mistake bounds under self-directed learning. For example the self-directed learning algorithm for orthogonal rectangles cleverly dovetails towards the corner predicting that all instances are negative, thus making only one mistake to find the corner. However, such a query sequence does not make a good teaching sequence since the learner could instead choose to predict that all instances are positive.

A direction of future research is to further study the helpful teacher and learner instance selectors of the extended mistake-bound model for other concept classes. As we have demonstrated the teaching dimension is highly dependent on the target concept. Another interesting measure might be to consider the expected teaching dimensions where the target concept is chosen at random. While, we have shown that the learner can often make a very small number of mistakes when it selects the query sequence, these learning algorithms often make many queries.

Another very interesting research direction would be to consider a variation of the self-directed learning model in which each query has a small cost, yet each mistake has a large cost. Such a model would encourage the design of learning algorithms that make few mistakes without making an exponential number of correctly-predicted queries.

Chapter 6

Exact Identification of Read-once Formulas Using Amplification Functions

In this chapter we describe efficient algorithms for *exactly* identifying certain families of Boolean formulas by observing the target formula's behavior on examples drawn randomly according to a fixed and simple distribution that is related to the formula's *amplification function*. The amplification function $A_f(p)$ for a function $f : \{0,1\}^N \to \{0,1\}$ is defined as the probability that the output of f is 1 when each of the N inputs to f is 1 independently with probability p. Amplification functions were first studied by Valiant [65] and Boppana [11] in obtaining bounds on monotone formula size for the majority function.

The method used by our algorithms is of central interest. For several classes of formulas, we show that the behavior of the amplification function is *unstable* near the fixed point; that is, the value of $A_f(p)$ varies greatly with a small change in p. This in turn implies that small but easily sampled perturbations of the fixed point distribution (that is, the distribution where each input is 1 with probability p, where $A_f(p) = p$)

This chapter describes joint research with Mike Kearns and Rob Schapire [20].

reveal structural information about the formula. A typical perturbation of the fixed point distribution would hard-wire a single variable to 1 and set the remaining variables to 1 with probability p.

We apply this method to obtain efficient algorithms for exact identification of classes of read-once formulas (in which each variable appears at most once) over various bases. These include the class of logarithmic-depth read-once formulas constructed with three-input majority gates (for which the fixed-point distribution is the uniform distribution), as well as the class of logarithmic-depth read-once formulas constructed with NAND gates (for which the fixed-point distribution assigns 1 to each input independently with probability $p \approx .62$). For these classes, the fixed point of the amplification function is the same for all circuits in the class, resulting in a single simple distribution for the entire class. Since these same classes of circuits are known to be not even weakly learnable in polynomial time in Valiant's model, our results may be interpreted as demonstrating that while there are some distributions which in a computationally bounded setting reveal essentially *no* information about the target circuit, there are natural and simple distributions which reveal *all* information.

For the class of read-once majority formulas there is no previous algorithm for exact identification; for Boolean read-once formulas (a superset of the class of formulas constructed from NAND gates) there is an efficient algorithm using membership queries due to Angluin, Hellerstein and Karpinski [5]. It is interesting to note that if we regard our algorithms' use of a *fixed* distribution as a form of "random" membership queries, then these queries are *non-adaptive*: each query is independent of all previous answers. In contrast, all previous exact identification algorithms, including the algorithm of Angluin, Hellerstein and Karpinski, use highly adaptive query sequences. We formalize these notions and then apply our results in proving the existence of *universal identification sequences* for classes of formulas.

6.1 Preliminaries

Given a Boolean function f from $\{0,1\}^N$ to $\{0,1\}$, Boppana [11] defines its *amplification function* A_f as follows: $A_f(p) = \text{Prob}[f(X_1,\ldots,X_N) = 1]$, where $X_1,\ldots,X_N$ are independent Bernoulli variables that are each 1 with probability p. The quantity $A_f(p)$ is called the *amplification of f at p*. Valiant [65] uses properties of the amplification function for Boolean formulas to prove the existence of a monotone Boolean formula for the majority function of size $O(N^{5.3})$. We sometimes find it useful to give a few of the variables special treatment, and so we introduce the notation $A_{f|\mathcal{S}\leftarrow q}(p)$, for $\mathcal{S} \subseteq \{x_1,\ldots,x_N\}$, to denote the probability that $f(X_1,\ldots,X_N) = 1$ where each X_i is an independent Bernoulli variables that is 1 with probability q if $x_i \in \mathcal{S}$, and with probability p otherwise. For example, $A_{f|\{x_1\}\leftarrow 1}(p)$ is the value of the amplification function for the function obtained by hard-wiring x_1 to be 1 in f. Finally, we denote by $\mathcal{D}^{(p)}$ the distribution over $\{0,1\}^N$ induced by having each variable independently set to one with probability p.

Unlike the previous chapters in which we used an on-line learning model, here we use a batch setting with stochastic instance selection. Furthermore, throughout this chapter the instance space is $\{0,1\}^N$. Finally, instead of using the absolute mistake-bound criterion, we use the exact-identification success criteria. That is, for any $\delta > 0$, the learner must with probability at least $1-\delta$ output a hypothesis that agrees with the target concept on all inputs. Recall that in this setting polynomial means polynomial in N and $1/\delta$. In this chapter we give efficient algorithms that with high probability exactly identify the following classes of formulas:

- **The class of logarithmic-depth read-once majority formulas:** This class consists of Boolean formulas of logarithmic-depth constructed from the basis $\{\text{MAJ}, \text{NOT}\}$ where a MAJ gate computes the majority of three inputs, and each variable appears at most once in the formula. Without loss of generality, we assume all NOT gates are at the input level. See Figure 6-1 for an example formula from this class.

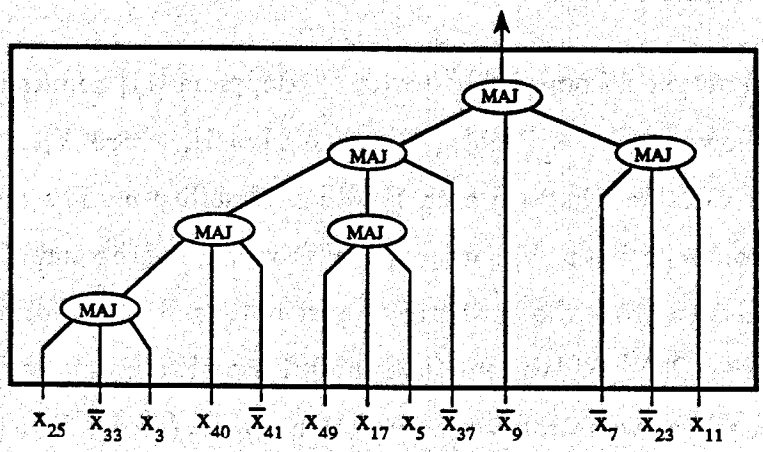


Figure 6-1: An example of a read-once majority formula.

- **The class of logarithmic-depth read-once positive NOR formulas:** This is the class of read-once positive NOR formulas of logarithmic-depth constructed from the basis {NOR} where a NOR gate computes the negation of the OR function of two inputs. Furthermore, each input appears at most once in the formula and all variables are positive (non-negated). See Figure 6-2 for an example of such a formula. Observe that any read-once positive NOR formula f is equivalent to a formula constructed from a basis {AND, OR} where each gate has fan-in two, the output from each OR gate is an input to an AND gate, the output from each AND gate is an input to an OR gate, and the inputs entering AND gates are first negated. This observation is easily proven by applying DeMorgan's law to every other level of f. See Figure 6-3 for the leveled OR/AND formula equivalent to the read-once positive NOR formula shown in Figure 6-2.

- **The class of logarithmic-depth read-once positive NAND formulas:** This is the class of read-once positive NAND formulas of logarithmic-depth constructed from the basis {NAND} where a NAND gate computes the negation of the AND function of two inputs. Furthermore, each input appears at most once in the formula and all variables are positive (non-negated). Like the correspondence described

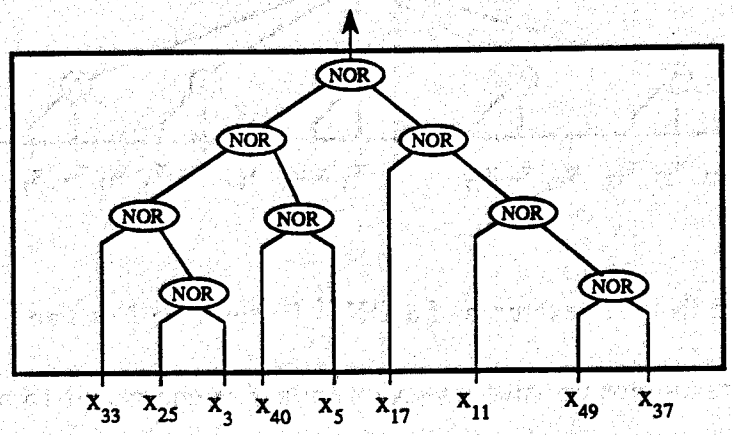


Figure 6-2: An example of a read-once positive NOR formula.

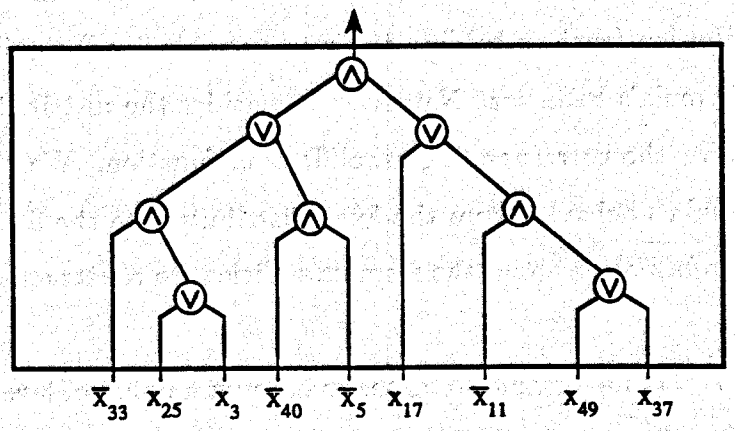


Figure 6-3: The Boolean formula equivalent to the read-once positive NOR formula of Figure 6-2.

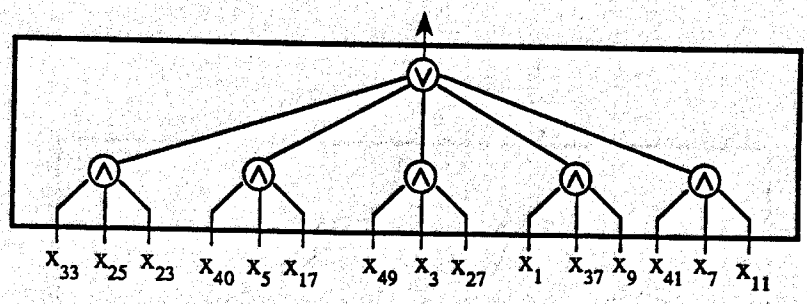


Figure 6-4: An example of a DNF_ℓ^t formula for $\ell = 3$ and $t = 5$.

above, any read-once positive NAND formula f is equivalent to a leveled OR/AND formula in which the inputs entering OR gates are first negated.

- **The class of read-once monotone DNF_ℓ^t formulas:** This is the class of read-once monotone DNF formulas for which there are t terms each with ℓ literals per term. See Figure 6-4 for an example of such a formula for $\ell = 3$ and $t = 5$.

The key idea for all of our learning algorithms is to find an input probability distribution on which the target formula is unstable, in the sense that applying a simple filter to this distribution (such as holding an input variable fixed) significantly alters the statistics of the formula's behavior. Namely, we consider the distribution $\mathcal{D}^{(p)}$ where p is the fixed point for the corresponding amplification function. We show that the difference in the formula's behavior from the fixed distribution to the filtered distributions reveals structural information about the formula sufficient to construct the corresponding formula.

We define the *depth* of a formula to be the maximum number of levels from any input to the output. We define the *level of a gate* g to be the number of gates (including g) on the path to the output. Thus, the output gate is at level 1. Likewise, we define the *level of an input* to be the level of the gate which the input enters. An input x_i *feeds* a gate g if the path from x_i to the output goes through g. For any two input bits x_i and x_j we define $\Gamma(x_i, x_j)$ to be the gate g nearest the input level that is fed by both x_i and

x_j. Likewise, $\Gamma(x_i, x_j, x_k)$ denotes gate g nearest the input level that is fed by x_i, x_j, and x_k. Since the formula is read-once, these gates are unique.

We use Hoeffding's inequality in analyzing the sample complexity of our algorithms, and so we briefly review these bounds now. (We note that these bounds are often referred to as Chernoff bounds.) Let $X_1, X_2, \ldots, X_m$ be independent Boolean random variables each with a probability p of being 1. Let $S = \sum_{i=1}^{m} X_i$; the expectation of S is thus pm. Hoeffding's inequality [32, 61] for bounding the tail of a binomial distribution states that:

$$\Pr[S > pm + \tau] \leq e^{-2m\tau^2} \tag{6.1}$$

$$\Pr[S > \gamma m] \leq e^{-2m(\gamma - p)^2} \tag{6.2}$$

$$\Pr[S < \beta m] \leq e^{-2m(\beta - p)^2} \tag{6.3}$$

for $\gamma \geq p$ and $\beta \leq p$.

6.2 The Class of Read-once Majority Formulas

In this section we use properties of amplification functions to obtain a polynomial-time algorithm that with high probability will exactly identify any read-once majority formula of logarithmic depth from random examples drawn according to a uniform distribution.

This type of formula is used in Schapire's [59] proof that a concept class is weakly learnable in polynomial time if and only if it is strongly learnable in polynomial time. That is, the hypothesis output by Schapire's boosting procedure can be viewed as a read-once majority formula whose variables have been replaced by hypotheses output by the weak learning algorithm. We also note that a read-once majority formula cannot in general be converted to a read-once Boolean formula over a {NOT, OR, AND} basis. We now show that the class of read-once majority formulas is not learnable in the distribution-free model.

Theorem 6.1 *The class of read-once majority formulas is not learnable in the distribution-free learning model (assuming the security of various cryptographic schemes).*

Proof: We exhibit a prediction-preserving reduction from NC^1 to this class, in the style of Pitt and Warmuth [54]; combining this with Kearns and Valiant's [36] result that NC^1 is not learnable (modulo cryptographic assumptions), proves the theorem.

Replace each AND gate of the NC^1 circuit by a MAJ gate with one input hard-wired to 0. Likewise, replace each OR gate of the NC^1 circuit by a MAJ gate with one input hard-wired to 1. Finally, apply Kearns et al.'s [35] technique of repeating each variable N times in the input to make the formula read-once. ■

Despite the hardness of this class in the general distribution-free framework, we show that the class is nevertheless learnable when examples are chosen from the uniform distribution. The algorithm is simple, and consists of two phases. In the first phase, we determine the relevant variables (whether they occur in the formula), their signs (whether they are negated or not), and their levels. To achieve this goal, for each variable, we hard-wire its value to 1 and estimate the amplification of the induced function at $\frac{1}{2}$ using examples drawn randomly from the uniform distribution. (Here, by hard-wiring a variable to 1, we really mean that we apply a filter that only lets through examples for which that variable is 1.) If the variable is relevant, then with high probability this estimate will be significantly smaller or greater than $\frac{1}{2}$, depending on whether the variable occurs negated or non-negated in the formula; otherwise, this estimate will be near $\frac{1}{2}$. Furthermore, the level of the given variable can be determined, with high probability, from the amount by which the amplification of the induced function varies from $\frac{1}{2}$.

In the second phase of the algorithm, we construct the formula. More precisely, we first construct the bottom level of the formula, and then recursively construct the remaining levels. To construct the bottom level of the formula, we begin by finding triples of variables that are inputs to the same bottom-level gate. To do this, for each triple of relevant variables that have the largest level number, we hard-wire the three variables to 1 and again estimate the amplification of the induced function from random examples. We show that with high probability we can determine whether the three variables all enter the same input-level gate based on this estimate. We now briefly

describe how the recursion works. Suppose that we are currently constructing level ℓ of the formula and we find that x_i, x_j, and x_k meet at the same level-ℓ gate. Then in the recursive call we replace x_i, x_j, and x_k by a level-$(\ell-1)$ meta-variable $y = \text{MAJ}(x_i, x_j, x_k)$. Since y is a known subformula its output on a random example can be easily computed. Furthermore since $\frac{1}{2}$ is the fixed point for the amplification function it follows that y is 1 with probability $\frac{1}{2}$. Thus for the recursive call we replace all triples of variables that meet at level ℓ by level-$(\ell-1)$ meta-variables, and we easily obtain our needed source of random examples drawn according to the uniform distribution on the new variable set from the original source of examples.

For the remainder of this section, we explore some of the properties of the amplification function of read-once majority formulas, leading eventually to a proof of the correctness of this algorithm.

The following observation can be proved by an easy induction on the depth of the formula:

Observation 6.1 *If f is a read-once majority formula, then $A_f(\frac{1}{2}) = \frac{1}{2}$.*

Proof: By induction on the depth t of formula f. When $t = 0$, the observation clearly holds. For the inductive step, let f_1, f_2 and f_3 be the function computed by the three subformulas obtained by deleting the output gate of f. Since, by the inductive hypothesis, $A_{f_i}(\frac{1}{2}) = \frac{1}{2}$ for $i = 1, 2, 3$ it follows that $f = \text{MAJ}(f_1, f_2, f_3) = 4\left(\frac{1}{2}\right)^3 = \frac{1}{2}$. Thus, $\frac{1}{2}$ is a fixed point for the amplification of any read-once majority formula. ∎

Our approach depends on the fact that the first derivative of A_f at $\frac{1}{2}$ is large, meaning that a slight perturbation of $\mathcal{D}^{(\frac{1}{2})}$ (i.e., the uniform distribution) tends to perturb the statistical behavior of the formula sufficiently to allow exact identification. See Figure 6-5 for a graph showing the amplification function for balanced read-once majority formulas of various depths.

We perturb $\mathcal{D}^{(\frac{1}{2})}$ by hard-wiring a small number of variables to be 1. We begin by considering the effect on the function's amplification of altering the probability with

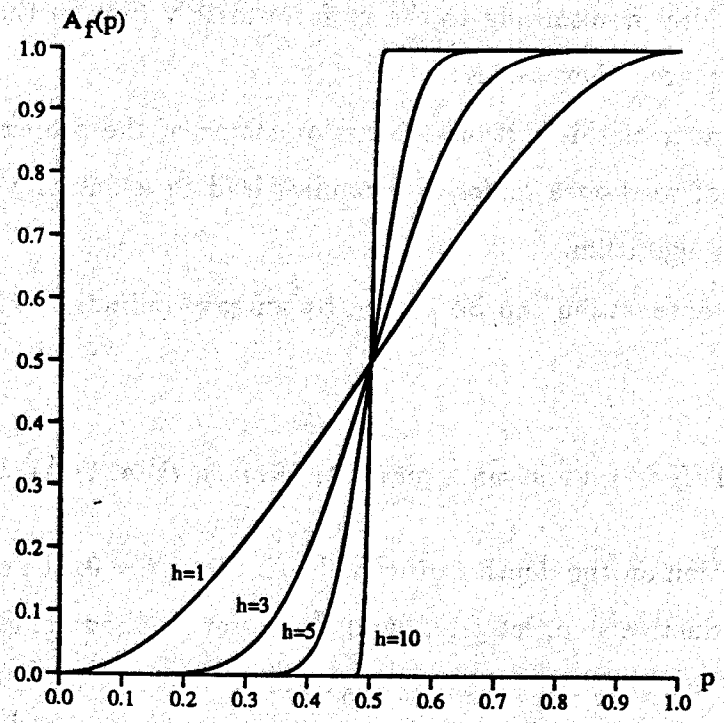


Figure 6-5: The amplification function for read-once majority formulas for complete ternary trees of depth h.

which one of the variables x_j is set to 1. This will be important in the analysis that follows.

Lemma 6.1 *Let f be a read-once majority formula, and let t be the level of some non-negated variable x_j. Then $A_{f|\{x_j\}\leftarrow q}\left(\frac{1}{2}\right) = q\left(\frac{1}{2}\right)^t + \frac{1}{2} - \left(\frac{1}{2}\right)^{t+1}$.*

Proof: By induction on t. When $t = 0$, the formula consists just of the variable x_j, and the lemma holds. For the inductive step, let f_1, f_2 and f_3 be the functions computed by the three subformulas obtained by deleting the output gate of f; thus, f is just the majority of f_1, f_2 and f_3. Note that x_j occurs in exactly one of these three subformulas—assume it occurs in the first. Since x_j occurs at level $t-1$ of this subformula, by the inductive hypothesis, $A_{f_1|\{x_j\}\leftarrow q}\left(\frac{1}{2}\right) = q\left(\frac{1}{2}\right)^{t-1} + \frac{1}{2} - \left(\frac{1}{2}\right)^t$, and since x_j does not occur in the other subformulas, $A_{f_i|\{x_j\}\leftarrow q}\left(\frac{1}{2}\right) = \frac{1}{2}$ for $i = 2, 3$. By a straightforward computation we compute that if a majority gate has inputs that are 1 with probability p_1, p_2, and p_3 then the output of the gate is one with probability $p_1p_2 + p_1p_3 + p_2p_3 - 2p_1p_2p_3$. Thus letting $p_1 = A_{f_1|\{x_j\}\leftarrow q}\left(\frac{1}{2}\right)$ and $p_2 = p_3 = \frac{1}{2}$, it follows that

$$\begin{aligned} A_{f|\{x_j\}\leftarrow q}\left(\tfrac{1}{2}\right) &= \frac{p_1}{2} + \frac{1}{4} \\ &= \tfrac{1}{2}\left(q\left(\tfrac{1}{2}\right)^{t-1} + \tfrac{1}{2} - \left(\tfrac{1}{2}\right)^t\right) + \tfrac{1}{4} \\ &= q\left(\tfrac{1}{2}\right)^t + \tfrac{1}{4} - \left(\tfrac{1}{2}\right)^{t+1} + \tfrac{1}{4} \\ &= q\left(\tfrac{1}{2}\right)^t + \tfrac{1}{2} - \left(\tfrac{1}{2}\right)^{t+1} \end{aligned}$$

completing the induction. ∎

It can now be seen how we use the amplification function to determine the relevant variables of f: if x_j is relevant then the statistical behavior of the output of f changes significantly when x_j is hard-wired to 1. Similarly, the sign and the level of each variable can be readily determined in this manner.

Theorem 6.2 *Let f be a read-once majority formula of depth h. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|\{x_j\}\leftarrow 1}\left(\frac{1}{2}\right)$ for some variable x_j, and assume that $|\hat{\alpha} - \alpha| < \left(\frac{1}{2}\right)^{h+2}$. Then*

- *x_j is relevant if and only if* $|\hat{\alpha} - \frac{1}{2}| > \left(\frac{1}{2}\right)^{h+2}$;
- *if x_j is relevant, then it occurs negated if and only if* $\hat{\alpha} < \frac{1}{2}$;
- *x_j occurs at level t if and only if* $\left(\frac{1}{2}\right)^{t+1} - \left(\frac{1}{2}\right)^{h+2} < |\hat{\alpha} - \frac{1}{2}| < \left(\frac{1}{2}\right)^{t+1} + \left(\frac{1}{2}\right)^{h+2}$.

Proof: This proof follows from straightforward calculations using Lemma 6.1. We first consider the case where the given variable is a *relevant non-negated* variable that occurs at level t. Using Lemma 6.1 and the assumption that $|\hat{\alpha} - \alpha| < \left(\frac{1}{2}\right)^{h+2}$ we get that:

$$\hat{\alpha} - \tfrac{1}{2} > \alpha - \tfrac{1}{2} - \left(\tfrac{1}{2}\right)^{h+2} = \left(\tfrac{1}{2}\right)^{t+1} - \left(\tfrac{1}{2}\right)^{h+2} \geq \left(\tfrac{1}{2}\right)^{h+2} > 0.$$

Likewise,

$$\hat{\alpha} - \tfrac{1}{2} < \alpha - \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{h+2} = \left(\tfrac{1}{2}\right)^{t+1} + \left(\tfrac{1}{2}\right)^{h+2}.$$

By symmetry we get that for the case in which the given variable is a *relevant negated* variable the above bounds for $\hat{\alpha} - \frac{1}{2}$ now apply for $\frac{1}{2} - \hat{\alpha}$. Finally, when the given variable is *not relevant* it follows from the assumption of the theorem and Observation 6.1 that $|\hat{\alpha} - \frac{1}{2}| < \left(\frac{1}{2}\right)^{h+2}$. The theorem follows immediately from these equations. ∎

Thus, if one estimates the value of the amplification function from a sample whose size is polynomial in 2^h, then with high probability one can determine which variables are relevant, as well as the sign and level of every relevant variable. Namely, using Hoeffding's inequality we can show that a sample of size $O\left(4^h(\ln \frac{1}{\delta} + \ln N)\right)$ is sufficient to ensure that with probability at least $1 - \delta$ all the above information is properly computed. We therefore assume henceforth that the level of every variable has been determined, and that all variables are relevant and non-negated.

More problematic is determining exactly how the variables are combined in f. A natural approach is to try tying pairs of variables to 1, and to again estimate the amplification of the induced function in the hopes that some structural information will be revealed. Recall that in the first phase of the algorithm we learn the level of each relevant variables. Thus it is sufficient to focus on the case in which variables at the *same level* are tied to 1. The following lemma, which is useful at a later point, shows that this natural approach fails.

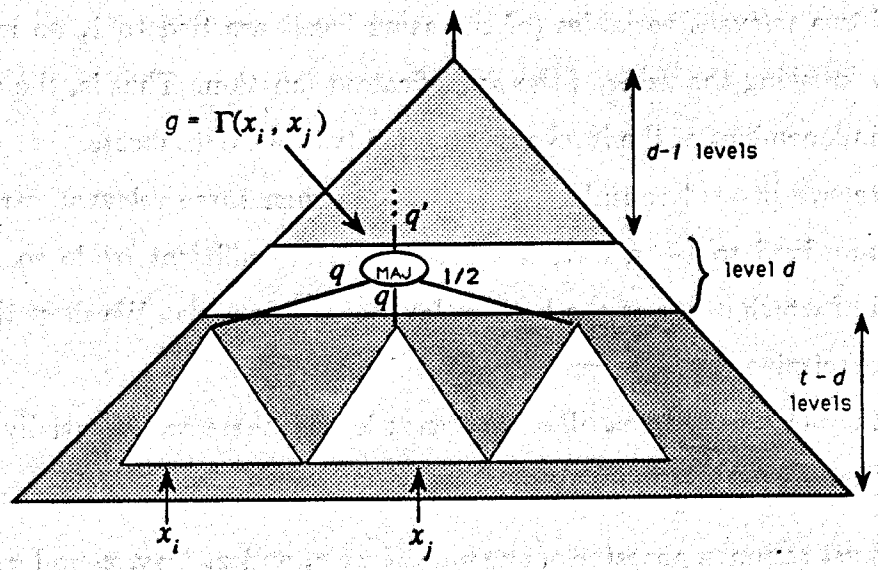


Figure 6-6: The subdivision of f when x_i and x_j meet at a level-d gate.

Lemma 6.2 *Let f be a read-once majority formula, and let x_i and x_j be two variables which occur at level t. Then $A_{f|\{x_i,x_j\}\leftarrow 1}\left(\frac{1}{2}\right) = \frac{1}{2} + \left(\frac{1}{2}\right)^t$, regardless of the depth d of $g = \Gamma(x_i, x_j)$.*

Proof: We split f into subformulas as shown in Figure 6-6. Consider the three subformulas feeding g. Two of these subformulas have depth $t-d$ and have one input fixed to 1. Applying Lemma 6.1, we see that each of these subformulas outputs 1 with probability $q = \frac{1}{2} + \left(\frac{1}{2}\right)^{t-d+1}$. Finally, by Observation 6.1 the third formula outputs 1 with probability $\frac{1}{2}$. Let q' be the probability that the output of g is 1. By a direct computation it can be seen that $q' = q^2/2 + q(1-q) + q^2/2 = q$. Finally, viewing the remaining formula as a read-once majority formula with one level-$(d-1)$ input modified to be 1 with probability q', we can again apply Lemma 6.1 to obtain:

$$\begin{aligned} A_{f|\{x_i,x_j\}\leftarrow 1}\left(\tfrac{1}{2}\right) &= q'\left(\tfrac{1}{2}\right)^{d-1} + \tfrac{1}{2} - \tfrac{1}{2}\left(\tfrac{1}{2}\right)^{d-1} \\ &= \left(\tfrac{1}{2}\right)^d + \left(\tfrac{1}{2}\right)^t + \tfrac{1}{2} - \left(\tfrac{1}{2}\right)^d \\ &= \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^t. \end{aligned}$$

■

Thus, if two relevant variables (of the same level) are tied to 1, no information is obtained by knowing the value of the amplification function. That is, the amplification function is independent of the level at which the two variables meet.

Therefore, we instead consider what happens when three relevant variables of the same level are fixed to 1—in fact, it turns out to be sufficient to do so for triples of variables all of which occur at the bottom level of the formula. We show that by doing so one can determine the full structure of the formula.

For each triple x_i, x_j and x_k all occurring at level t, there are essentially two cases to consider; either

1. the lowest common ancestor of one pair of x_i, x_j and x_k (say, x_i and x_j) is different than the lowest common ancestor of all three inputs (i.e. $\Gamma(x_i, x_j) \neq \Gamma(x_i, x_j, x_k)$); or

2. the lowest common ancestor of every pair of x_i, x_j and x_k is the lowest common ancestor of all three variables (i.e. $\Gamma(x_i, x_j) = \Gamma(x_i, x_k) = \Gamma(x_j, x_k) = \Gamma(x_i, x_j, x_k)$). We divide this case into two sub-cases:

   (a) x_i, x_j and x_k are inputs to the same gate so that $\Gamma(x_i, x_j, x_k)$ occurs at level t; or

   (b) the common ancestor of x_i, x_j and x_k occurs higher up in the formula so that $\Gamma(x_i, x_j, x_k)$ occurs at some level $d < t$.

We are interested in separating Case 2a from the other cases by estimating the amplification of the function when all three variables are tied to 1. This is sufficient to reconstruct the structure of the formula: if we can find three variables that are inputs to some gate g (and there always must exist such a triple), then we can essentially replace the subformula consisting of the three variables and the gate g by a new "meta-variable" whose value can easily be determined from the values of the original three variables Furthermore, since $\frac{1}{2}$ is a fixed point for all read-once majority formulas the meta-variables' statistics will be

just like that of the original variables. Thus, the total number of variables is reduced by two, and the rest of the formula's structure can be determined recursively.

The following two lemmas analyze the amplification of the function when three variables of the same level are hard-wired to 1 in both of the above cases. We begin with Case 2:

Lemma 6.3 *Let f be a read-once majority formula. Let x_i, x_j and x_k be three level-t inputs for which $\Gamma(x_i, x_j) = \Gamma(x_i, x_k) = \Gamma(x_j, x_k) = \Gamma(x_i, x_j, x_k) = g$. Let d be the level of g. Then $A_{f|\{x_i,x_j,x_k\}\leftarrow 1}\left(\frac{1}{2}\right) = \frac{1}{2} + 3\left(\frac{1}{2}\right)^{t+1} - \left(\frac{1}{2}\right)^{3t-2d+1}$.*

Proof: We first split f into subformulas, essentially as pictured in Figure 6-6 except that all three inputs to the MAJ gate g shown at level d are 1 with probability q. Each of the subformulas feeding g has a level-$(t-d)$ input fixed to 1. Thus, applying Lemma 6.1 we obtain $q = \frac{1}{2} + \left(\frac{1}{2}\right)^{t-d+1}$. Let q' be the probability of a 1 being output from g. Then, $q' = 3q^2 - 2q^3 = \frac{1}{2} + \left(\frac{3}{2}\right)^{t-d+2} - \left(\frac{1}{2}\right)^{3t-3d+2}$. Finally, we view the remaining formula as a formula with one input at level $d-1$ modified to be 1 with probability q' and apply Lemma 6.1 to obtain

$$\begin{aligned} A_{f|\{x_i,x_j,x_k\}\leftarrow 1}\left(\tfrac{1}{2}\right) &= q'\left(\tfrac{1}{2}\right)^{d-1} + \tfrac{1}{2} - \tfrac{1}{2}\left(\tfrac{1}{2}\right)^{d-1} \\ &= \left(\tfrac{1}{2}\right)^{d} + \tfrac{3}{4}\left(\tfrac{1}{2}\right)^{t-1} - \tfrac{1}{4}\left(\tfrac{1}{2}\right)^{3t-2d-1} + \tfrac{1}{2} - \left(\tfrac{1}{2}\right)^{d} \\ &= \tfrac{1}{2} + \tfrac{3}{2}\left(\tfrac{1}{2}\right)^{t} - \tfrac{1}{2}\left(\tfrac{1}{2}\right)^{3t-2d}. \end{aligned}$$

■

So, unlike the situation in which two variables of the same level are hard-wired to one, here the value of the amplification function depends on the level of the formula in which the three variables meet. However, when fixing x_i, x_j, and x_k it may not be the case that $\Gamma(x_i, x_j) = \Gamma(x_j, x_k) = \Gamma(x_i, x_k)$ (i.e., we may be in Case 1). The next lemma considers this case.

Lemma 6.4 *Let f be a read-once majority formula. Let x_i, x_j and x_k be three level-t inputs for which $g' = \Gamma(x_i, x_j) \neq \Gamma(x_i, x_j, x_k) = g$. Let d and ℓ be the levels of gates g and g'. Then $A_{f|\{x_i,x_j,x_k\}\leftarrow 1}\left(\frac{1}{2}\right) = \frac{1}{2} + 3\left(\frac{1}{2}\right)^{t+1}$.*

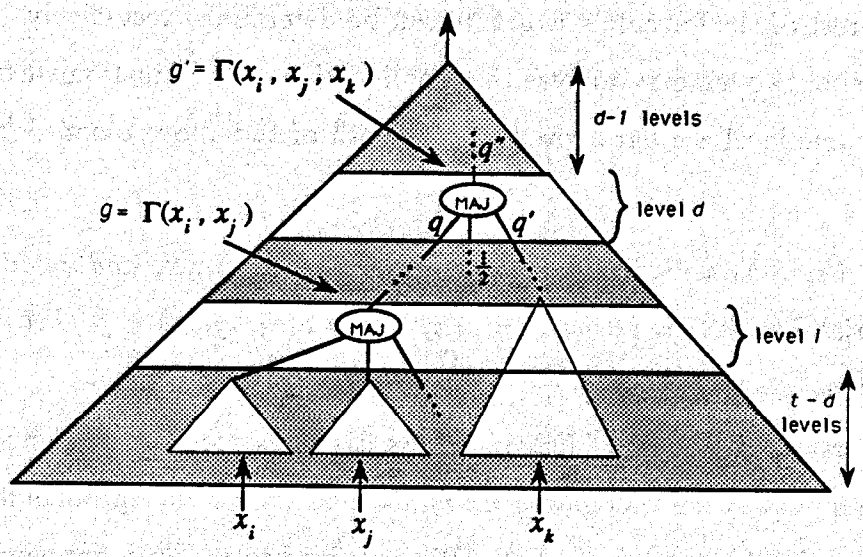


Figure 6-7: The subdivision of f when $\Gamma(x_i, x_j)$ is at level l and $\Gamma(x_i, x_j, x_k)$ is at level d.

Proof: Again we split f into subformulas, as shown in Figure 6-7. Consider the three subformulas feeding g. Clearly, one of these subformulas (the one containing none of x_i, x_j, or x_k) outputs 1 with probability $\frac{1}{2}$. Applying Lemma 6.1, we see that the subformula containing only x_k computes 1 with probability $q' = \frac{1}{2} + \left(\frac{1}{2}\right)^{t-d+1}$. Finally, applying Lemma 6.2, it can be shown that the output of the third subformula, which contains x_i, x_j and the gate g', computes 1 with probability $q = \frac{1}{2} + \left(\frac{1}{2}\right)^{t-d}$. Thus, the output of g is 1 with probability $q'' = (q + q')/2 = \frac{1}{2} + 3\left(\frac{1}{2}\right)^{t-d+2}$. Finally, the desired result follows by applying Lemma 6.1 as in the preceding lemmas. Namely, we now view the remaining formula as a formula of height $d-1$ with one input modified to be 1 with probability q''. Thus,

$$\begin{aligned} A_{f|\{x_i,x_j,x_k\}\leftarrow 1}\left(\tfrac{1}{2}\right) &= q''\left(\tfrac{1}{2}\right)^{d-1} + \tfrac{1}{2} - \tfrac{1}{2}\left(\tfrac{1}{2}\right)^{d-1} \\ &= \left(\tfrac{1}{2}\right)^{d} + \tfrac{3}{4}\left(\tfrac{1}{2}\right)^{t-1} + \tfrac{1}{2} - \left(\tfrac{1}{2}\right)^{d} \\ &= \tfrac{1}{2} + \tfrac{3}{2}\left(\tfrac{1}{2}\right)^{t}. \end{aligned}$$

■

Note that in this case the amplification function is independent of both ℓ and d.

Combining these lemmas, we now show that Case 2a can be separated from the other cases by estimating the function's amplification with triples of variables tied to 1.

Theorem 6.3 *Let f be a read-once majority formula. Let x_i, x_j and x_k be three level-t inputs. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|\{x_i,x_j,x_k\}\leftarrow 1}\left(\frac{1}{2}\right)$ for which $|\hat{\alpha}-\alpha| < 3\left(\frac{1}{2}\right)^{t+4}$. Then x_i, x_j and x_k are inputs to the same gate of f if and only if $\hat{\alpha} < \frac{1}{2} + 19\left(\frac{1}{2}\right)^{t+4}$.*

Proof: This proof is a straightforward consequence of the above lemmas. Combining Lemma 6.3 with the assumptions that $d = t$ and $|\hat{\alpha}-\alpha| < 3\left(\frac{1}{2}\right)^{t+4}$ we get that in Case 2a:

$$\hat{\alpha} < \alpha + 3\left(\tfrac{1}{2}\right)^{t+4} = \tfrac{1}{2} + 2\left(\tfrac{1}{2}\right)^{t+1} + 3\left(\tfrac{1}{2}\right)^{t+4} = \tfrac{1}{2} + 19\left(\tfrac{1}{2}\right)^{t+4}.$$

Likewise, when in Case 2b since $d \leq t-1$ we get that:

$$\hat{\alpha} > \alpha - 3\left(\tfrac{1}{2}\right)^{t+4} \geq \tfrac{1}{2} + 3\left(\tfrac{1}{2}\right)^{t+1} - \left(\tfrac{1}{2}\right)^{t+3} - 3\left(\tfrac{1}{2}\right)^{t+4} = \tfrac{1}{2} + 19\left(\tfrac{1}{2}\right)^{t+4}.$$

Finally, combining Lemma 6.4 with the assumption that $|\hat{\alpha}-\alpha| < 3\left(\frac{1}{2}\right)^{t+4}$ we get that in Case 1:

$$\hat{\alpha} > \alpha - 3\left(\tfrac{1}{2}\right)^{t+4} = \tfrac{1}{2} + 3\left(\tfrac{1}{2}\right)^{t+1} - 3\left(\tfrac{1}{2}\right)^{t+4} = \tfrac{1}{2} + 21\left(\tfrac{1}{2}\right)^{t+4}.$$

This completes the proof of the theorem. ■

We are now ready to state the main result of this section:

Theorem 6.4 *There exists an algorithm A that, given $h, N, \delta > 0$, and examples drawn from the uniform distribution on $\{0,1\}^N$, with probability $1-\delta$, exactly identifies any depth h read-once majority formula of N variables, at most n of which are relevant. The sample complexity of algorithm A is $O(4^h(\ln\frac{1}{\delta} + \ln N))$ and the time complexity is $O((n^3+N)4^h(\ln\frac{1}{\delta} + \ln N))$.*

Proof: First, for each variable x_i, estimate the function's amplification with x_i hard-wired to 1. (We will ensure that, with high probability, this estimate is within $\left(\frac{1}{2}\right)^{h+2}$ of

the true amplification.) It follows from Theorem 6.2 that after this phase of the algorithm, with high probability we know which variables are relevant, and the sign and depth of each relevant variable. (So, we assume from now on that the formula is monotone.)

In the second phase of the algorithm, we build the formula level by level from bottom to top. To build the bottom level, for all triples of variables x_i, x_j, x_k that enter the bottom level, we estimate the amplification with x_i, x_j, and x_k hard-wired to 1. (We will ensure that, with high probability, this estimate is within $3\left(\frac{1}{2}\right)^{h+4}$ of the true amplification.) It follows from Theorem 6.3 that we can determine which variables enter the same bottom-level gates.

We want to recurse to compute the other levels; however, we cannot hard-wire too many variables without the filter requiring too many examples. The key observation is that on examples drawn from the uniform distribution, the output of any subformula is one with probability $\frac{1}{2}$. Thus, the inputs into any level are in fact distributed according to a uniform distribution. Since we compute the formula from bottom to top, the filter can just compute the value for the ***known*** levels to determine the inputs to the level currently being learned. Our algorithm is described in Figure 6-8. Given that, with high probability, the estimates for the amplification function have the needed accuracy, the proof of correctness follows from Theorems 6.2 and 6.3.

We now use a standard application of Hoeffding's inequality to compute size of the sample needed to accurately estimate the amplification functions. Since we want the probability that all estimates are "good" to be at least $1-\delta$, we require that each of the two phases has good estimates with probability at least $1-\delta/2$.

We begin by computing the sample complexity for the first phase of this algorithm. From Theorem 6.2 we know that if we compute our estimates for the amplification function to within a factor $\gamma = \left(\frac{1}{2}\right)^{h+2} = \frac{1}{4}\left(\frac{1}{2}\right)^{h}$ of the true value, then we can determine which variables are relevant and compute the sign and level of each relevant variable. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|\{x_i\}\leftarrow 1}\left(\frac{1}{2}\right)$. We first compute the number of examples m needed for which x_i is 1. We then compute the required size m' of the original sample so

Learn-Majority-Formula(N, h)
Let $m_1 \leftarrow 32 \cdot 4^h \left(\ln \frac{8}{\delta} + \ln N\right)$
Draw a set $\mathcal{E}$ of m_1 examples from the uniform distribution on $\{0,1\}^N$
For $1 \leq i \leq N$
    $\mathcal{E}' \leftarrow$ examples from $\mathcal{E}$ where $x_i = 1$
    $\hat{\alpha} \leftarrow$ fraction of $\mathcal{E}'$ that are positive
    if $|\hat{\alpha} - \frac{1}{2}| > \left(\frac{1}{2}\right)^{h+2}$
        then if $\hat{\alpha} > \frac{1}{2}$ then x_i is in the formula
            else $\overline{x_i}$ is in the formula
    $t(x_i) \leftarrow$ *compute-level*$(\hat{\alpha})$
Let $\mathcal{X}$ be the set of relevant literals
Let $m_2 \leftarrow 228 \cdot 4^h \left(\ln \frac{4}{\delta} + 3 \ln |\mathcal{X}|\right)$
if $m_2 > m_1$
    then $\mathcal{E} \leftarrow \mathcal{E} \cup \{m_2 - m_1$ examples drawn from uniform dist on $\{0,1\}^N\}$
Build-Formula$(h, \mathcal{X}, \mathcal{E})$

Build-Formula$(t, \mathcal{X}, \mathcal{E})$
For all triples x_i, x_j, x_k such that $t(x_i) = t(x_j) = t(x_k) = t$
    $\mathcal{E}' \leftarrow$ examples from $\mathcal{E}$ where $x_i = x_j = x_k = 1$
    $\hat{\alpha} \leftarrow$ fraction of $\mathcal{E}'$ that are positive
    if $\hat{\alpha} < \frac{1}{2} + 19\left(\frac{1}{2}\right)^{t+4}$
        then x_i, x_j, x_k meet at level-t gate g
            Replace x_i, x_j, x_k by $x_i \leftarrow$ output from g
            $t(x_i) \leftarrow t - 1$
if $t > 1$
    then *Build-Formula*$(t-1, \mathcal{X}, \mathcal{E})$

Figure 6-8: Algorithm for exactly identifying read-once majority formulas of depth h. The procedure *compute-level*$(\hat{\alpha})$ computes the level associated with $\hat{\alpha}$ as given by Theorem 6.2.

that with high probability a sample of size m passes through the filter.

Using Hoeffding's inequality as given in Equation 6.1 we get that

$$\begin{aligned}\Pr[|\hat{\alpha}-\alpha| \geq \gamma] &\leq \Pr[\hat{\alpha} \geq \alpha+\gamma] + \Pr[\hat{\alpha} \leq \alpha-\gamma] \\ &\leq 2e^{-2m\gamma^2}.\end{aligned}$$

We want the probability of obtaining any bad estimates to be at most $\delta/4$. (The remainder of the $\delta/2$ probability of error allocated to the first phase is needed for the case in which too much is filtered out of the sample.) Furthermore, we need to divide this $\delta/4$ probability of error among the N estimates that must be made. Thus we want

$$2e^{-2m\gamma^2} \leq \frac{\delta}{4N}.$$

Solving for m we get that a filtered sample of size

$$\begin{aligned}m &\geq \left(\ln\frac{8}{\delta} + \ln N\right)/(2\gamma^2) \\ &= 8\cdot 4^h\left(\ln\frac{8}{\delta} + \ln N\right)\end{aligned}$$

is sufficiently large to ensure all estimates are good with probability at least $1-\delta/4$.

Now we compute the number of examples m' required so that with probability at least $1-\delta/4$ the filtered sample has size at least m. The expected number of examples to pass through the filter is $m'/2$ since we are only hard-wiring one variable. Thus if we draw $m' = 4m$ examples, it follows from Hoeffding's inequality as given in Equation 6.3 that

$$\Pr[\text{number examples passing through the filter} < m] \leq e^{-2m'(\frac{1}{4}-\frac{1}{2})^2} = e^{-m'/8}.$$

Setting $e^{-m'/8} \leq \delta/4$ we get that a samples of size $m' = \max\{4m, 8\ln\frac{4}{\delta}\}$ is sufficiently large.

We now use the above technique to compute the sample complexity for the second phase of the algorithm. From Theorem 6.3 we know that if we estimate the amplification function to within $\gamma = 3\left(\frac{1}{2}\right)^{h+4}$ then we can determine if any triple of variables meet

at a bottom-level gate. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|\{x_i,x_j,x_k\}\leftarrow 1}\left(\frac{1}{2}\right)$. As above, we first compute the number of examples m needed after filtering, and then compute the required size of the original sample m' so that, with probability at least $\delta/2$, all estimates are good.

Since we must compute $\binom{n}{3} \leq n^3/6$ estimates in the second phase, we want

$$2e^{-2m\gamma^2} \leq \frac{6\delta}{4n^3}.$$

Solving for m we get that

$$m \geq \frac{\ln\frac{4}{3\delta} + 3\ln n}{2\gamma^2}.$$

Since $\gamma = 3\left(\frac{1}{2}\right)^{h+4}$ we get that a filtered sample of size $m = \frac{128}{9}\cdot 4^h(\ln\frac{4}{3\delta} + 3\ln n)$ is sufficiently large to ensure a probability of at most $\delta/4$ of any bad estimates in the second phase.

As above, we must now compute the number of examples m' required so that with probability at least $1-\delta/4$ the filtered sample has size at least m. Since we are hard-wiring three variables here, the expected number of examples to pass through the filter is $m'/8$. Thus if we draw $m' = 16m$ examples, it follows from Hoeffding's inequality as given in Equation 6.3 that

$$\Pr[\text{number examples passing through the filter} \leq m] \leq e^{-2m'(\frac{1}{16}-\frac{1}{8})^2} = e^{-m'/128}.$$

Setting $e^{-m'/128} \leq \delta/4$ we get that a samples of size $m' = \max\{16m, 128\ln\frac{4}{\delta}\}$ examples to ensure that the probability of too few examples passing through the filter is at most $\delta/4$.

Combining the bounds from both phases we get that a sample of size

$$228\cdot 4^h\left(\ln\frac{8}{\delta} + \ln N + 3\ln n\right) = O\left(4^h\left(\ln\frac{1}{\delta} + \ln N\right)\right)$$

is sufficiently large to ensure that with probability at least $1-\delta$ all estimates are good.

Finally, the time complexity follows from the fact that in the first phase N estimates are needed and in the second phase $O(n^3)$ estimates are needed. ■

It follows immediately that any read-once majority formula of depth $O(\lg N)$ can be exactly identified in polynomial time.

Corollary 6.1 *There exists an algorithm A that, given any $h, N, \delta > 0$, and examples drawn from the uniform distribution on $\{0,1\}^N$, with probability $1-\delta$, exactly identifies any read-once majority formula over $\{x_1, \ldots, x_N\}$ of logarithmic depth. The time and sample complexity of A are polynomial in N and $\ln 1/\delta$.*

6.3 The Class of Read-once Positive NOR Formulas

In this section we use the properties of the amplification function to obtain a polynomial-time algorithm that with high probability will exactly identify any read-once positive NOR formula of logarithmic depth from $\mathcal{D}^{(\lambda)}$ where λ is the constant $(3-\sqrt{5})/2 \approx 0.382$. Observe that λ is $1-\phi$ where ϕ is the "golden ratio" $(\sqrt{5}-1)/2$.

It is interesting to compare this result with what is known about learning this class of formulas in other models. It follows from the results of Kearns and Valiant [36], and Pitt and Warmuth [53] that learning this class of formulas is hard in the distribution-free model (under cryptographic assumptions). Thus, there exist distributions that reveal essentially no information about the formula that is useful for prediction. If one views the sampling of the distribution $\mathcal{D}^{(\lambda)}$ as a form of non-adaptive "random membership queries", our result can also be compared with the algorithm of Angluin, Hellerstein and Karpinski [5] which uses membership queries that are considerably more complicated and are highly dependent on the target concept; on the other hand, their algorithm can be used to identify a much broader class of formulas.

We show that this class of formulas is learnable when examples are chosen from a distribution in which each variable is one with probability λ. The basic structure of the algorithm is just like that of the preceding algorithm for identifying read-once majority formulas. In the first phase of the algorithm, we determine the relevant variables and their depths by hard-wiring each variable to 1, and estimating the amplification of the induced

function at λ using random examples from $\mathcal{D}^{(\lambda)}$. In the second phase of the algorithm, we construct the formula by finding pairs of variables that are direct inputs to a bottom-level gate. Here, we show that this is possible by hard-wiring pairs of variables to 1 and estimating the function's amplification. After learning the structure of the bottom level of the formula, we again are able to construct the remaining levels recursively.

We turn now to a discussion of some of the properties of the amplification function of read-once positive NOR formulas; we conclude with a proof of the correctness of our algorithm.

The following observation can be proved by an easy induction on the depth of the formula:

Observation 6.2 *If f is a read-once positive* NOR *formula, then $A_f(\lambda) = \lambda$.*

Proof: By induction on the depth t of the formula f. When $t = 0$, the observation clearly holds. For the inductive step, let f_1 and f_2 be the functions computed by the two subformulas obtained by deleting the output gate of f. Since, by the inductive hypothesis $A_{f_1}(\lambda) = \lambda$ and $A_{f_2}(\lambda) = \lambda$ it follows that $f = \text{NOR}(f_1, f_2) = (1-\lambda)^2 = \lambda$. Thus, λ is the fixed point for the amplification of any read-once positive NOR formula. ■

Throughout this section, we use the following identities to simplify formulas for the amplification function.

1. $(1-\lambda)^2 = \lambda$.
2. $2\lambda - \lambda^2 = 1 - \lambda$.

Once again, our approach depends on the fact that the first derivative of A_f at λ is large, meaning that a slight perturbation of $\mathcal{D}^{(\lambda)}$ tends to perturb the statistical behavior of the formula sufficiently to allow exact identification. See Figure 6-9 for a graph showing the amplification for balanced read-once positive NOR formulas of various depths.

Lemma 6.5 *Let f be a read-once positive* NOR *formula, and let t be the level of some non-negated variable x_j. Then $A_{f|\{x_j\}\leftarrow q}(\lambda) = q(\lambda-1)^t + \lambda - \lambda(\lambda-1)^t$.*

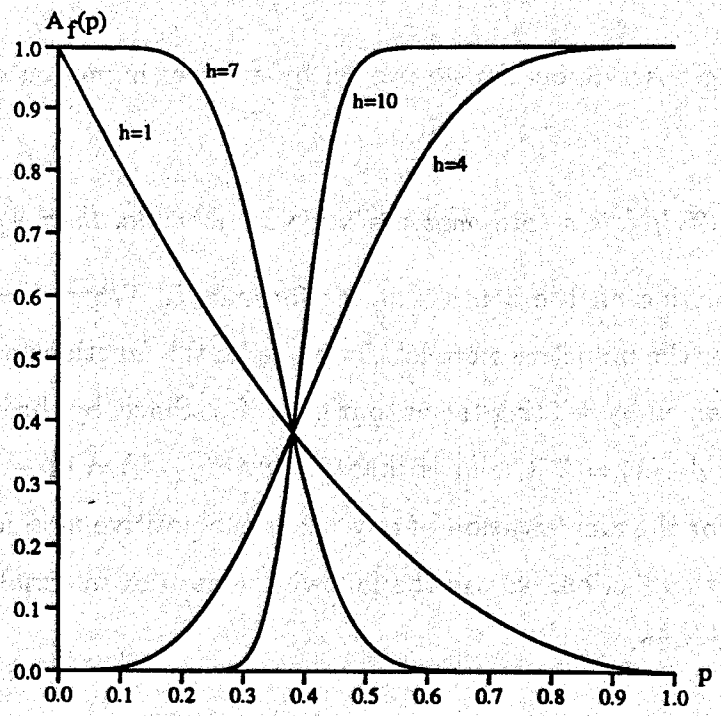


Figure 6-9: The amplification function for read-once positive NOR formulas for complete binary trees of depth h.

Proof: By induction on t as in Lemma 6.1. When $t = 0$, the formula consists just of the variable x_j, and the lemma holds. For the inductive step, let f_1 and f_2 be the functions computed by the two subformulas obtained by deleting the output gate of f; thus, $f = \overline{(f_1 \vee f_2)}$. Note that x_j occurs in exactly one of these subformulas—assume it occurs f_1. Since x_j occurs at level $t-1$ of f_1, by the inductive hypothesis, $A_{f_1|\{x_j\}\leftarrow q}(\lambda) = q(\lambda-1)^{t-1} + \lambda - \lambda(\lambda-1)^{t-1}$, and since x_j does not occur in the other subformula, $A_{f_2|\{x_j\}\leftarrow q}(\lambda) = \lambda$. A NOR gate with inputs that are 1 with probability p_1 and p_2 has output that is 1 with probability $(1-p_1)(1-p_2)$, and thus

$$\begin{aligned} A_{f|\{x_j\}\leftarrow q}(\lambda) &= \left(1 - q(\lambda-1)^{t-1} - \lambda + \lambda(\lambda-1)^{t-1}\right)(1-\lambda) \\ &= (1-\lambda) + q(\lambda-1)^t - \lambda(1-\lambda) - \lambda(\lambda-1)^t \\ &= q(\lambda-1)^t + (1-\lambda)(1-\lambda) - \lambda(\lambda-1)^t \\ &= q(\lambda-1)^t + \lambda - \lambda(\lambda-1)^t \end{aligned}$$

completing the induction. ∎

Since $(\lambda-1)^t = (-1)^t(1-\lambda)^t$, when hard-wiring x_j to 1 we have that

$$A_{f|\{x_j\}\leftarrow 1}(\lambda) = \lambda - (\lambda-1)^{t+1} = \lambda + (1-\lambda)^{t+1}(-1)^t. \tag{6.4}$$

Thus hard-wiring an even-leveled input to 1 increases the amplification and hard-wiring on odd-leveled input to 1 decreases the amplification function. To give some intuition explaining this behavior, consider the correspondence described in Section 6.1 between read-once positive NOR formulas and leveled OR/AND/ formulas. An even-leveled input corresponds to an input to an OR and thus hard-wiring that input to 1 clearly increases the amplification. However, an odd-leveled input corresponds to an input that is first negated and then enters an AND gate—thus this case corresponds to hard-wiring the input to an AND gate to 0 which clearly decreases the amplification function.

As we saw in the last section, the amplification function can be used to determine the relevant variables of f: if x_j is relevant then the statistical behavior of the output of

f changes significantly when x_j is hard-wired to 1. Similarly, the level of each variable can be computed in this manner.

Theorem 6.5 *Let f be a read-once positive* NOR *formula of depth h. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|\{x_j\}\leftarrow 1}(\lambda)$ for some variable x_j, and assume that $|\hat{\alpha} - \alpha| < \lambda(1-\lambda)^h/2$. Then*

- *x_j is relevant if and only if $|\hat{\alpha} - \lambda| > \lambda(1-\lambda)^h/2$;*
- *x_j occurs at level t if and only if $(1-\lambda)^{t+1} - \lambda(1-\lambda)^h/2 < |\hat{\alpha} - \lambda| < (1-\lambda)^{t+1} + \lambda(1-\lambda)^h/2$.*

Proof: We prove this theorem using straightforward calculations from Lemma 6.5. Let the given variable be a *relevant* variable that occurs at level t. Combining equation (6.4) and the assumption that $|\hat{\alpha} - \alpha| < \lambda(1-\lambda)^h/2$ we get that:

$$\begin{aligned}
|\hat{\alpha} - \lambda| &> |\hat{\alpha} - \lambda| - \lambda(1-\lambda)^h/2 \\
&= |\lambda - (\lambda-1)^{t+1} - \lambda| - \lambda(1-\lambda)^h/2 \\
&= (1-\lambda)^{t+1} - \lambda(1-\lambda)^h/2 \\
&\geq (1-\lambda)^{h+1} - \lambda(1-\lambda)^h/2 \\
&\geq \lambda(1-\lambda)^h - \lambda(1-\lambda)^h/2 \\
&= \lambda(1-\lambda)^h/2.
\end{aligned}$$

Likewise,

$$|\hat{\alpha} - \lambda| < |\hat{\alpha} - \lambda| + \lambda(1-\lambda)^h/2 = (1-\lambda)^{t+1} + \lambda(1-\lambda)^h/2.$$

Finally, when x_j is not relevant it follows from the assumption of the theorem and Observation 6.2 that

$$|\hat{\alpha} - \lambda| < |\hat{\alpha} - \lambda| + \lambda(1-\lambda)^h/2 = \lambda(1-\lambda)^h/2.$$

The lemma immediately follows from these equations. ∎

Once again, since we compute the level of all relevant variables in the first phase, we need only consider tying to 1 pairs of variables that enter the same level. We now

consider the effect on the amplification function of fixing two such inputs. Unlike the case of read-once majority formulas, measuring the amplification of the function when pairs of variables are tied to 1 reveals a great deal of information about the structure of the formula. In particular, the value of the amplification function when two level-t variables, x_i and x_j, are tied to 1 depends critically on the depth of $\Gamma(x_i, x_j)$.

The following lemma analyzes the amplification of the formula induced by hard-wiring two level-t variables to 1.

Lemma 6.6 *Let f be a read-once positive* NOR *formula, and let x_i and x_j be two variables which occur at level t for which $g = \Gamma(x_i, x_j)$ is at level d. Then $A_{f|\{x_i,x_j\}\leftarrow 1}(\lambda) = \lambda + \lambda^{t-d+1}(\lambda-1)^{d-1} - 2(\lambda-1)^{t+1}$.*

Proof: We split f into subformulas in the same manner as in the proof of Lemma 6.2. Consider the two subformulas feeding $\Gamma(x_i, x_j)$. Each of these subformulas has depth $t-d$ and has one input fixed to 1. Applying Lemma 6.5, we see that each subformula outputs 1 with probability $q = \lambda - (\lambda-1)^{t-d+1}$. Let q' be the probability that the output of $\Gamma(x_i, x_j)$ is 1. By a direct computation it can be seen that

$$\begin{aligned} q' &= (1-q)^2 \\ &= 1 - 2(\lambda - (\lambda-1)^{t-d+1}) + \lambda^2 - 2\lambda(\lambda-1)^{t-d+1} + (\lambda-1)^{2(t-d+1)} \\ &= 1 - 2\lambda + \lambda^2 + 2(\lambda-1)^{t-d+1} - 2\lambda(\lambda-1)^{t-d+1} + \lambda^{t-d+1} \\ &= \lambda + \lambda^{t-d+1} + 2(\lambda-1)^{t-d+1}(1-\lambda) \\ &= \lambda + \lambda^{t-d+1} - 2(\lambda-1)^{t-d+2}. \end{aligned}$$

Finally, viewing the remaining formula as a read-once positive NOR formula with one level-$(d-1)$ input modified to output 1 with probability q', we can again apply Lemma 6.5 to obtain:

$$\begin{aligned} A_{f|\{x_i,x_j\}\leftarrow 1}(\lambda) &= q'(\lambda-1)^{d-1} + \lambda - \lambda(\lambda-1)^{d-1} \\ &= \lambda(\lambda-1)^{d-1} + \lambda^{t-d+1}(\lambda-1)^{d-1} - 2(\lambda-1)^{t+1} + \lambda - \lambda(\lambda-1)^{d-1} \\ &= \lambda + \lambda^{t-d+1}(\lambda-1)^{d-1} - 2(\lambda-1)^{t+1}. \end{aligned}$$

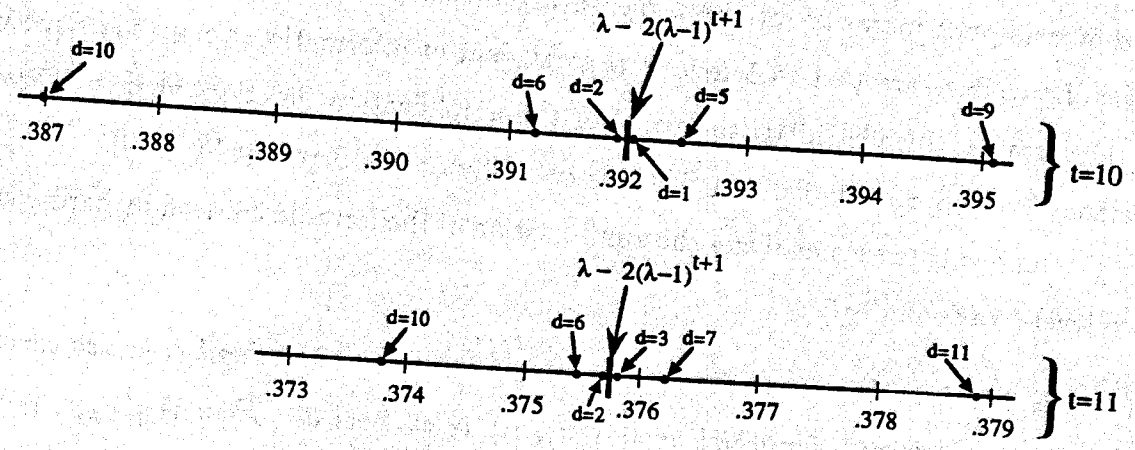


Figure 6-10: A graph plotting $\alpha_{d,t} = A_{f|\{x_i,x_j\}\leftarrow 1}(\lambda)$ where t is the level of x_i and x_j, and d is the level of $\Gamma(x_i, x_j)$. Recall that $\lambda \approx 0.382$.

■

To illustrate how the value of $\alpha_{d,t} = A_{f|\{x_i,x_j\}\leftarrow 1}(\lambda)$ depends on the level, t, of x_i and x_j and on the level, d, of $\Gamma(x_i, x_j)$ in Figure 6-10 we plot $\alpha_{d,t}$ for several values of d and t.

Using the same ideas seen in the last section, we now prove that given a "good" estimate of the amplification function, one can determine which variables meet at bottom-level gates.

Theorem 6.6 *Let f be a read-once positive* NOR *formula. Let x_i and x_j be two level-t inputs. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|\{x_i,x_j\}\leftarrow 1}(\lambda)$ for which $|\hat{\alpha} - \alpha| < \lambda(1-\lambda)^{t+1}/2$. Then x_i and x_j are inputs to the same bottom-level gate of f if and only if*

$$|\hat{\alpha} - \lambda + 2(\lambda-1)^{t+1}| > (1-\lambda)^{t+1} - \lambda(1-\lambda)^{t+1}/2.$$

Proof: This proof is a straightforward consequence of the above lemmas. Let $\alpha_{t,t} = A_{f|\{x_i,x_j\}\leftarrow 1}(\lambda)$ for the case in which $\Gamma(x_i, x_j)$ is at level t and let $\alpha_{d,t} = A_{f|\{x_i,x_j\}\leftarrow 1}(\lambda)$ for the case in which $\Gamma(x_i, x_j)$ is at level $d < t$. From Lemma 6.6 we have that

$$\alpha_{t,t} = \lambda + (\lambda-1)^{t+1} - 2(\lambda-1)^{t+1}.$$

Thus we have that

$$|\alpha_{t,t} - \lambda + 2(\lambda - 1)^{t+1}| = (1-\lambda)^{t+1}. \tag{6.5}$$

Likewise, we compute that

$$\alpha_{d,t} = \lambda - 2(\lambda-1)^{t+1} - \lambda^{t-d+1}(1-\lambda)^{d-1}(-1)^d.$$

Thus,

$$|\alpha_{d,t} - \lambda + 2(\lambda-1)^{t+1}| = \lambda^t \left(\frac{1-\lambda}{\lambda}\right)^{d-1}.$$

Since $\frac{1-\lambda}{\lambda} > 1$, $d \leq t-1$, it follows that:

$$|\alpha_{d,t} - \lambda + 2(\lambda-1)^{t+1}| \leq \lambda^t \left(\frac{1-\lambda}{\lambda}\right)^{t-2} = (1-\lambda)^{t+2}. \tag{6.6}$$

Thus for the case in which x_i and x_j meet at a bottom-level gate, from equation (6.5) and the assumption that $|\hat{\alpha} - \alpha| < \lambda(1-\lambda)^{t+1}/2$ it follows that

$$|\hat{\alpha} - \lambda + 2(\lambda-1)^{t+1}| > (1-\lambda)^{t+1} - \lambda(1-\lambda)^{t+1}/2.$$

Likewise, for the case in which x_i and x_j *do not* meet at a bottom-level gate, equation (6.6) and the assumption that $|\hat{\alpha} - \alpha| < \lambda(1-\lambda)^{t+1}/2$ gives that

$$\begin{aligned} |\hat{\alpha} - \lambda + 2(\lambda-1)^{t+1}| &< (1-\lambda)^{t+2} + \lambda(1-\lambda)^{t+1}/2. \\ &= (1-\lambda)^{t+1} - \lambda(1-\lambda)^{t+1}/2. \end{aligned}$$

This completes the proof of the theorem. ∎

We are now ready to state the main result of this section:

Theorem 6.7 *There exists an algorithm A that, given $h, N, \delta > 0$, and examples drawn from the distribution $\mathcal{D}^{(\lambda)}$, with probability $1-\delta$, exactly identifies any depth h read-once positive* NOR *formula of N variables, at most n of which are relevant. The sample complexity is $O((\frac{1}{\lambda})^h(\ln\frac{1}{\delta} + \ln N))$, and the time complexity is $O((n^2+N)(\frac{1}{\lambda})^h(\ln\frac{1}{\delta} + \ln N))$.*

Learn-NOR-formula(N, h)
1 Let $m_1 \leftarrow 4 \cdot \left(\frac{1}{\lambda}\right)^{h+3} \left(\ln \frac{8}{\delta} + \ln N\right)$
2 Draw a set $\mathcal{E}$ of m_1 examples from $\mathcal{D}^{(\lambda)}$
3 For $1 \leq i \leq N$
4 $\quad \mathcal{E}' \leftarrow$ examples from $\mathcal{E}$ where $x_i = 1$
5 $\quad \hat{\alpha} \leftarrow$ fraction of $\mathcal{E}'$ that are positive
6 $\quad$ **if** $|\hat{\alpha} - \lambda| > \lambda(1-\lambda)^h/2$
7 $\quad\quad$ **then** x_i is in the formula
8 $\quad t(x_i) \leftarrow$ *compute-level*$(\hat{\alpha})$
9 Let $\mathcal{X}$ be the set of relevant literals
10 Let $m_2 \leftarrow 4 \cdot \left(\frac{1}{\lambda}\right)^{h+5} \left(\ln \frac{4}{\delta} + 2 \ln n\right)$
12 **if** $m_2 > m_1$
13 $\quad$ **then** $\mathcal{E} \leftarrow \mathcal{E} \cup \{m_2 - m_1$ examples drawn from $\mathcal{D}^{(\lambda)}\}$
14 *Build-Formula*$(h, \mathcal{X}, \mathcal{E})$

Build-Formula$(t, \mathcal{X}, \mathcal{E})$
1 For all pairs x_i, x_j such that $t(x_i) = t(x_j) = t$
2 $\quad \mathcal{E}' \leftarrow$ examples from $\mathcal{E}$ where $x_i = x_j = 1$
3 $\quad \hat{\alpha} \leftarrow$ fraction of $\mathcal{E}'$ that are positive
4 $\quad$ **if** $|\hat{\alpha} - \lambda + 2(\lambda - 1)^{t+1}| > (1-\lambda)^{t+1} - \lambda(1-\lambda)^{t+1}/2$
5 $\quad\quad$ **then** x_i, x_j meet at level-t gate g
6 $\quad\quad\quad$ Replace x_i, x_j by $x_i \leftarrow$ output from g
7 $\quad\quad\quad t(x_i) \leftarrow t - 1$
8 **if** $t > 1$
9 $\quad$ **then** *Build-Formula*$(t-1, \mathcal{X}, \mathcal{E})$

Figure 6-11: Algorithm for exactly identifying read-once positive NOR formulas of depth h. The procedure *compute-level*$(\hat{\alpha})$ computes the level associated with $\hat{\alpha}$ as given by Theorem 6.5.

Proof: Our algorithm for this problem is very much like the one given for learning read-once majority formulas. First, for each variable x_i, estimate the function's amplification with x_i hard-wired to 1. (We will ensure that, with high probability, this estimate is within $\lambda(1-\lambda)^h/2$ of the true amplification.) It follows from Theorem 6.5 that after this phase of the algorithm, with high probability we know which variables are relevant, and the level of each relevant variable.

In the second phase of the algorithm, we recursively build the formula from bottom to top. To build the bottom level, for all pairs of variables x_i, x_j that enter the bottom level, we estimate the amplification with x_i and x_k hard-wired to 1. (We will ensure that, with high probability, this estimate is within $\lambda(1-\lambda)^{h+1}/2$ of the true amplification.) It follows from Theorem 6.6 that we can determine which variables enter the same bottom-level gates.

We want to recurse to compute the other levels; however, we cannot hard-wire too many variables without the filter requiring too many examples. Once again, we use the observation that on examples drawn from $\mathcal{D}^{(\lambda)}$, the output of any subformula is one with probability λ. Thus, the inputs into any level are in fact distributed according to $\mathcal{D}^{(\lambda)}$. Since we compute the formula from bottom to top, the filter can just compute the value for the *known* subformula to determine the inputs to the level currently being learned. Our algorithm is described in Figure 6-11.

Given that, with high probability, the estimates for the amplification function have the needed accuracy, the proof of correctness follows from Theorems 6.5 and 6.6. The bounds of the time and sample complexity are computed using Hoeffding's inequality just as in the proof of Theorem 6.4

We begin by computing the sample complexity for the first phase of this algorithm. From Theorem 6.5 we know that the estimates for the amplification function need only be computed to within $\gamma = \lambda(1-\lambda)^h/2$. Since we must compute N estimates in this phase, we want $2e^{-2m\gamma^2} \leq \frac{\delta}{4N}$. Solving for m we get that, with probability $1-\delta/4$, a

filtered sample of size

$$m = 2\left(\frac{1}{\lambda}\right)^{h+2}\left(\ln\frac{8}{\delta} + \ln N\right)$$

is sufficiently large. Finally, an initial sample of size

$$m' = \max\left\{\frac{2m}{\lambda}, \frac{2}{\lambda^2}\ln\frac{4}{\delta}\right\}$$

is sufficiently large to ensure that, with probability $1-\delta/4$, enough samples pass through the filter.

We now compute the sample complexity for the second phase of the algorithm. From Theorem 6.6 we know that the estimates for the amplification function need only be computed to within a factor of $\gamma = \lambda(1-\lambda)^{h+1}/2$. Since we must compute $\binom{n}{2} \leq n^2/2$ estimates in the second phase, we want $2e^{-2m\gamma^2} \leq 2\delta/4n^2 = \delta/2n^2$. Solving for m we get that, with probability $1-\delta/4$, a filtered sample of size

$$m = 2\left(\frac{1}{\lambda}\right)^{h+3}\left(\ln\frac{4}{\delta} + 2\ln n\right)$$

is sufficiently large. Since we are hard-wiring two variables here, we must take

$$m' = \max\left\{\frac{2m}{\lambda^2}, \frac{2}{\lambda^4}\ln\frac{4}{\delta}\right\}$$

examples to ensure that with probability $1-\delta/4$ enough pass through the filter.

Combining the bounds from both phases we get that a sample of size

$$4\left(\frac{1}{\lambda}\right)^{h+5}\left(\ln\frac{8}{\delta} + \ln N + 2\ln n\right) = O\left(\left(\frac{1}{\lambda}\right)^{h}\left(\ln\frac{1}{\delta} + \ln N\right)\right)$$

is sufficiently large to ensure that with probability at least $1-\delta$ all estimates are good.

Finally, the time complexity follows from the fact that in the first phase N estimates are needed and in the second phase $O(n^2)$ estimates are needed. ■

It follows immediately that any read-once positive NOR formula of depth at most $O(\lg N)$ can be exactly identified in polynomial time.

Corollary 6.2 *There exists an algorithm A that, given any $\delta > 0$, and examples drawn from $\mathcal{D}^{(\lambda)}$ over $\{0,1\}^N$, with probability $1-\delta$, exactly identifies any read-once positive* NOR *formula over $\{x_1, \ldots, x_N\}$ of logarithmic depth. The time and sample complexity of A are polynomial in N and $\ln 1/\delta$.*

6.4 The Class of Read-once Positive NAND Formulas

In this section we describe how our algorithm for learning any logarithmic-depth read-once positive NOR formula can be used to learn any logarithmic-depth read-once positive NAND formula. We obtain this result by giving a simple transformation from read-once positive NAND formulas to read-once positive NOR formulas.

It is easily shown using DeMorgan's law that if each input to a read-once positive NAND formula is negated and the output is also negated then the resulting formula is in fact a read-once positive NOR formula. Thus we get the following corollary to Theorem 6.7.

Corollary 6.3 *There exists an algorithm A that, given any $\delta > 0$, and examples drawn from $\mathcal{D}^{(1-\lambda)}$ over $\{0,1\}^N$, with probability $1-\delta$, exactly identifies any read-once positive* NAND *formula over $\{x_1, \ldots, x_N\}$ of logarithmic depth. The time and sample complexity of A are polynomial in N and $\ln 1/\delta$.*

Proof: Simply use the algorithm described in Figure 6-11 with the following modifications: On line 5 of *Learn-NOR-formula* and line 3 of *Build-Formula* let $\hat{\alpha}$ be the fraction of the example set that are *negative*. We also note that $\mathcal{D}^{(1-\lambda)}$ can be easily simulated from $\mathcal{D}^{(\lambda)}$ by just inverting all bits in each example. ∎

6.5 A Subclass of Read-once Monotone DNF Formulas

In this section we use the properties of the amplification function to exactly identify read-once monotone DNF formulas of t terms with ℓ literals per term. For the case that t and ℓ are provided, we give an algorithm that with high probability exactly identifies the formula from $\mathcal{D}^{(p^*)}$ where $p^* = (1-2^{-1/t})^{1/\ell}$. If t and ℓ are not known then we give an algorithm to learn this class of DNF when provided with $\mathcal{D}^{(p)}$ for all $0 < p < 1$.

Our algorithm for learning any DNF_ℓ^t formula is very much like the algorithms we presented in the preceding sections. In fact, it is a simpler algorithm since all relevant variables have depth two: all inputs enter an ℓ-input AND gate and then the output from each AND gate enters a single t-input OR gate. Thus in the first phase we determine (with high probability) which variables are relevant by hard-wiring each variable to 0 and estimating the amplification of the induced function. To reconstruct the formula, in the second phase we just determine which relevant variables enter the same AND gate; no recursion is needed. To achieve this goal we hard-wire all pairs of relevant variables to 0 and estimate the amplification of the induced function: with high probability this estimate varies significantly between the case in which both hard-wired variables enter the same gate and the case in which they enter different gates.

We now explore some properties of the amplification function for these class of formulas, ending in a proof of correctness of this algorithm. We begin by observing that $A_f(p) = 1 - (1 - p^\ell)^t$. As we shall see, a value of p for which small changes in the amplification function can be detected is the value p^* for which $A_f(p^*) = \frac{1}{2}$. One can easily compute that $p^* = (1 - 2^{-1/t})^{1/\ell}$.

We now consider the effect on the amplification function from hard-wiring a single variable to 0.

Lemma 6.7 *The amplification function* $A_{f|\{x_j\}\leftarrow 0}(p^*) = 1 - 2^{(1-t)/t}$.

Proof: Since x_j is hard-wired to 0, the term containing x_j must always be 0. Thus the amplification function of the induced function is equivalent to the amplification function for a formula with only $t - 1$ terms, each of which has ℓ literals per term. Thus

$$\begin{aligned} A_{f|\{x_j\}\leftarrow 0}(p^*) &= 1 - (1 - (p^*)^\ell)^{t-1} \\ &= 1 - (2^{-1/t})^{t-1}. \end{aligned}$$

■

Since setting a single input bit to 0 has a significant effect on the amplification function at p^*, we are able to detect which variables are relevant.

Theorem 6.8 *Let f be a DNF_t^t formula. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|\{x_j\}\leftarrow 0}(p^*)$ for some variable x_j and assume that $|\hat{\alpha} - \alpha| < \ln 2/4t$. Then x_j is relevant if and only if $\hat{\alpha} < \frac{1}{2} - \ln 2/4t$.*

Proof: We first consider the case in which x_j is relevant. Using Lemma 6.7 and the assumption that $|\hat{\alpha} - \alpha| < \ln 2/4t$ we get that:

$$\begin{aligned}\hat{\alpha} - \frac{1}{2} &< \alpha - \frac{1}{2} + \ln 2/4t \\ &= 1 - 2^{(1-t)/t} - \frac{1}{2} + \ln 2/4t \\ &= (1 - 2^{1/t})/2 + \ln 2/4t.\end{aligned}$$

Finally, since $2^{1/t} = e^{\ln 2/t} > 1 + \ln 2/t$ it follows that:

$$\hat{\alpha} < \frac{1}{2} - \ln 2/2t + \ln 2/4t = \frac{1}{2} - \ln 2/4t.$$

We now consider the case in which x_j is not relevant. It follows from the assumption of the theorem and the fact that $A_f(p^*) = \frac{1}{2}$ that $\hat{\alpha} > \frac{1}{2} - \ln 2/4t$. This completes the proof of the theorem. ■

For the second phase of the algorithm we must determine which of the relevant variables are in the same term of the formula (i.e., enter the same AND gate). We thus consider what happens when two relevant variables are fixed to 0. We will show that by doing so one can determine the full structure of the formula.

For each pair x_i and x_j of relevant variables, there are two cases that may occur: Either x_i and x_j enter the same AND gate, or x_i and x_j enter different AND gates. To reconstruct the formula, we need just separate these two cases. Once we know which variables enter each AND gate we are finished since all AND gates enter a single OR gate. The following two lemmas analyze the amplification function in each of the above cases. We begin with the case in which x_i and x_j meet at the same AND gate.

Lemma 6.8 *Let f be a DNF_t^t formula. Let x_i and x_j be two relevant inputs that enter the same gate. Then $A_{f|\{x_i,x_j\}\leftarrow 0}(p^*) = 1 - 2^{(1-t)/t}$.*

Proof: This situation is exactly like that in Lemma 6.7. ■

Next we consider the case in which x_i and x_j meet at different gates:

Lemma 6.9 *Let f be a DNF_ℓ^t formula. Let x_i and x_j be two relevant inputs that enter at different gates. Then $A_{f|\{x_i,x_j\}\leftarrow 0}(p^*) = 1 - 2^{(2-t)/t}$.*

Proof: Since x_i and x_j are hard-wired to 0, the term containing x_i and the term containing x_j must always be 0. Thus the amplification function of the induced function is equivalent to the amplification function for a formula with only $t-2$ terms, each of which has ℓ literals per term. Thus

$$A_{f|\{x_j\}\leftarrow 0}(p^*) = 1 - (1-(p^*)^\ell)^{t-2} = 1 - (2^{-1/t})^{t-2}.$$

■

Combining these lemmas, we now show that the two cases described above can be separated by estimating the function's amplification with pairs of variables tied to 0.

Theorem 6.9 *Let f be a DNF_ℓ^t formula. Let x_i and x_j be two relevant variables. Let $\hat{\alpha}$ be an estimate of $A_{f|\{x_i,x_j\}\leftarrow 0}(p^*)$ for which $|\hat{\alpha} - \alpha| < \ln 2/4t$. Then x_i and x_j are inputs to the same gate if and only if $\hat{\alpha} > \frac{1}{2} - \ln 2/4t$.*

Proof: This proof is a straightforward consequence of the above lemmas. We also use the following two inequalities: $2^{1/t} - 1 > \ln 2/t$ and $2^{(1-t)/t} \geq \frac{1}{2}$ for $t \geq 1$. Combining Lemma 6.8 with the assumption that $|\hat{\alpha} - \alpha| < \ln 2/4t$ gives, when x_i and x_j enter the same gate:

$$\begin{aligned} \hat{\alpha} &> \alpha - \ln 2/4t \\ &= 1 - 2^{(1-t)/t} - \ln 2/4t \\ &> \frac{1}{2} - \ln 2/4t. \end{aligned}$$

Likewise, combining Lemma 6.9 with the assumption of the theorem gives, when x_i and x_j enter different gates:

$$\hat{\alpha} < \alpha + \ln 2/4t$$

$$\begin{aligned} &= 1 - 2^{\frac{2-t}{t}} + \ln 2/4t \\ &= 1 - 2^{(1-t)/t} 2^{1/t} + \ln 2/4t \\ &\leq 1 - 2^{1/t}/2 + \ln 2/4t \\ &< 1 - (1 + \ln 2/t)/2 + \ln 2/4t \\ &= \frac{1}{2} - \ln 2/4t. \end{aligned}$$

This completes the proof of the theorem. ∎

We now apply these properties of the amplification function to obtain an algorithm to exactly identify a read-once monotone DNF_ℓ^t.

Theorem 6.10 *There is an algorithm that given $\mathcal{D}^{(p^*)}$ for $p^* = (1 - 2^{-1/t})^{1/\ell}$, with probability $1-\delta$, exactly identifies any read-once monotone DNF_ℓ^t formula of N variables, at most $n = t\ell$ of which are relevant. For any fixed value of ℓ, the sample complexity is $O(t^2(\ln 1/\delta + \ln N))$, and the time complexity is $O(t^2(N + n^2)(\ln 1/\delta + \ln N))$.*

Proof: Our algorithm is fairly simple. First, for each variable x_i, estimate the function's amplification with x_i hard-wired to 0. Given that the estimate is accurate enough, it follows from Theorem 6.8 that we can determine which variables are relevant. Then in the second phase of the algorithm we build the formula by determining which variables enter the same gate. From Theorem 6.9 we know we can do this from estimating the amplification function when each pair of variables is hard-wired to 0. Our algorithm is described in Figure 6-12.

Finally, we compute the size of the sample needed for the estimates to have the desired accuracy. From Theorems 6.8 and 6.9 we know that in both phases of the algorithm the estimates for the amplification function need only be computed to within a factor of $\gamma = \ln 2/4t$. Using Hoeffding's inequality we get that for the first phase a filtered sample of size $17t^2(\ln 8/\delta + \ln N)$ is sufficient; and in the second phase a filtered sample of size $17t^2(\ln 4/\delta + 2\ln n)$ is sufficient. Finally we observe that for a fixed ℓ, $p^* \leq 2^{-1/\ell}$ for all $t \geq 1$. Since, for fixed ℓ, we have a constant upperbound for p^* we know that an additional

Learn–DNF(t,ℓ,V)
1 Compute $p^* = (1 - 2^{-1/t})^{1/\ell}$
2 Draw a sufficiently large set $\mathcal{E}$ of examples from $\mathcal{D}^{(p^*)}$
3 For $1 \leq i \leq V$
4 $\quad \mathcal{E}' \leftarrow$ examples from $\mathcal{E}$ where $x_i = 0$
5 $\quad \hat{\alpha} \leftarrow$ fraction of $\mathcal{E}'$ that are positive
6 $\quad$ **if** $\hat{\alpha} < \frac{1}{2} - \ln 2/4t$
7 $\quad\quad$ **then** x_i is relevant
8 $\quad\quad$ **else** x_i is not relevant
9 For all pairs x_i, x_j of relevant variables
10 $\quad \mathcal{E}' \leftarrow$ examples from $\mathcal{E}$ where $x_i = x_j = 0$
11 $\quad \hat{\alpha} \leftarrow$ fraction of $\mathcal{E}'$ that are positive
12 $\quad$ **if** $\hat{\alpha} > \frac{1}{2} - \ln 2/4t$
13 $\quad\quad$ **then** x_i and x_j in the same term
14 $\quad\quad$ **else** x_i and x_j not in the same term

Figure 6-12: Algorithm for exactly identifying DNF_t^t formulas.

sample of size $O(\ln 1/\delta)$ suffices to ensure a large enough sample passes through the filter. Thus the sample complexity is $O(t^2(\ln\left(\frac{1}{\delta}\right) + \ln N)$.

Finally, the time complexity follows from the fact that in the first phase N estimates are needed and in the second phase $O(n^2)$ estimates are needed. ■

6.6 Universal Identification Sequences

In this section we describe an interesting consequence of our results. Observe that if we regard our algorithms' use of a *fixed* distribution as a form of "random" membership queries, then it is apparent that these queries are *non-adaptive*; each query is independent of all previous answers. In other words, our algorithms pick all membership queries before seeing the outcome of any. From this observation we can apply our results to prove the existence of polynomial-size *universal identification sequences* for classes of formulas, i.e., sequences of instances that distinguish all concepts from one another.

More formally, we define an *instance sequence* to be an unlabeled sequence of instances, and an *example sequence* to be a labeled sequence of instances. We say an instance sequence S *distinguishes* a concept c if the example sequence obtained by labeling S according to c distinguishes c from all other concepts in C_n. A *universal identification sequence* for a concept class C_n is an instance sequence that distinguishes *every* concept $c \in C_n$. In other words, a universal identification sequence gives a *single* teaching sequence (modulo different labelings) for all concepts in the class.

To provide some intuition for this result, we describe why our algorithm for learning read-once majority formulas implies the existence of a universal identification sequence for this class. Let C_n be all logarithmic-depth read-once majority formulas on n variables. Our main result of Section 6.2 yields that for any $c \in C_n$, with high probability a random example sequence exactly identifies c. By a simple counting argument it is easy to prove that $|C_n| \leq 2^{O(n \lg n)}$. Thus if $\delta = 2^{-kn \lg n}$ for a sufficiently large constant k, the probability that a random instance sequence fails to identify *any* $c \in C_n$ is strictly less than 1. Thus there exists some instance sequence that exactly identifies *all* $c \in C_n$.

The following theorem gives general conditions for when a probabilistic exact identification algorithm implies the existence of a polynomial-length universal identification sequence. We then apply this theorem to our algorithms of the previous sections.

Theorem 6.11 *Let C_n be a concept class such that $|C_n| \leq 2^{p(n)}$ for some polynomial $p(n)$. Let A be a algorithm that, given $\delta > 0$ and examples drawn from some fixed distribution $\mathcal{D}$, exactly identifies any $c \in C_n$ with probability $1-\delta$. Furthermore, suppose that the sample complexity of A is $q(n)\ln^k(1/\delta)$ for some polynomial $q(n)$ and constant k. Then there exists a polynomial-length universal identification sequence for C_n.*

Proof: The proof uses a standard probabilistic argument. Let S be the random example sequence drawn by algorithm A when learning some $c \in C$. Since algorithm A achieves exact identification of c with probability $1 - \delta$ we have that the probability, taken over all random example sequences for c, that A fails to exactly identify the *particular* target concept c is at most δ. Letting $\delta = 2^{-p(n)-1}$ it follows that the probability, taken over all

random instance sequences, that A fails to identify *any* target concept c is at most $1/2$. Thus, an instance sequence of length $q(n)(p(n)+1)^k$ drawn randomly from $\mathcal{D}$ exactly identifies any $c \in C_n$ with probability at least $1/2$. Therefore there must exist some instance sequence S of this length that exactly identifies all $c \in C_n$.

We now show that S *distinguishes* all $c \in C_n$. For $c \in C_n$, let S_c be the example sequence obtained by labeling S according to c. Suppose that S exactly identifies C yet there exists some $c' \in C_n$ ($c' \neq c$) that is also consistent with S_c. Since S exactly identifies all $c \in C_n$, it follows that c' must also be consistent with $S_{c'}$. However there must exist some $x \in S$ such that S_c and $S_{c'}$ giving a contradiction. Thus S is a universal identification sequence for C_n. ■

We note that if the hypothesis output by A can always be represented using at most $p(n)$ bits then $|C_n| \leq 2^{p(n)}$. We now apply this theorem to our exact identification algorithms to obtain the following corollary.

Corollary 6.4 *There exists polynomial-length universal identification sequences for the classes of logarithmic-depth read-once majority formulas and logarithmic-depth read-once positive* NOR *formulas.*

6.7 Conclusions and Open Problems

There are several other interesting consequences of our algorithms—the proofs of these results will appear in the long version of our paper [20].

We have proven that our algorithms are *robust* against a large amount of *random misclassification noise*. Specifically, if η_0 and η_1 represent the respective probabilities that outputs of 0 and 1 are misclassified, then a robust version of our algorithm can handle any noise rate for which $\eta_0 + \eta_1 \neq 1$; the sample size required increases only by an inverse quadratic factor in $|1 - \eta_0 - \eta_1|$.

We also have developed an algorithm that learns *any* (not necessarily logarithmic-depth) read-once majority formula in the distribution-free learning model against the

uniform distribution. To obtain this result we first show that the target formula can be well approximated by truncating the formula to have only logarithmic depth. We then generalize our algorithm for learning logarithmic-depth read-once majority formulas to handle such truncated formulas.

We expect that the general technique outlined here—that of inferring structural information about the target formula by observing its behavior on a critical distribution—can be applied to other classes of formulas. In this direction, we have shown that for any read-once formula f built from an arbitrary monotone gate whose amplification function has a fixed point p for $0 < p < 1$, and that also meets some symmetry conditions, our technique can be used to determine which variables are relevant and the depth of each variable from a polynomial-size sample drawn from $\mathcal{D}^{(p)}$. Although we suspect that the second phase of our technique will also work for these classes of formulas, we currently are unable to prove such a general result without knowing more about the given gate.

However, if this method is to be successful for a wider class of formulas (e.g. not necessarily read-once formulas), one of the main technical advances needed is a method of partially analyzing the probabilistic behavior of the formulas in the absence of complete independence among the inputs.

Chapter 7

Concluding Remarks

In this thesis, we studied several learning problems under various formal learning models. In the first part, we considered a mistake-bound model. To study how the the complexity of the learner's task depends on the sequence of queries presented to the learner, we presented an extended mistake-bound model in which the query sequence is selected by a helpful teacher, by the learner, by an adversary, or at random.

Using this extended mistake-bound model, we studied the problem of learning a relation between two sets of objects. If the relation has no structure, the learner cannot possibly make good predictions. First we imposed structure by restricting one set of objects to have relatively few "types". Next we considered the problem of learning a total order on a set of elements. That is, we restricted the predicate of the relation to be a total order. In studying the problem of learning a total order, we uncovered an interesting relationship between learning theory and randomized approximation schemes.

Next we applied the extended mistake-bound model to problems from the domain of concept learning. First we considered the question: what is the minimum number of examples a teacher must reveal to uniquely identify the target concept? As we saw, it is an interesting paradox that for many concept classes, the number of mistakes made with a helpful teacher may be worse than the number of mistakes made when the learner selects the sequence. In the case that the learner chooses the sequence of questions, we

showed that the number of mistakes can be significantly smaller than the number of queries needed.

Finally, we presented a new technique for *exactly identifying* read-once formulas from random examples. Our method was based on sampling the input-output behavior of the target formula on a probability distribution which is determined by the *fixed point* of the formula's *amplification function*. We presented algorithms for exactly identifying families of read-once formulas over various bases—including formulas of majority gates and a large subclass of formulas over the standard basis. Finally, we applied these results to prove the existence of polynomial-length *universal identification sequences* for large classes of formulas.

Appendix A

Warmuth's Algorithm for Learning Binary Relations

We now present Manfred Warmuth's algorithm for learning k-binary-relations; this algorithm achieves an $O(km + n\sqrt{m \lg k})$ mistake bound against an adversary-selected query sequence. Both the algorithm and the analysis are entirely due to Manfred.

A.1 The Algorithm

The algorithm is based on the weighted-majority algorithm of Littlestone and Warmuth [44]. Let n be the number of rows in the matrix to be learned and m be the number of columns. We say an entry (i, j) is *known* if the learner was previously presented that entry. We assume without loss of generality that the learner is never asked to predict the value of a known entry. The primary data structure for the algorithm is a weighted directed graph G on n vertices where vertex v_i corresponds to matrix row i. (The learner also keeps a matrix containing the known entries.) Initially, we let G be a complete graph with all edge weights set to 1.

We now describe how the learner makes a prediction for an unseen matrix entry (i, j). When making a prediction for (i, j), we say that v_k is *active* if entry (k, j) is known. To

make its prediction the learner takes a weighted majority of all active neighbors of v_i. After receiving the true value of (i,j), the learner sets the weight of the edge from v_i to v_k to 0 if $(k,j) \neq (i,j)$. Finally, when a mistake occurs the learner doubles the weight of the edge from v_i to v_k if $(k,j) = (i,j)$.

We note that each prediction can be made in time $O(n)$ since it just involves looking up the predictions made by the active neighbors of the node of interest and taking a majority vote.

A.2 The Analysis

We now analyze the number of mistakes made by this algorithm. We begin with some preliminary definitions and lemmas that are used in the proof.

Definition A.1 *A function $f : \mathbf{R}^+ \to \mathbf{R}^+$ is concave (respectively convex) over an interval D of $\mathbf{R}^+$ if for all $x \in D$, $f''(x) \geq 0$ ($f''(x) \leq 0$).*

In our analysis we will repeatedly use the following two standard lemmas [25, 50].

Lemma A.1 *Let f be a function from $\mathbf{R}^+$ to $\mathbf{R}^+$ that is concave over some interval D of $\mathbf{R}^+$. Let $q \in \mathbf{N}$, and let $x_1, x_2, \ldots, x_q \in D$. Then*

$$\sum_{i=1}^{q} x_i \leq S \Rightarrow \sum_{i=1}^{q} f(x_i) \geq q f(S/q).$$

Lemma A.2 *Let f be a function from $\mathbf{R}^+$ to $\mathbf{R}^+$ that is convex over some interval D of $\mathbf{R}^+$. Let $q \in \mathbf{N}$, and let $x_1, x_2, \ldots, x_q \in D$. Then*

$$\sum_{i=1}^{q} x_i \leq S \Rightarrow \sum_{i=1}^{q} f(x_i) \leq q f(S/q).$$

A basic observation that is essential to the analysis is that the edge weights cannot get too large. Namely we have the following lemma.

Lemma A.3 *Throughout the learning session, the weights on all edges adjacent to any node sum to at most $n-1$.*

Proof: We use a proof by induction. Clearly the base case holds. For the inductive step observe that whenever a mistake occurs, the amount of weight set to 0 is at least as large as the amount that is doubled. ■

When making a prediction for (i, j), we define the *force* of a mistake to be the number of rows of the same type as row i that were active (i.e. column j was known) when the mistake occurred. We now upper bound the number of mistakes of a given force that can occur for a given row type.

Lemma A.4 *For each row type r and force f there are at most m mistakes of force f.*

Proof: We use a proof by contradiction. Suppose that for row type r the learner makes $m+1$ force f mistakes. Then there must be two mistakes that occur for the same column. Suppose the first of these mistakes occurs when predicting (i, j) and the second occurs when predicting (i', j) where both rows i and i' are of type r. However, after making a force f mistake when predicting (i, j) that entry is known and thus the force of the (i', j) mistake must be at least $f + 1$ giving the desired contradiction. ■

We are now ready to proof the main result of this section.

Theorem A.1 *Warmuth's algorithm to learn k-binary-relations under an adversary-selected query sequence makes at most $O(km + n\sqrt{m \lg k})$ mistakes.*

Proof: The proof is structured as follows. Let f_i be the force of the ith mistake, and let t be the total number of mistakes made by the learner. We prove that for the set I that contains all edges connecting two vertices of the same type:

$$\sum_{i=1}^{t} f_i = \sum_{e \in I} \lg w(e) \tag{A.1}$$

where $w(e)$ is the weight of edge e. The remainder of the proof proceeds as follows. Next we obtain an upper bound for the right hand side of equation (A.1). We then prove a lower bound of the left hand side of equation (A.1). Finally, we combine these bounds to obtain an upper bound on the number of prediction mistakes made by Warmuth's algorithm.

Let n_i be the number of rows of type i. We now derive equation (A.1) given above. Observe that a force f mistake causes the weights on exactly f edges from I to be doubled. Thus after all mistakes have occurred:

$$2^{\sum_{i=1}^{t} f_i} = \prod_{e \in I} w(e)$$

where $w(e)$ is the final weight of edge e. Taking logarithms of both sides we obtain equation (A.1).

We now prove an upper bound on the right hand side of equation (A.1). Let $I(i)$ be the edges in I associated with row type i. Then

$$\sum_{e \in I} \lg w(e) \leq \sum_{i=1}^{k} \sum_{e \in I(i)} \lg w(e).$$

If there is only a single row of some type, then that row does not contribute anything to the sum and thus without loss of generality we assume that $n_i \geq 2$ for all i. From Lemma A.3 the sum of all inner weights adjacent to a vertex is at most $n-1$. Furthermore, the number of weights adjacent to a vertex of type i is $n_i - 1$. Since the function $f(x) = \lg x$ is easily shown to be convex and for all i, $n_i \geq 2$, if follows from Lemma A.2 that

$$\begin{aligned} \sum_{e \in I} \lg w(e) &\leq \sum_{i=1}^{k} n_i(n_i - 1) \lg \frac{n-1}{n_i - 1} \\ &\leq \sum_{i=1}^{k} n_i^2 \lg \frac{n-1}{n_i} \end{aligned} \tag{A.2}$$

Finally we apply Lemma A.2 and the observation that the function $f(x) = x^2 \lg \frac{n-1}{x}$ is convex over $[1, \infty]$ and the equality $\sum_{i=1}^{k} n_i = n$ to further bound equation (A.2) as follows:

$$\begin{aligned} \sum_{i=1}^{k} \sum_{e \in I(i)} \lg w(e) &\leq \sum_{i=1}^{k} n_i^2 \lg \frac{n-1}{n_i} \\ &\leq k \left(\frac{n}{k}\right)^2 \lg \frac{(n-1)k}{n} \\ &= \frac{n^2}{k} \left(\lg k + \lg \left(1 - \frac{1}{n}\right)\right) \\ &< \frac{n^2}{k} \lg k - \frac{n}{k} \lg e \end{aligned} \tag{A.3}$$

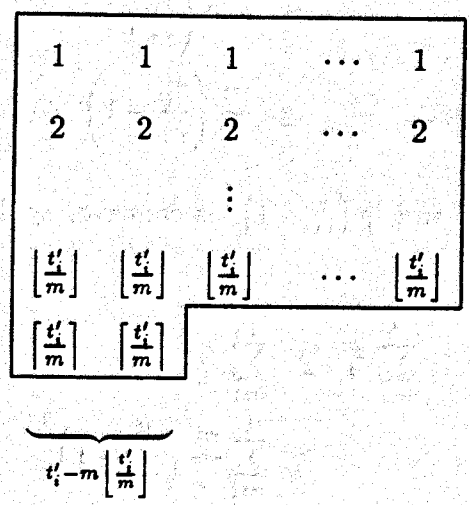


Figure A-1: First t'_i elements of the sequence $\langle 1 \rangle^m \langle 2 \rangle^m \langle 3 \rangle^m \ldots$.

where the final step follows from the inequality $e^{-\frac{1}{n}} > 1 - \frac{1}{n}$.

We now compute a lower bound for the left hand side of Equation (A.1). Let t' be the number of mistakes of non-zero force. By Lemma A.4 it follows that there are at most km mistakes of force 0 and thus

$$t \leq t' + km. \tag{A.4}$$

Let f'_i be the force of the ith non-zero force mistake, and let t'_i be the number of non-zero force mistakes made when predicting an entry in a row of type i. (Thus $\sum_{i=1}^{k} t'_i = t'$.) From Lemma A.4 it follows that the sum of all non-zero force mistakes of type i is lower bounded by the sum of the first t'_i elements of the sequence

$$\langle 1 \rangle^m \langle 2 \rangle^m \langle 3 \rangle^m \ldots.$$

The sum of the first t'_i elements of the sequence is $\sum_{i=1}^{m} \ell_i$ where ℓ_i is the length of column i in the matrix shown in Figure A-1. Let $S(x) = \sum_{i=1}^{x} i$. Since $\sum_{i=1}^{m} \ell_i = t'_i$ and $S(x)$ is concave, it follows by Lemma A.1 that

$$\sum_{i=1}^{t'} f'_i \geq \sum_{i=1}^{m} \ell_i$$

$$\begin{aligned} &\geq mS\left(\frac{t'_i}{m}\right) \\ &\geq \frac{m}{2}\left(\frac{t'_i}{m}-1\right)^2. \end{aligned}$$

Similarly, since $\sum_{i=1}^{k} t'_i = t'$ and $\frac{m}{2}\left(\frac{x}{m}-1\right)^2$ is concave, applying Lemma A.1 we obtain that

$$\begin{aligned} \sum_{i=1}^{t} f_i &\geq \sum_{i=1}^{t'} f'_i \\ &\geq \sum_{i=1}^{k} \frac{m}{2}\left(\frac{t'_i}{m}-1\right)^2 \\ &\geq \frac{km}{2}\left(\frac{t'}{km}-1\right)^2 \end{aligned} \tag{A.5}$$

Finally combining Equations (A.3) and (A.5) we get that

$$\frac{km}{2}\left(\frac{t'}{km}-1\right)^2 \leq \frac{n^2}{k}\lg k - \frac{n}{k}\lg e.$$

Solving for t' yields that $t' \leq \sqrt{2mn^2 \lg k - 2nm \lg e} + km$. Next we apply Equation (A.4) to get that $t \leq \sqrt{2mn}\sqrt{n \lg k - \lg e} + 2km$. Finally, simplifying the above inequality we get that

$$t \leq n\sqrt{2m \lg k} + 2km$$

giving the desired upper bound on the number of mistakes made by Warmuth's algorithm.

■

Bibliography

[1] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51:76–87, 1981.

[2] Dana Angluin. Types of queries for concept learning. Technical Report YALEU/DCS/TR-479, Yale University Department of Computer Science, June 1986.

[3] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[4] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[5] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning read-once formulas with queries. Technical Report UCB/CSD 89/528, University of California Berkeley Computer Science Division, 1989.

[6] J. Barzdin and R. Freivald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

[7] Gyora M. Benedek and Alon Itai. Learnability by fixed distributions. In *First Workshop on Computational Learning Theory*, pages 80–90. Morgan Kaufmann, August 1988.

[8] Avrim Blum. An $\tilde{O}(n^{0.4})$-approximation algorithm for 3-coloring. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 535–542, May 1989.

[9] Avrim Blum. Separating PAC and mistake-bound learning models over the Boolean domain. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, October 1990.

[10] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[11] Ravi B. Boppana. Amplification of probabilistic Boolean formulas. In *26th IEEE Symposium on Foundations of Computer Science*, pages 20–29, October 1985.

[12] A. Bundy, B. Silver, and D. Plummer. An analytical comparison of some rule-learning programs. *Artificial Intelligence*, 27:137–181, 1985.

[13] R. Carmichael. *Introduction to the Theory of Groups of Finite Order.* Dover Publications, New York, 1937.

[14] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[15] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, Cambridge, Massachusetts, 1990.

[16] M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for estimating the volumes of convex bodies. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 375–381, May 1989.

[17] Martin Dyer and Alan Frieze. On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing*, 17:967–974, 1988.

[18] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, September 1989.

[19] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[20] Sally A. Goldman, Michael J. Kearns, and Robert E. Schapire. Exact identification of circuits using fixed points of amplification functions. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, October 1990.

[21] Sally A. Goldman, Michael J. Kearns, and Robert E. Schapire. On the sample complexity of weak learning. In *Proceedings of the Third Annual Workshop on Computational Learning Theory.* Morgan Kaufmann, August 1990.

[22] Sally A. Goldman, Ronald L. Rivest, and Robert E. Schapire. Learning binary relations and total orders. Technical Report MIT/LCS/TM–413, MIT Laboratory for Computer Science, May 1990. A preliminary version is available in *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 46–51, 1989.

[23] T. Gradshteyn and I. Ryzhik. *Tables of Integral, Series, and Products.* Academic Press, New York, 1980. corrected and enlarged edition by A. Jeffrey.

[24] Thomas R. Hancock. Identifying μ-formula decision trees with queries. In *Proceedings of the Third Annual Workshop on Computational Learning Theory.* Morgan Kaufmann, August 1990.

[25] G.H. Hardy, J.E. Littlewood, and G. Pólya. *Inequalities.* Cambridge University Press, Cambridge, 1959.

[26] David Haussler. Learning conjunctive concepts in structural domains. Technical Report UCSC-CRL-87-1, Computer Research Laboratory, University of Santa Cruz, February 1987.

[27] David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[28] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 1990. To appear. A preliminary version is available in *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 42–55, 1988.

[29] David Haussler, Nick Littlestone, and Manfred Warmuth. Expected mistake bounds for on-line learning algorithms. Unpublished manuscript, 1988.

[30] David Haussler, Nick Littlestone, and Manfred K. Warmuth. Predicting {0,1}-functions on randomly drawn points. In *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science*, pages 100–109, 1988.

[31] David Helmbold, Robert Sloan, and Manfred K. Warmuth. Learning nested differences of intersection-closed concept classes. *Machine Learning*, 1990. Special issue for COLT 89, to appear. A preliminary version is available in *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 41–56, 1989.

[32] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[33] Mark Jerrum and Alistair Sinclair. Conductance and the rapid mixing property for Markov chains: the approximation of the permanent resolved. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 235–244, May 1988.

[34] Jeff Kahn and Michael Saks. Balancing poset extensions. *Order 1*, pages 113–126, 1984.

[35] Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. On the learnability of Boolean formulae. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 285–295, May 1987.

[36] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 433–444, May 1989.

[37] Michael J. Kearns. *The Computational Complexity of Machine Learning.* MIT Press, Cambridge, Massachusetts, 1990.

[38] Ming Li and Paul Vitanyi. A theory of learning simple concepts under simple distributions and average case complexity for the universal distribution. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 34–39, October 1989.

[39] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 574–579, October 1989.

[40] Nathan Linial, Yishay Mansour, and Ronald L. Rivest. Results on learnability and the Vapnik-Chervonenkis dimension. In *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science*, pages 120–129, October 1988.

[41] Nati Linial and Umesh Vazirani. Graph products and chromatic numbers. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 124–128, October 1989.

[42] Nicholas Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms.* PhD thesis, U. C. Santa Cruz, March 1989.

[43] Nick Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[44] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 256–261, October 1989.

[45] László Lovász. An algorithmic theory of numbers, graphs and convexity. In *CBMS-NSF Regional Conference Series on Applied Mathematics*, 1986.

[46] Wolfgang Maass and György Turán. On the complexity of learning from counterexamples. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 262–267, October 1989.

[47] Peter Matthews. Generating a random linear extension of a partial order. Unpublished manuscript, 1989.

[48] John McCarthy. Programs with common sense. In *Proceedings of the Symposium on the Mechanization of Thought Processes*, volume 1, pages 77–84. National Physical Laboratory, 1958. Reprinted in Minsky's (ed.) *Semantic Information Processing*, MIT Press(1968), 403–409.

[49] Thomas M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings IJCAI-77*, pages 305–310. International Joint Committee for Artificial Intelligence, August 1977.

[50] D.S. Mitrinović. *Analytic Inequalities.* Springer-Verlag, New York, 1970.

[51] B. K. Natarajan. On learning Boolean functions. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 296–304, May 1987.

[52] Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *Journal of the Association for Computing Machinery*, 35(4):965–984, 1988.

[53] Leonard Pitt and Manfred Warmuth. Reductions among prediction problems: On the difficulty of predicting automata (extended abstract). In *3rd IEEE Conference on Structure in Complexity Theory*, pages 60–69, June 1988.

[54] Leonard Pitt and Manfred K. Warmuth. Prediction preserving reducibility. *Journal of Computer and Systems Sciences*, 1990. Special issue for 3rd IEEE Conference on Structure in Complexity Theory, to appear. A preliminary version is available as U.C. Santa Cruz Computer Research Laboratory Technical Report UCSC-CRL-88-26, 1988.

[55] Ronald Rivest. Personal communication.

[56] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[57] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 411–420, May 1989.

[58] Robert E. Schapire. Pattern languages are not learnable. In *Proceedings of the Third Annual Workshop on Computational Learning Theory.* Morgan Kaufmann, August 1990.

[59] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 1990. Special issue for COLT 89, to appear. A preliminary version is available in *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 28–33,1989.

[60] Alistair Sinclair. *Randomised Algorithms for Counting and Generating Combinatorial Structures.* PhD thesis, University of Edinburgh, Department of Computer Science, November 1988.

[61] Robert H. Sloan. Some notes on Chernoff bounds. (Unpublished), 1987.

[62] Robert Hal Sloan. *Computational Learning Theory: New Models and Algorithms.* PhD thesis, MIT Dept. of Electrical Engineering and Computer Science, May 1989.

[63] Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 118–126, May 1983.

[64] Leslie Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:198–201, 1979.

[65] Leslie Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5:363–366, 1984.

[66] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[67] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, XVI(2):264–280, 1971.

[68] Manfred Warmuth. Personal communication.

[69] Peter Winkler. Personal communication.

Index