# Computational Learning Theory: New Models and Algorithms

by

Robert Hal Sloan

S.M. EECS, Massachusetts Institute of Technology (1986)
B.S. Mathematics, Yale University (1983)

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

June 1989



Signature of Author_____
Department of Electrical Engineering and Computer Science
May 23, 1989

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

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

## Abstract

In the past several years, there has been a surge of interest in computational learning theory—the formal (as opposed to empirical) study of learning algorithms. One major cause for this interest was the model of probably approximately correct learning, or pac learning, introduced by Valiant in 1984.

This thesis begins by presenting a new learning algorithm for a particular problem within that model: learning submodules of the free $\mathbf{Z}$-module $\mathbf{Z}^k$. We prove that this algorithm achieves probable approximate correctness, and indeed, that it is within a log log factor of optimal in a related, but more stringent model of learning, on-line mistake bounded learning.

We then proceed to examine the influence of noisy data on pac learning algorithms in general. Previously it has been shown that it is possible to tolerate large amounts of random classification noise, but only a very small amount of a very malicious sort of noise. We show that similar results can be obtained for models of noise in between the previously studied models: a large amount of malicious classification noise can be tolerated, but only a small amount of random attribute noise.

Next, we overcome a major limitation of the pac learning model by introducing a variant model with a more powerful teacher. We show how to learn any concept representable as a boolean function, with the help of a teacher who breaks the concept into subconcepts and teaches one subconcept per lesson. The learner outputs not the unknown boolean circuit, but rather a program which, on a given input, either produces the same answer as the unknown boolean circuit would, or else says "I don't know." Thus, unlike many learning programs, the output of this learning procedure is *reliable*. Furthermore, with high probability the output program is nearly always *useful* in that it says "I don't know" on only a small fraction of the domain.

Finally, we look at a new model for an older learning problem, inductive inference. This new model combines certain features of the traditional model of Gold for inductive inference together with the concern of the Valiant model for efficient computation and also with notions of Bayesianism. The result is a model that captures certain qualitative aspects of the classic scientific method.

**Keywords:** Machine learning, computational learning theory, concept learning, noise, inductive inference, scientific method.

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

# Acknowledgments

My first and greatest thanks go to my advisor, Ron Rivest. Ron was a pleasure to work with throughout my time at MIT, and much of this thesis was joint work with him. Whenever I was stuck on a problem, Ron had new approaches to suggest. His approach would always be some clever idea that I wished I had thought of; moreover, his clever idea very often solved the problem. In particular, when I was stymied by the problem of learning by subconcepts discussed in Chapter 5, it was Ron who said, "Well, why not just keep *all* the possibilities for the right function?"

I was also fortunate to get to work with Manfred Warmuth and David Helmbold. The results of Chapter 3 were joint work with them. I would like to thank the two of them both for their gracious permission to reproduce that work here, and for the fun I had working with them. More generally, I would like to thank both Manfred and David Haussler for inviting me to visit U. C. Santa Cruz in the summer of 1988, where I had many fruitful technical conversations with Manfred and both Davids.

Several people helped me proofread and edit this document. Ron Rivest was extremely helpful in this activity. Additionally, careful proofreading of parts of this thesis by Rob Gross, Maury Neiberg, and Manfred Warmuth resulted in numerous small improvements in the text.

Finally, I want to acknowledge the funding agencies without whom this sort of research would not occur. I was supported while writing this thesis by an NSF graduate fellowship, and received additional support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and the Siemens Corporation.

# Contents

# Chapter 1

# Introduction

*Learning is not attained by chance, it must be sought for with ardor and attended to with diligence.*
—ABIGAIL ADAMS, *Letter to John Quincy Adams*

## 1.1 Prologue: An afternoon stroll

It is a bright, sunny afternoon. I take my robot out for a walk with me. We stop and sit down on a bench in a busy part of town. For every person who walks by us I say to my robot either, "That's a male," or "That's not a male," as the case may be. We stay quite some time, and eventually go home.

The following afternoon the weather is the same, and we go back to the same bench. This time, for every person who walks by, my robot says to me either "That's a male," or "That's not a male." My robot is correct ninety-six percent of the time.

It seems fair to say that my robot has learned the concept *male*, or at least a close approximation of that concept. One of the main subjects of this thesis will be what algorithms a robot might use to accomplish such learning.

## 1.2 An introduction to learning theory

Formally, the subject of this thesis is computational learning theory. For our purposes, learning means induction. The task of a learner is to sample some portion of the world, or whatever more limited domain may be under consideration, and come to some conclusion about the nature of the entire domain. What distinguishes *computational* learning theory is that one of the efficiency issues that we care about is how much computation time the learner uses. Our goal is twofold: we want to specify interesting formal models of the problem of learning, and we want to present algorithms for achieving learning within these models.

All of the problems we study in this thesis can be fit into the following broad pattern. There is some universe of objects that is under consideration. We call this set the *instance space* or the *domain.* The domain might be the set of points in the real plane, bit vectors, the set of all the fish in the Boston Aquarium, or the set of all human beings on earth. The elements of the domain are called *instances.* These instances are split into two categories: positive instances and negative instances. The set of positive instances is called the *target concept.* Our *learner* receives as input a sample of instances with labels that tell whether they are positive instances or negative instances. We call such labeled instances *examples.* The goal of the learner is to to find some rule which distinguishes positive instances from negative instances.

For instance, in the example given above in the Prologue, the domain is the set of all people, and the positive instances are male people. In that case I did not demand that the learner come up with a completely accurate rule for classifying people; I was happy with a good approximation.

There are several issues we must consider before we have posed a well specified learning problem. In particular, Rivest [46] suggests that (at least) the following seven questions must be answered in order to specify a learning problem.

1. What is being learned?

2. From what is it learned?

3. What a priori knowledge does the learner begin with?

4. How is what is learned represented?

5. By what method is it learned?

6. How well is it learned?

7. How efficiently is it learned?

We now briefly examine each of these questions in turn.

### 1.2.1 What is learned

There are many different domains one may study learning for. One might be interested in learning about people or maps or nuclear particles. This thesis is a theoretical work, and we examine exclusively abstract mathematical domains such as bit vectors, Euclidean $n$-space, and $k$-tuples of integers. We study these domains both because they are interesting in their own right, and in the belief that they are rich enough to form good mathematical models of real world problems.

For instance, the robot discussed in the Prologue presumably would represent people as a feature vector which includes (at least) real, integer, and boolean valued components. The problem of choosing a good representation is both fascinating and difficult—indeed, it is one of the central problems of Artificial Intelligence—but it falls outside the scope of this thesis.

### 1.2.2 From what is it learned

The data our learner gets consists of labeled instances. As noted above, these instances come from various abstract mathematical domains. We generally ignore the details of how these instances are represented, and just assume some reasonable fixed encoding scheme.

We are, however, very concerned with the issue of how training examples are chosen from the domain. If the examples are chosen to be extremely helpful, then philosophically one might object that what is going on is programming rather than learning. A more practical objection is that we now have the problem of finding a teacher capable of picking out such extremely helpful examples. On the other hand, if the examples are not somewhat representative of the whole domain, then the learning task may be infeasible.

Another issue that sometimes arises is whether the learner is given both positive and negative training examples or only positive examples. Which is more natural depends on the domain. In the case of learning to distinguish male people, both positive and negative training instances would presumably be available. On the other hand, in the case of learning to distinguish well formed English sentences from nonsense sounds, only positive instances might be available.

We are also interested in what happens when there is some noise in the training data. For obvious reasons we prefer learning algorithms that are robust against some amount of noise.

### 1.2.3 What a priori knowledge does the learner have

Our learner does not necessarily begin its[1] work in a state of total ignorance. For instance, we often help the learner by telling it ahead of time that the target concept

[1]In this thesis learners, robots, and learning algorithms will be referred to by the pronoun "it."

comes from some particular class of concepts. (More formally, we design learning algorithms that are only guaranteed to work on the assumption that the target concept comes from some particular class.) Some limiting of the class of possible concepts is clearly necessary. If any subset of the instance space could be the target concept, and if all such subsets were equally likely, then the learner would have no basis whatsoever for performing its induction.

Another sort of initial knowledge a learner may have consists of a priori probabilities of the truth of various propositions such as "Concept $c$ is the target concept."

### 1.2.4 How is what is learned represented

In some cases we are happy if the learner outputs any representation of the target concept. Sometimes, however, we further complicate the learner's task by requiring that its output be in some particular form. In particular, when it is known a priori that the target concept is representable in some special form (for instance, a boolean formula in 3DNF), we may require that the learner's output also be in that form.

### 1.2.5 By what method is it learned

The answers to the previous questions determine a learning problem. Our goal is then to find a solution to that problem: an algorithm that learns according to the definition of learning given by our answers to the previous questions.

The goal of computational learning theory in general, and this thesis in particular, is to pose interesting learning problems, and to find algorithms that solve those problems.

### 1.2.6 How well is it learned

A learning algorithm is a proposed solution to a problem in learning. Having proposed a solution, one must next ask how good that solution is.

Two very different notions of learning well have been studied. In one, generally called *inductive inference*, the learner is given a presentation of examples one at a time indefinitely, and its goal is to converge in the limit to the correct rule for classifying instances. Inductive inference was first studied by Gold [14], and it has received considerable attention since then. For the most part we are not concerned with inductive inference in this thesis (although it was one of the inspirations for the model we study in Chapter 6). For interested readers, [6] is an excellent survey article; an excellent introductory book on the subject is [39].

In this thesis, we are instead concerned with what is generally known as *learning concepts from examples* or simply *learning from examples.* The learner receives only some limited number of examples, and after that is supposed to output some representation of a rule for distinguishing positive instances from negative instances. Intuitively, the idea is that the presentation of the labeled instances constitutes the training of the learner, and that after training the learner should be able to classify previously unseen instances for itself.

In the context of learning from examples, we normally call the rule for distinguishing positive from negative instances a *concept.* As mentioned above, the true rule is called the *target concept.* To decide whether the learner has learned well, we may ask simply whether the learner has produced as output some representation of the target concept. Learning, however, is a difficult task, and unless the task is highly constrained, always finding exactly the target concept concept may be difficult or even impossible. (Remember that the whole point of learning from examples is to avoid showing the learner all possible instances in the training phase.) Thus we often settle for the learner outputting a concept that is "close to" the target concept, or even "probably" close to the target concept. Of course, "close to" and "probably" must be precisely defined in any particular formal model of concept learning.

Having decided upon some criteria for learning well, we must then decide how to

measure whether those criteria have been met. One obvious approach is to implement any proposed learning algorithm and then run it on test data. Much artificial intelligence research on machine learning follows that empirical approach.

This thesis, however, contains no empirical results at all. Instead, we give formal *proofs* that our algorithms meet various criteria for learning well. The emphasis on proving the goodness of algorithms is, in fact, one key property that distinguishes computational learning theory from other machine learning research.

### 1.2.7 How efficiently is it learned

In addition to requiring the learner to learn well, we also require the learner to learn efficiently. There are three sorts of resources the learner consumes:

1. Computation time.

2. Memory.

3. Labeled instances.

We require that our learner be restricted to efficient computation. At the present time, "efficient computation" is generally understood to mean whatever can be computed in probabilistic polynomial time (technically *BPP*). That is the restriction we impose on all learning algorithms studied in this thesis. Of course, in order to restrict our learner to probabilistic polynomial time computations, we have to answer the question, "Polynomial in what?"

For the most part, we are content simply with finding algorithms that meet the broad *BPP* definition of efficient, and ignore the question of the particular asymptotic running times of our algorithms, though we occasionally discuss running times. Notice that this approach is may be unrealistic if one's goal is to actually build real-time robots.

Similarly, we generally ignore the memory usage of our algorithms. The running time of an algorithm places a crude bound on its memory usage, and we settle for that.

In addition to the usual resources of time and space, learning algorithms also consume labeled instances. The number of instances consumed forms an obvious lower bound on the running time of an algorithm, so we restrict all our algorithms to a polynomial number of instances.

In the other direction, we study how many examples must be present to solve certain learning problems. Just as the problem of sorting has an $\Omega(n \log n)$ lower bound on its time complexity—independent of how much memory a sorting algorithm has available—certain learning problems have various lower bounds on their *sample complexity* independent of how much time or memory they have available. Intuitively a particular sample complexity for a learning problem means that at least that much data are required for a statistically adequate sample.

## 1.3 Overview of remaining chapters

In the next chapter we formally specify a particular model of concept learning, Valiant's pac learning model [52]. That model addresses many of the concerns discussed in this introductory chapter.

In Chapter 3 we exhibit a learning algorithm for learning a certain class of concepts within the pac learning model. The instance space is $\mathbf{Z}^k$, or more generally, any Euclidean domain, and the concepts of interest are any submodule. We give a detailed analysis of the performance of the algorithm, and the lower bounds for the problem it solves, and examine several applications of the algorithm. We show that this algorithm is optimal within the pac learning model, and near optimal in a related but more stringent learning model called on-line learning.

The basic pac learning model of Valiant [52] ignores the issue of noise. Since the

model was published, several authors have considered the effects of noisy data within that model [51, 5, 25, 31]. Those papers all assumed one of two particular types of noise that might corrupt the data: one very malicious, and one very benign. Not surprisingly, it was shown that the maximum tolerable amount of noise is much greater for the benign model than for the malicious model. In chapter 4, we consider where the dividing line between these models falls. In particular, we study two new models of noise, both "in between" the models previously studied.

One limitation of the pac learning model is that certain learning problems are computationally intractable in that model. In Chapter 5, we specify a learning model with a more helpful teacher than the one allowed in pac learning. With the aid of this teacher, we be able to learn any reasonable concept class, and also achieve a stronger sort of learning than pac learning.

Finally, in Chapter 6, we look at an altogether different type of learning. In that chapter we introduce a new model of learning that combines certain aspects of learning from examples and inductive inference. The result is something that crudely models the scientific method.

# Chapter 2

# Pac learning

## 2.1 Introduction

In this chapter we formally define one particularly important model of learning, "probably approximately correct learning," or pac learning for short [52]. All of the results in this thesis except for those in Chapter 6 are in either the pac learning model or some variation thereof. We briefly discuss the intuition behind the pac model, and then give an example of an algorithm for a particular learning problem within this model, learning monomials.

Next we discuss a number of technical issues of pac learning. These issues are important in Chapter 4 where we discuss the effect of noisy data on pac learning algorithms, and in Chapter 5 where we discuss a variation on the pac learning model.

We conclude by discussing sufficient conditions for pac learning, including the important combinatorial parameter known as the Vapnik-Chervonenkis dimension [55]. An understanding of that parameter is necessary for some of the results in Chapter 3.

With the exception of Theorem 2.2, which is original, the material in this chapter is all a review of the computational learning theory literature.

## 2.2 Pac learning

In an influential 1984 article [52], Valiant introduced the pac learning model. In short, an algorithm pac learns from examples if it can, in a feasible amount of time, find (with high probability), a rule that is highly accurate. Now we must define what we mean by such terms as "find a rule," "with high probability," and "highly accurate."

Fix an instance space $X$. Formally a *concept* $c$ for instance space $X$ is some subset of $X$. If instance $x \in X$ is contained in concept $c$, then we say that $x$ is a *positive instance* of $c$; otherwise we say that $x$ is a *negative instance* of concept $c$. (We are slightly sloppy throughout and refer to concepts interchangeably both as subsets of the instance space and as $\{0,1\}$-valued functions defined on the instance space.)

The *length* of concept $c$, denoted $|c|$, is the number of bits it takes to write down $c$ in some agreed-upon encoding scheme. For example, if our instance space is $\{0,1\}^n$, possible representations for concepts include truth tables, boolean formulas, and boolean circuits. (Haussler [20] gives a good discussion of issues concerning choice of representation of concepts.) The length of an instance is defined similarly.

Let $\mathcal{C}$ be a set of concepts or *concept class* over $X$. Formally, $\mathcal{C} \subseteq 2^X$. If $X$ is the inhabitants of the U. S., concepts would include both the rather simple concept *males*, and the doubtless more complicated concept, *people whose marginal federal income tax rate is thirty-three percent.* An example of a concept class would be TAX BRACKETS which would include the concepts *people paying no income tax* and *people whose marginal tax rate is thirty-three percent.*

We call the concept $c \in \mathcal{C}$ that our algorithm is trying to learn the *target concept.* The target concept may be any concept whatsoever in the concept class. We think of it as being arbitrarily chosen by some outside teacher or supervisor.

We assume that our learning algorithm has available to it a black box called EXAMPLES, and that each call to the black box returns a labeled example, $(x, s)$, where

$x \in X$ is an instance, and $s$ is either "+" or "−" according to whether $x$ is a positive or negative instance of the target concept $c$. Furthermore, the EXAMPLES box generates the instances $x$ according to some fixed probability distribution $\mathbf{P}$ on $X$. We make no assumptions whatsoever about the nature of $\mathbf{P}$, and our learner is not told what $\mathbf{P}$ is.

First we define formally what it means for a concept to be an accurate approximation of target concept $c$, and then we define pac learning itself.

DEFINITION. Fix an instance space $X$ and a probability distribution $\mathbf{P}$ on $X$. We say that concept $c_1$ is an $\epsilon$-*approximation* of concept $c_0$ if and only if

$$\sum_{c_1(x) \neq c_0(x)} \mathbf{P}(x) \leq \epsilon. \tag{2.1}$$

DEFINITION. Let $\mathcal{C}$ be a class of concepts on domain $X$. Algorithm $A$ *probably approximately correctly learns (pac learns)* $\mathcal{C}$ if and only if for every $c \in \mathcal{C}$, for every probability distribution on $X$, for every positive $\epsilon$ and $\delta$, algorithm $A$, given only $\epsilon, \delta$ and access to EXAMPLES($c$), meets the following two criteria.

**Learning criterion** Algorithm $A$ outputs some representation of a concept $c'$ such that

$$\Pr[c' \text{ is an } \epsilon\text{-approximation of } c] \geq 1 - \delta \tag{2.2}$$

where the probability is taken over the output of EXAMPLES and any coin tosses $A$ may make.

**Efficiency criterion** The running time of $A$ is bounded by some polynomial function of $1/\epsilon$, $1/\delta$, the length of an instance, and the length of the target concept.

We say that a concept class $\mathcal{C}$ is *pac learnable* if there exists some algorithm that pac learns $\mathcal{C}$.

We note here for later use the following weaker definition.

DEFINITION. *Statistical pac learning or s-pac learning* is defined the same as pac learning except that the efficiency criterion is that the sample complexity rather than the running time must be bounded by by some polynomial function of $1/\epsilon$, $1/\delta$, the length of an instance, and the length of the target concept.

## 2.3 Discussion of the definition

Intuitively, we are saying that the learner is supposed to do the following:

1. Ask nature for a random set of examples of the target concept.
2. Run in polynomial time.
3. Output some concept that with high probability agrees with the target concept on most of the instances.

We think of Nature as providing examples to the learner according to the (unknown) probability with which the examples occur in Nature. Though the learner does not know this probability distribution, he does know that the concept he outputs needs to closely approximate the target concept only for this probability distribution.

Intuitively, there may be some extremely bizarre but low probability examples that occur in Nature, and it would be unreasonable to demand that the learner's output concept classify them correctly. Hence we require only approximate correctness. Moreover, since the examples the learner receives from Nature are drawn stochastically, there is some small but nonzero chance that those examples will be wildly unrepresentative. Therefore we cannot require the learner to *always* output an approximately correct concept; we only require that the learner do so with high probability.

The notion of s-pac learning was studied in the statistical pattern recognition literature before the notion of pac learning was introduced in 1984. The difference between

the two is that s-pac learning requires us to be efficient in our consumption of examples but not in our computation time. Notice that any algorithm that pac learns must s-pac learn since the sample complexity of an algorithm is a lower bound on its running time. Thus the strongest infeasibility results we can obtain are showing the infeasibility of s-pac learning. In general it is often easier to prove theorems about s-pac learning than about pac learning.

The excellent survey article of Kearns et al. gives a more lengthy discussion of the entire pac learning model, and also provides an overview of recent results obtained using this model [26]. Blumer et al. and Valiant both contain additional material on the motivation for the pac learning model [9, 52].

## 2.4 An example of pac learning

We now exhibit an algorithm that pac learns monomials [52]. The instance space is $\{0,1\}^n$ for some positive $n$. The concept class is the set of all boolean expressions in $n$ variables that can be represented as monomials (simple conjunctions of literals).

The algorithm $A$ for pac learning is as follows: $A$ first makes one call to EXAMPLES, and figures out what $n$ is. (Alternatively, $n$ may be an additional input to $A$.) $A$ then initializes its hypothesis to be the conjunction of all $2n$ literals, $x_1\bar{x}_1 \cdots x_n\bar{x}_n$. $A$ then makes $m$ more calls to EXAMPLES, where

$$m = \frac{1}{\epsilon}\left(n \ln 3 + \ln(1/\delta)\right).$$

For each positive instance, $(instance, +)$, algorithm $A$ crosses off $x_i$ from its hypothesis if $x_i$ is 1 in *instance*, and crosses off $\bar{x}_i$ if $x_i$ is 0 in *instance*. Algorithm $A$ simply ignores negative instances. The output of $A$ is the monomial consisting of whatever literals have not been crossed off after the $m$ instances are seen. (If all of the $m$ instances are negative, then $A$'s hypothesis is $x_1\bar{x}_1 \cdots x_n\bar{x}_n$, which is always false.)

The running time of of $A$ is clearly polynomial in $1/\epsilon, 1/\delta$, and $n$, and the length of an instance is $n$, so $A$ meets the efficiency criterion for pac learning. It follows easily from the results of Blumer et al. [11] that $A$ meets the learning criterion for pac learning. The value of $m$ is not the best possible, but it makes the proof especially simple.

## 2.5 Some definitions and technical details

### 2.5.1 Asymptotics

Throughout, we strive to find computationally efficient learning algorithms. We therefore need to discuss the asymptotic difficulty of our problems. Otherwise, all problems can be solved in "constant" time.

For instance, the concept class $\mathcal{C}_n^{\text{poly}}$ of all subsets of $\{0,1\}^n$ accepted by some polynomial-size circuit is not pac learnable if any one-way function exists (Valiant [52] using cryptographic tools of Goldreich, Goldwasser, and Micali [15]). Nevertheless, if we simply fix $n$, in spite of that result, it *would* be possible to pac learn the class $\mathcal{C}_n^{\text{poly}}$. The running time of the algorithm to do so would be a particular polynomial function of $1/\epsilon$ and $1/\delta$. That polynomial would, however, contain some multiple of the "constant" $2^n$ as one of its coefficients.

Thus instead of examining algorithms for one fixed learning problem, we always examine algorithms for an infinite sequence of learning problems. There are two different ways we can construct this sequence: by allowing instance length to grow and by allowing concept length to grow.

In the first approach we must parameterize the instance space. Thus our instance space $X = \cup_{n=1}^{\infty} X_n$. We assume that there is some polynomial $l$ such that all instances in $X_n$ have length between $n$ and $l(n)$. We also assume that any one concept $c$ in

our concept class $\mathcal{C}$ is completely contained in $X_n$ for some $n$. We define $\mathcal{C}_{n,s}$ to be $\{c \in \mathcal{C} | c \subseteq X_n \text{ and } |c| \leq s\}$, and $\mathcal{C}_n = \cup_s \mathcal{C}_{n,s}$. Various classes of boolean functions are often studied in this way. Another example of a concept class that can be studied this way is the class of half-spaces of $\Re^n$. We note for future reference that we consider such a concept class to be finite if $|\mathcal{C}_n|$ is finite for every $n$.

The other way to make our learning problems asymptotic is to take a fixed instance space, but to allow concepts to grow arbitrarily complex. For instance, we could fix the instance space to be the real plane, and let $\mathcal{C}$ be the set of all convex polygons. We could then make $\mathcal{C}_n$ the set of convex $n$-gons, or alternatively, the set of convex polygons with at most $n$ edges.

### 2.5.2 Method of sampling

Notice that our definition of pac learning did not require that the learning algorithm make all its calls to EXAMPLES at once. Indeed, the pac learning algorithm we gave for learning monomials first makes one call to determine the length of instances, and then makes $m$ further calls, where $m$ is dependent on what that first call to EXAMPLES returned.

Early work in pac learning generally required that a learning algorithm make a fixed number of calls to EXAMPLES dependent only on the size of instances, $\epsilon$, and $\delta$. We refer to that model of learning as *static sampling pac learning.* Note that in the static model, the size of an instance is normally an additional input parameter. Linial, Mansour, and Rivest [33] discuss the ramifications of allowing arbitrary calls to EXAMPLES instead of requiring static sampling.

### 2.5.3 Samples and consistency

For a fixed target concept $c_t$, we call a set of instances each labeled positive or negative a *sample* (of $c_t$). Whenever there is an underlying probability distribution on the instance space, a sample is assumed to be randomly drawn according to that probability distribution. The size of a sample is the number of instances it contains. Notice that in the usual case where the sample is obtained stochastically, the number of distinct instances in the sample may be less than the size of the sample.

We say that a concept $c$ is *consistent* with a given sample if $c$ contains all the positive instances in a sample and none of the negative instances. The *consistency problem* for a concept class is to find a consistent concept for a given sample of some concept in the class.

### 2.5.4 Variations on pac learning

The pac learning model itself can be altered in many ways. For instance, instead of having a single probability distribution on the whole instance space and a single box EXAMPLES, there can be two separate probability distributions, one each for positive examples and negative examples, and two boxes, $\text{EXAMPLES}^+$ and $\text{EXAMPLES}^-$, each giving only one kind of instance. In this case, the learning algorithm is allowed to make calls to either box, and its hypothesis is required to come within $\epsilon$ on both positive and negative examples. This model is called the two-button model; the model we discussed first is called the one-button model.

There are in fact numerous different minor variations on pac learnability. Haussler et al. compare many of these variations, and show them to be substantially equivalent.

## 2.6 Sufficient conditions for pac learnability

Let us briefly examine when a concept is pac learnable. A starting place place is the case when the concept class has very small cardinality. In the extreme, consider the case where the concept class contains only two concepts, $c_1$ and $c_2$.

In general, we are concerned only with situations where it is computationally efficient to determine whether any particular instance is contained in a given concept. Let us assume that is the case for $c_1$ and $c_2$. Now all we need to do is draw a sufficiently large sample from EXAMPLES, and select whichever concept is consistent with the sample, or, if both concepts are consistent, choose either one arbitrarily.

We can extend this line of reasoning to considerably larger (finite) concept classes. Our treatment follows Blumer et al. [11]. Consider an algorithm that draws a sample of size $m(|\mathcal{C}|, \epsilon, \delta)$, and then outputs any concept $c \in \mathcal{C}$ that is consistent with the sample. Ignoring for a moment the issue of computation time, what is a lower bound on the function $m$ that insures that the learning algorithm meets the s-pac learning condition?

DEFINITION. Fix a target concept and a probability distribution on the instance space. We say that a concept is *$\epsilon$-good* if it is an $\epsilon$-approximation of the target concept, and *$\epsilon$-bad* if it is not.

We want to guarantee that the probability that the algorithm returns an $\epsilon$-bad concept is at most $\delta$. The probability that a particular $\epsilon$-bad concept would be consistent with $m$ randomly drawn examples is bounded above by

$$(1 - \epsilon)^m. \tag{2.3}$$

A crude upper bound on the probability that *any* $\epsilon$-bad concept is consistent with $m$ randomly drawn examples is

$$|\mathcal{C}|\,(1 - \epsilon)^m. \tag{2.4}$$

Hence we want to choose $m$ large enough to insure that

$$|\mathcal{C}|\,(1-\epsilon)^m \leq \delta. \tag{2.5}$$

Taking logs of both sides, and solving for $m$ yields

$$m \geq \frac{1}{-\ln(1-\epsilon)}\left(\ln(|\mathcal{C}|) + \ln(1/\delta)\right). \tag{2.6}$$

We can write a simpler inequality that implies inequality (2.6) by noticing that by Taylor's Theorem

$$\begin{aligned}\ln\left(\frac{1}{1-\epsilon}\right) &> \ln 1 + \epsilon\frac{1}{(1-\epsilon)} + \frac{\epsilon^2}{2!}\frac{1}{(1-\epsilon)^2} \\ &> \frac{\epsilon}{1-\epsilon} \\ &> \epsilon.\end{aligned}$$

Hence a sufficient condition is

$$m \geq \frac{1}{\epsilon}\left(\ln(|\mathcal{C}|) + \ln\left(\frac{1}{\delta}\right)\right). \tag{2.7}$$

We say that $m$ is an upper bound on the *sample complexity* of the concept class $\mathcal{C}$. More generally we say that an arbitrary concept class $\mathcal{D}$ has sample complexity at most $m$ if there is a learning algorithm with sample complexity $m$ that s-pac learns $\mathcal{D}$.

Let us assume that we have an instance space, such as $\{0,1\}^n$, where all instances have length $n$, and the length of every concept in our concept class $\mathcal{C}$ is bounded by some polynomial function of $n$. As long as $|\mathcal{C}| = O\left(2^{p(n)}\right)$, for some polynomial $p$, then $\mathcal{C}$ has polynomial sample complexity. If $\mathcal{C}$ has polynomial sample complexity, and if the problem of finding a concept from $\mathcal{C}$ consistent with a given sample can be solved in polynomial time, then $\mathcal{C}$ is pac learnable. We call any pac learning algorithm that draws a sample of size $m(\epsilon, \delta, |\mathcal{C}|)$ for some function $m$, and then returns an arbitrary $c \in \mathcal{C}$ that is consistent with the sample a *static consistent learning algorithm.*

There are at most $3^n$ different monomials on $n$ boolean variables, so the concept class of monomials has polynomial sample complexity. The algorithm for learning monomials given above in Section 2.4 is a static consistent algorithm. It draws a sample of a size that meets the bound of inequality (2.7), and in polynomial time finds a monomial consistent with that sample.

On the other hand, the class of all boolean functions on $n$ variables has size $2^{2^n}$, so we can not hope to find a static consistent algorithm for it of this sort.

### 2.6.1 Occam algorithms

A static consistent algorithm draws a sample, and outputs some hypothesis consistent with that sample. There is a more general class of algorithms with that behavior which achieve pac learning, the *Occam algorithms* [11].

DEFINITION. An Occam algorithm for $\mathcal{C}$ with constant parameters $c \geq 1$ and $0 \leq \alpha < 1$ is an algorithm that:

1. produces a concept (not necessarily from $\mathcal{C}$) of length at most $n^c m^\alpha$ when given a labeled sample of length $m$ of any target concept in $\mathcal{C}$ of length at most $n$, and
2. runs in time polynomial in the length of the concept.

The importance of Occam algorithms is shown in the following theorem of [11]:

**Theorem 2.1 (BEHW)** *Any Occam algorithm for $\mathcal{C}$ pac learns $\mathcal{C}$.*

Notice that static consistent algorithms are the special case of Occam algorithms with $\alpha = 0$.

Observe that an Occam algorithm $A$ must be a data compression algorithm. If we get a sample of labeled instances we can compress it by storing just the instances and the concept output by the Occam algorithm. To decompress we use the concept to recover the original labels.

In particular, the definition of Occam algorithm implies that there is a constant $0 < \beta < 1$ such that for every $n$, for all sufficiently large $m$, on input a sample of length $m$ of a concept of length at most $n$ algorithm $A$ must output a consistent concept of length at most $m^\beta$. We know that the length of $A$'s output can be at most $n^c m^\alpha$ for some $\alpha < 1$. Now for any fixed $n$, for sufficiently large $m$, $m^{\frac{1-\alpha}{2}} > n^c$. Thus setting $\beta = (1+\alpha)/2$ suffices.

In general there is a strong connection between the ability to compress and the ability to learn. Indeed, the ability to compress the sample somewhat is a necessary condition for at least static sampling pac learning.

**Theorem 2.2** *Let $C = \cup_{n=1}^{\infty} C_n$ be a concept class where $C_n$ is defined on $\{0,1\}^n$. If $C$ is statically pac learnable, then there must be an algorithm $A$ such that for every sufficiently large $n$, there exists a constant $\epsilon > 0$ such that for sufficiently large $m$, given a sample of length $m$ of any concept $c \in C_n$, algorithm $A$ with probability at least $1 - |n|^{-\omega(1)}$* returns some representation of a hypothesis of length at most $(1-\epsilon)m$ that is consistent with the sample.*

*Proof* We employ cryptographic tools from Yao's theory of "computational information theory" [57] to show that if such an algorithm does not exist, then we cannot hope to pac learn (or even weakly approximate) $C$.

For each $n$, let the probability distribution $\mathbf{P}$ on $\{0,1\}^n$ be the uniform distribution, and fix some concept $c_n \in C_n$. For $n \geq 1$, let $S_{n+1}$ be a device that stochastically generates a string consisting of an instance $x$ chosen randomly from $\{0,1\}^n$ followed by $c_n(x)$. Let $X_1$ be an arbitrary stochastic source of strings of length 1. Then $S = S_1, S_2, \ldots$ forms a uniform source ensemble. That is to say, each $S_n$ is a stochastic

*Recall that $f(n) = \omega(g(n))$ if and only if $g(n) = o(n)$. In particular, $f(n) = |n|^{-\omega(1)}$ if and only if $f(n)$ vanishes faster than the reciprocal of any polynomial in $n$, or, formally, if for every positive $c$, for all sufficiently large $n$, we have $f(n) < 1/n^c$.

source of strings of length $n$. (For the precise definition see either Yao [57] or a text on information theory.)

Let the true random number ensemble (on alphabet $\{0,1\}$) be the source ensemble $\mathcal{R} = R_1, R_2, \ldots$ where $R_n$ assigns probability $2^{-n}$ to strings of length $n$ and probability 0 to all other strings.

Claim: If $\mathcal{S}$ is polynomial indistinguishable from $\mathcal{R}$, then concept class $\mathcal{C}$ is not statically pac learnable.

We show the contrapositive of this claim. Assume that we have an algorithm $SPL$ that statically pac learns $\mathcal{C}$. Then we can use algorithm $SPL$ as a subroutine in an algorithm to distinguish $\mathcal{S}$ from $\mathcal{R}$. Let $s(n, \epsilon, \delta)$ be the number of instances from EXAMPLES that $SPL$ requires.

The distinguishing algorithm will work to distinguish samples of $S_n$ from samples of $R_n$ as long as the sample length is at least $1 + s(n-1, \frac{1}{4}, \frac{1}{4})$. The behavior of the distinguishing algorithm is as follows: We feed all the labeled examples in the sample except the last one to $SPL$, to obtain an output concept $\hat{c}$. The distinguishing algorithm outputs 1 if the classification of the last instance in the sample according to $\hat{c}$ matches its true classification, and 0 otherwise. If the sample came from $S_n$, then the output of the distinguishing algorithm must be 1 with probability at least $(1 - \frac{1}{4})(1 - 14) = \frac{9}{16}$. On the other hand, if the the sample came from the true random number ensemble, then the output of the distinguishing algorithm is 1 with probability exactly $\frac{1}{2}$.

Hence it must be that $\mathcal{S}$ is polynomial distinguishable from the true random number ensemble. Yao [57] shows that this implies that it must be possible to communicate a sufficiently long sample from $S_n$ from one polynomial time computing agent to another using polynomially fewer bits than are in the sample with overwhelming probability. (The probability is taken over the output of the source and over any coins the two communicating parties may toss.) In particular, for sufficiently large $m$, it must be possible with probability at least $1 - |n|^{-\omega(1)}$, to communicate a sample of of length

$m$ (containing $mn$ bits) in at most $m(n - 1/n^k)$ bits for some positive integer $k$. Now of the $mn$ bits, $m(n-1)$ bits come from drawing randomly from $\{0,1\}^{n-1}$, and so cannot be compressed at all. That means that it must be possible to write down some representation of the $m$ label bits in at most

$$[m(n - 1/n^k)] - m(n-1) = m(1 - 1/n^k)$$

bits. Putting $\epsilon = 1/n^k$ yields the desired result. □

### 2.6.2 VC dimension

Previously we have examined sufficient conditions for pac learnability. In this section we examine a combinatorial parameter that provides a necessary and sufficient condition for static sampling pac learnability, the Vapnik-Chervonenkis dimension [55], hereinafter VC dimension or VCdim. Since it is a combinatorial parameter of set systems, we define it in terms of sets, though of course the sets we are interested are concepts.

DEFINITION. Fix a domain $X$, and let $\mathcal{C} \subseteq 2^X$. A set $S \subset X$ is *shattered* (by $\mathcal{C}$) if for each subset $S' \subseteq S$, there is a set $c \in C$ which contains all of $S'$, but none of the points in $S - S'$.

*Remark:* The term shatter is now well established. However, as Pollard points out [43], the right picture to keep in mind is not really $S$ being broken into lot of tiny pieces by $\mathcal{C}$. Rather, one should imagine a diligent $\mathcal{C}$ picking out all the different subsets of $S$.

DEFINITION. The *VC dimension* of $\mathcal{C} \subseteq 2^X$ is the cardinality of the largest set of points from $X$ shattered by $\mathcal{C}$.

*Examples:* The class of all rectangular regions in the plane has VC dimension 4. The class of all spheres in $\Re^n$ has dimension $n+1$. For any finite concept class $\mathcal{C}$, we

have $VCdim(\mathcal{C}) \leq \log(|\mathcal{C}|)$.[†]

The VC dimension was first studied in connection with statistical pattern recognition. Pollard and Vapnik have both written good books discussing it from that point of view [43, 54]. The first source that I am aware of to point out that it has some connection to efficient concept learning is Pearl [40].

The key fact about the VC dimension for our purposes is that a concept class has polynomial sample complexity if and only if it has finite VC dimension. In particular, if the VC dimension of concept class $\mathcal{C}$ is $d$, then Blumer et al. [10] showed that the sample complexity of $\mathcal{C}$ is bounded by

$$\text{sample complexity} \leq \max\left(\frac{4}{\epsilon}\log\left(\frac{2}{\delta}\right), \frac{8d}{\epsilon}\log\left(\frac{13}{\epsilon}\right)\right). \tag{2.8}$$

Notice that since finite VC dimension is a necessary and sufficient condition for polynomial sample complexity assuming the static sampling model, finite VC dimension is a necessary condition for static sampling pac learnability. It is not, however, a sufficient condition. We must also be able to find a concept consistent with any sample in probabilistic polynomial time. For some concept classes, such as monomials, we know how to do this. However, for many interesting concept classes the problem of finding a concept consistent with a sample is NP-complete.

[†]Except where explicitly stated otherwise, all logarithms in this work are base 2.

# Chapter 3

# A learning algorithm for submodules

## 3.1 Introduction

In this chapter we present an algorithm for learning the concept class $C_k$ of all subsets of $\mathbf{Z}^k$ closed under addition and multiplication by integers. In algebraic terms $C_k$ consists of all submodules of the free $\mathbf{Z}$-module of rank $k$. Using other terminology, $C_k$ is the set of all integer lattices contained in $\Re^k$.

We give a learning algorithm that has performance within a log log factor of optimal in the on-line mistake bound model of learning, which we describe below. That learning model is "more strict" than the pac learning model: any learning algorithm with good performance in the mistake bound model can be used as a subroutine to construct a pac learning algorithm, but not all pac learning algorithms have good mistake bounds.

In particular, we prove an absolute mistake bound for the algorithm we present of $k \log n$, where $n$ is an upper bound on the absolute value of any component of any example seen, and prove that no learning algorithm can have a mistake bound of less

than $(1-\epsilon)\frac{k \log n}{\log \log n}$ for any $\epsilon > 0$. Thus we achieve a very strong learning performance in a very strict model of learning. By way of contrast, Abe [1] presents a learning algorithm for semilinear sets, a much broader class of $k$-tuples of integers, but that algorithm has a much worse mistake bound.

The algorithm we present has a certain high level similarity to the so-called $L^3$ algorithm for lattice basis reduction [32]. In both cases a tentative basis for the lattice is maintained at all times, and certain algebraic transformations are made to the basis. However, the similarity is only superficial, because the $L^3$ algorithm in fact is solving a very different problem. The goal there is to find an new basis for the same lattice containing one vector of small Euclidean norm. In the learning problem, our goal is to converge towards a basis for the lattice of all positive instances, and every time we update our basis, the lattice generated by the it will strictly increase.

In addition to the algorithm, we also present several applications. For instance, an interesting subclass of the above class $C_k$ is the class $C'_k$ of zero-reversible commutative regular languages over alphabets of size $k$. (A sufficient but not necessary condition for a commutative regular language to be zero-reversible if for it to have an accepting DFA in which the only final state equals the start state.) Recently it has been shown by Pitt and Warmuth that for any fixed polynomial $Q$, the problem: "given a set of examples (from some $L \in C'$ accepted by a DFA of $k$ states) find a DFA or NFA with fewer than $Q(k)$ states consistent with the examples," is $NP$-hard [42]. Surprisingly the algorithm we present bypasses that hardness result by representing its hypothesis in matrix form rather than as a DFA.

## 3.2 Mistake bounded learning

There are many ways we can evaluate how well a learning algorithm has learned. We list below four possible criteria, in what intuitively appears to be the order of increasing

strictness.

1. Pac learnability—the learning algorithm must with high probability produce a hypothesis of small error [52].

2. The probability of making a mistake on predicting the label of the last instance [22].

3. The expected total number of mistakes for on-line prediction of the labels of the first $t$ instances [22].

4. The worst case total number of mistakes for on-line prediction of the labels of any (possibly infinite) sequence of instances [34].

In fact, Haussler et al. [21] have shown that the first two criteria are substantially equivalent. In this chapter we present an algorithm that learns well with respect to the fourth, strictest, criteria. It follows that this algorithm can be easily converted to a pac learning algorithm.

We follow Littlestone [34] in defining mistake based performance criteria for learning algorithms. In particular, for learning algorithm $A$ and target concept $c$ define $M_A(c)$ to be the maximum number of mistakes algorithm $A$ makes for any possible sequence of instances. For any non-empty concept class $C$, define $M_A(C) = \max_{c \in C} M_A(c)$. Any bound on $M_A(C)$ is called a *mistake bound* for algorithm $A$ applied to class $C$. The *optimal mistake bound* for concept class $C$, $opt(C)$, is the minimum over all learning algorithms $A$ of $M_A(C)$.

## 3.3 Learning submodules

We now present algorithm $SM$ for the on-line learning of the concept class $C_k$ of submodules of the free $\mathbf{Z}$-module $\mathbf{Z}^k$. The main data structure this algorithm uses is a

$k$ by $k$ upper triangular matrix $M$ which is initially all 0, and gradually has nonzero rows added to it as we see positive examples. At any point the algorithm's current hypothesis is the row span of $M$. Algorithm $SM$ makes a mistake only when it gets a new positive example $x$ is not in the span of $M$. When this happens, $M$ must be updated to a matrix whose span also includes $x$. Sometimes we can accomplish this by simply adding $x$ as a new row; more often we have to perform an operation similar to Gaussian elimination.

We begin by giving a precise description of algorithm $SM$ in section 3.3.1, and then move on to analyze its mistake rate and give several applications. Note that the algorithm itself involves standard techniques in linear algebra. The major contributions of this chapter are the analysis and the applications of the algorithm. In particular, we show that $SM$ achieves an absolute mistake bound within a $\log \log n$ factor of optimal, where $n$ is the largest (in absolute value) component of any instance seen.

We also show that $SM$ can actually be generalized to learning a submodule of any free module over a Euclidean domain, or a coset of any such submodule.

### 3.3.1 The algorithm $SM$

Throughout we maintain an upper triangular matrix $M$ of integers. Matrix $M$ is initialized to be all 0.

1. Initially, we respond "Negative" to every instance (other than $0^k$), until we eventually make a mistake on instance $x$. We then update $M$ to consist of the single row $x$.

2. At a general step, we determine whether our new instance, $x$, can be written as an integer combination of the rows in the matrix. (This can be done by back-substitution with $O(k_2)$ arithmetic operations.) If so, we respond "Positive," since it is certain that $x$ is in the submodule. Otherwise, we respond "Negative."

3. If we make a mistake, we make append $x$ as a new row to the bottom of matrix $M$. After adding the new row $x$, we must ensure that $M$ is still upper triangular of at most $k$ rows. If it is not, then we perform an operation, similar to Gaussian Elimination, which we call "reducing" $M$. (We call it reduction because it will sometimes reduce by one the number of rows in $M$.)

   The reduction process is as follows:

   For $c := 1$ to $k$ do the following to obtain a new row $c$:

   (a) If $m_{cc} = 0$, find the least $j > c$, if any exists, such that $m_{cj} \neq 0$. If there was such a $j$ swap rows $c$ and $j$.

   (b) If all rows below row $c$ have 0 in column $c$, do nothing.

   (c) Otherwise, let the nonzero entries of $M$ in this column from row $c$ downward be $\{m_{i_1,c}, m_{i_2,c} \ldots, m_{i_j,c}\}$. (Notice that $i_1 = c$.) Compute $g$, a greatest common divisor of these elements, and $d_l$ such that $g = \sum_{l=1}^{j} d_l m_{i_l,c}$. Now create a new row $c$ for $M$ by setting

   $$newrow = \sum_{l=0}^{j} d_l M(l),$$

   (where $M(l)$ is the $l$-th row of $M$) and inserting *newrow* into $M$ between rows $c-1$ and $c$.

   (d) Now zero out the entry in column $c$ of all rows below *newrow* in $M$ by subtracting out an appropriate multiple of *newrow*.

   The above steps ensure us an upper triangular matrix. Finally, we remove any rows consisting entirely of 0, and we must be left with at most $k$ rows.

The general strategy of $SM$ is that whenever we make a mistake on instance $x$ we add some elements, including $x$, to the row span of matrix $M$.

### 3.3.2 Running time

The running time of $SM$ depends on $k$, and also on the absolute value of the largest component of any number seen. Call the latter parameter $n$.

To make a prediction requires time $O(k^2)$.

Updating $M$ requires computing the greatest common divisor of $k$ numbers of size at most $n$. That requires time $O(k + \log n)$ (word steps—one division operation, for instance, is counted as one unit of time). There are up to $k$ such operations and also general matrix operations, so the total running time for an update is $O(k^3 + k \log n)$.

### 3.3.3 Mistake bound

In this subsection we calculate $SM$'s mistake bound, $M_{SM}(C_k)$ and consider how close it is to $opt(C_k)$. It turns out that the mistake bound depends on the size of the instances seen, so we denote by $C_k(n)$ the restriction of concept class $C_k$ to domain $\{-n, \ldots, 0, \ldots, n\}^k$. Notice that we are not limiting the size of the input of $SM$; we are simply describing its performance as a function of the size of its input.

First we prove a technical lemma that we will need in the analysis. This lemma says that a trivial fact about matrices with entries from $\Re$ (or any field) is also true for matrices with entries from $\mathbf{Z}$.

**Lemma 3.1** *Let $M_1$ and $M_2$ be two upper triangular matrices of integers such that the integer row space of $M_1$ is a subset of the integer row space of $M_2$. Then the number of rows of $M_2$ containing some nonzero entry is greater than or equal to the number of rows of $M_1$ containing some nonzero entry.*

*Proof* Assume WLOG that $M_1$ and $M_2$ are both square $k$ by $k$ matrices. Denote by $M(l)$ row $l$ of matrix $M$. Assume WLOG each matrix has the property that for every

$1 \leq i \leq i$ either row $i$ is all zero, or its $i,i$ entry is nonzero. Now we show that for every $1 \leq i \leq k$, if $M_1(i)$ is not all zero, then $M_2(i)$ is not all zero.

Row $M_1(i)$ is in the integer row space of $M_1$. It must therefore also be in the integer row space of $M_2$. Hence there are integers $z_1, \ldots, z_k$ such that $M_1(i) = \sum_{j=1}^{k} z_j M_2(j)$.

Assume $M_1(i)$ contains nonzero components. Then the it must be that components 1 through $i-1$ are zero, and component $i$ is nonzero. Therefore $M_1(i) = \sum_{j=i}^{k} z_j M_2(j)$. In particular, the $i,i$ entry of $M_1$ is $z_i$ times the $i,i$ entry of $M_2$. Therefore $M_2(i)$ is not all zero. □

**Theorem 3.1** $M_{SM}(C_k(n)) = k + k \lceil \log n \rceil$.

*Proof* When the algorithm responds "Positive," it is always correct. Thus to count the number of mistakes, we only need to figure out how many times the algorithm can say "Negative" on a positive example. Whenever we do this, we update matrix $M$ to a new matrix $M'$.

There are two different sorts of mistakes we can make, corresponding to two different ways $M'$ can have changed from $M$.

1. $M'$ can have one more row than $M$. At the end of processing a mistake of any sort, the matrix is returned to upper triangular form. Since the matrix's row span never decrease, it follows from Lemma 3.1 that the matrix must never decrease in number of rows. Therefore there can be at most $k$ mistakes where $M$ gains a row. (Notice that our very first mistake must always fall into this category.)

2. Otherwise, $M'$ has the same number of rows as $M$. We now analyze this case.

Claim: If at the end of a reduce we have the same number of rows as at the beginning, it must be that for every diagonal element, $m'_{ii}$ of $M'$, $m'_{ii} \mid m_{ii}$, and for some $i$, $m'_{ii}$ is not $\pm m_{ii}$.

Proof of claim: The integer row span of $M'$ must contain the integer row span of $M$. In particular, it must contain the $i$th row of $M$ which we denote by $M(i)$. The first $i-1$ components of $M(i)$ are 0, and the $i$-th component is $m_{ii}$. Writing $M(i) = \sum z_j M'(j)$, $z_j \in \mathbf{Z}$, it must be that $m_{ii} = z_i m'_{ii}$. Hence $m'_{ii} \mid m_{ii}$ as claimed.

Now we need to show that some diagonal element actually changes. We could also prove this with a linear algebra style argument, but for variety's sake, we finish the claim by examining the algorithm's function. The first time that the algorithm actually inserts a row $c$ into $M'$ that is different from $M(c)$ in step 3c must be for the least $c$ such that $m_{cc} \not| x_c$ , where $x$ was the instance not in the row span of $M$. The new $m'_{cc}$ is $\gcd(m_{cc}, x_c)$, which is a proper divisor of $m_{cc}$. This completes the proof of the claim.

It now follows that with errors of the second sort, the worst we can do is $\lceil \log n \rceil$ mistakes per diagonal element, for a total of $k \lceil \log n \rceil$. □

How good is $SM$'s performance? To answer this question, we must calculate $opt(C_k(n))$. [34] shows that the VC-dimension provides a lower bound on the optimal mistake bound.

We begin by exactly calculating the VC-dimension for $C_1(n)$ ("learning multiples of size at most $n$").

**Theorem 3.2** *Let $d$ be the VC-dimension of $C_1(n)$. Then*

$$d = \max \{r \mid 2 \cdot 3 \cdot 5 \cdots p_r \leq 2n\}$$

*where $p_i$ is the $i$th prime.*

*Proof* We start with a definition we need in our proof:

DEFINITION. Let $S$ be any shattered set; let $T \subseteq S$. We call a concept $c$ such that $T = c \cap S$ a *witness* for $T$.

Now, we show $d \geq r$. Let $r$ be maximum such that $P = \prod_{i=1}^{r} p_i \leq 2n$. First we exhibit a set $S \subset X$, $|S| = r$ that is shattered: $S$ is all products of all but one of the first $r$ primes; in symbols $S = \{P/p_i \mid 1 \leq i \leq r\}$. Since $P/p \leq n$ for all primes $p$, every element of $S$ is in the instance space. For every $x = P/p_i$ in $S$, define $\hat{x} = p_i$.

$S$ is shattered: The witness for any nonempty $T \subseteq S$ is $P/\prod_{t \in T} \hat{t}$. The witness for the empty set is 0.

Hence $d \geq r$. Now we show that $d \leq r$:

Let $S = \{x_1, x_2, \ldots x_k\}$ be shattered. First we argue that we may assume that there is no $s > 1$ that divides all elements of $S$. If there is such an $s$, we can work instead with $S' = \{x_1/s, x_2/s, \ldots x_k/s\}$ and divide each witness by $s$.

Call any subset of $k-1$ elements of $S$ a *minor*. $S$ has $k$ minors, $S_i$. Let $t_i$ be the witness for $S_i$.

Now no $t_i$ can be 1, since for every $i$ there is some $x \in S$, $t_i \not| x$. (Note that assuming $|S| > 1$, $0 \notin S$, because 0 is a positive instance of every concept save one, and $S$ is shattered.) Furthermore, $(t_i, t_j) = 1$ for all $i \neq j$, since $(t_i, t_j) \mid x$ for every $x \in S$, and we assumed that 1 is the only number with this property. Thus it much be that for each $i$ there is a prime $p_i$ such that $p_i \mid t_i$ but $p_i \not| p_j$ for any $j \neq i$.

The product of any $k-1$ $t_i$s is a witness for some one element of $S$; therefore the product must be at most $n$. Since each $t_i$ has a unique prime divisor, and the product of any $k-1$ of them must be at most $n$, the product of all of them must be at most $2n$. □

We next note the following fact from number theory. (See, eg. Hardy and Wright [19, page 263].)

**Theorem 3.3** *Let $f(n)$ be the maximum number of consecutive primes such that $\prod_{i=1}^{f(n)} p_i \leq n$. Then for every $\epsilon > 0$, for all sufficiently large $n$ we have*

$$f(n) > \frac{(1-\epsilon)\ln n}{\ln \ln n}.$$

From the preceding two theorems it follows that

**Corollary 3.1** *Let $d$ be the VC-dimension of the problem of learning multiples of size at most $n$. Then for all $\epsilon$, for sufficiently large $n$,*

$$d > \frac{(1-\epsilon)\ln n}{\ln\ln n}.$$

*Remark:* The preceding should make it clear that if we place no bound on $n$, then the VC-dimension of the problem of learning multiples is infinite.

We now get a lower bound on the VC-dimension of our true problem, learning submodules of $\mathbf{Z}^k$ for $k$ arbitrary, and hence a lower bound on the optimal mistake rate.

**Theorem 3.4** *$C_k(n)$ has for all $\epsilon$, for sufficiently large $n$,*

$$VCdim \geq k(1-\epsilon)\frac{\ln n}{\ln\ln n}.$$

*Proof sketch:* Let $S$ be the set of numbers that we shattered in the proof of Theorem 3.2. Let $U$ be the set of size $k|S|$ consisting of all $k$-tuples of integers containing $k-1$ 0s and one entry from $S$. We can shatter any $T \subseteq U$ by writing $T$ as the union of $k$ subsets of tuples, each having all its nonzero entries in the same component, and for our shattering concept selecting each component from $S$ in the same way we did in the proof of Theorem 3.2.

It follows from Theorem 3.3 that the size of $S$ has the desired property. □

As claimed, algorithm $SM$ is within a $\log\log n$ factor of optimal.

### 3.3.4 Special case: $k = 1$

For the case $k = 1$, when instances are integers and concepts are sets of multiples, instead of using algorithm $SM$, we can implement the halving algorithm [7, 34]. To

implement it, we have to perform one factorization of an instance, but that is one factorization for the whole run of the algorithm, not one per mistake.

**Theorem 3.5** *On instances of absolute value at most $n$, the halving algorithm achieves a mistake bound of $1+\max_{m\leq n}\lfloor\log\tau(m)\rfloor$ where $\tau(m)$ is the number of positive divisors of $m$.*

*Proof* The algorithm makes one mistake on the first positive instance it sees. Call this instance $m$. Without loss of generality, $0 \leq m \leq n$. Once $m$ has been seen, the divisors of $m$ are the only candidates for being the target concept, and the halving algorithm cuts the number of candidates by at least half with every mistake. Hence the mistake bound is, as claimed, $1 + \max_{m\leq n}\lfloor\log\tau(m)\rfloor$. □

The performance of the halving algorithm (See Hardy and Wright [19, page 260] for bounds on numbers of divisors of a number.) turns out to be close to the lower bound implied by Corollary 3.1 but to have a gap in multiplicative factors of $(1+\epsilon)$ versus $(1-\epsilon)$. However, the halving algorithm is, in fact, virtually optimal. In this case the VC-dimension happens not to be a tight lower bound.

**Theorem 3.6**

$$opt(C_1(n)) \geq \max_{m\leq n}\sum_{i=1}^{s}\lfloor\log(e_i+1)\rfloor$$

*where* $m = \prod_{i=1}^{s} p_i^{e_i}$.

*Proof* Let $m = \prod p_i^{e_i}$ be a number less than or equal to $n$ with a maximal number of divisors. An adversary trying to force a learning algorithm to make many mistakes would start by giving $m$ as a positive instance. The adversary can force the algorithm to do a binary search for the value of each exponent as follows. After $m$, the next set of instances begin with $p_1^{\lfloor e_1/2\rfloor}\prod_{i\geq 2} p_i^{e_i}$, and continues with various exponents for $p_1$. The algorithm can be forced into $\lfloor\log(e_1+1)\rfloor$ mistakes since there are $e_1+1$ choices for the

exponent of $p_1$. The next set of instances all have the exponent of $p_1$ set to its correct value, force the algorithm to search for the value of the exponent of $p_2$, and have the exponents of the primes starting with $p_3$ set to $e_i$. □

The halving algorithm's mistake bound is just slightly above optimal, since in the notation of Theorem 3.6 it is $1 + \max_{m \leq n} \lfloor \sum \log(e_i + 1) \rfloor$.

We can use Theorem 3.6 to obtain a somewhat better bound on $opt(C_k(n))$ than the one given by the VC dimension.

**Corollary 3.2**

$$opt(C_k(n)) \geq k \left( \max_{m \leq n} \sum_{i=1}^{s} \lfloor \log e_i + 1 \rfloor \right)$$

*where* $m = \prod_{i=1}^{s} p_i^{e_i}$.

*Proof* The method used by the adversary in the proof of Theorem 3.6 can be easily extended to the general case. There are $k$ rounds, and in each round the adversary gives examples with all components but the $k$-th set to 0. For the $k$-th component the adversary gives the values he gave in the proof of Theorem 3.6. □

### 3.3.5 Generalizations

#### Arbitrary Euclidean domains

We need not limit ourselves to submodules of $\mathbf{Z}^k$. Algorithm $SM$ can in fact learn submodules of a free $D$-module for any Euclidean domain $D$. Careful examination of the algorithm shows that it does not rely on any properties of $\mathbf{Z}$ not possessed by all Euclidean domains. This generalization gives us learning algorithms for various exotic domains such as $k$-tuples of Gaussian integers. Also, as we mention below, taking $D$ to be a field, we obtain an algorithm for learning vector subspaces.

In order to analyze the performance of algorithm $SM$ in this case, we need to introduce some definitions.

DEFINITION. Let $D$ be a Euclidean domain. Let $a \in D^*$. We define the *length of* $a$, $l(a)$, to be the total number of primes in a prime factorization[1] $a = p_1 p_2 \cdots p_r$, $p_i$ prime, of $a$. If $a$ is a unit, then define $l(a) = 1$.

*Example:* For $a \in \mathbf{Z}^*$, $a = \pm 1 \prod p_i^{e_i}$, $l(a) = \sum e_i$.

DEFINITION. Let $S \subset D^*$. Define $l(S) = \max_{a \in S} l(a)$.

*Example:* If $F$ is any field, then for any $S \subset F^*$, $l(S) = 1$.

In the general case, Theorem 3.1 becomes:

**Theorem 3.7** *The mistake bound of the submodule algorithm is $k + kl(S)$, where $S \subseteq D^*$ is the set of all nonzero elements of $D$ in all instances seen by the algorithm.*

The proof is a straightforward modification of the proof of Theorem 3.1.

*Remark:* Notice that for $n \in= \mathbf{Z}$, the measure $l(n)$ is bounded by $\log a$ (and equal to $\log a$ when $a$ is a power of 2), so we do get Theorem 3.1 when we specialize Theorem 3.7. In fact, we could have used the tighter bound of $k + kl(n)$ in the statement of Theorem 3.1.

### Cosets of submodules

A simple trick allows us to generalize the class of concepts we can learn from submodules to arbitrary cosets of submodules (viewing the submodule as an abelian group).

The algorithm still responds, "Negative" until it makes a mistake on some positive instance $x$. We now start to run the basic algorithm given in Section 3.3.1, except that we first subtract $x$ from every instance. Thus, at a total cost of one extra mistake, arbitrary cosets can be learned.

[1] As is the case for $\mathbf{Z}$, every nonzero, nonunit element of an arbitrary Euclidean domain has a unique prime factorization up to order and multiplication by units.

## 3.4 Applications of modules

### 3.4.1 Abelian groups

Algorithm $SM$ efficiently solves the learning version of the word problem for finitely generated abelian groups. We begin with a brief review of the word problem for groups. (A more complete discussion may be found in many textbooks; see Magnus, Karrass, and Solitar [35] for instance.)

Fix an infinite alphabet of symbols, $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$, and form the new symbols $\sigma_1^{-1}, \sigma_2^{-1}, \ldots$ Let $\Sigma^{-1} = \{\sigma_1^{-1}, \sigma_2^{-1}, \ldots\}$. Let $\Sigma_k = \{\sigma_1, \ldots, \sigma_k\}$ and let $\Sigma_k^{-1} = \{\sigma_1^{-1}, \ldots, \sigma_k^{-1}\}$. Let $W_k = \{(w, k) : w \in (\Sigma_k \cup \Sigma_k^{-1})^*\}$. $W_k$ is the set of all words on $k$ letters. (Technical note: It is more traditional to specify $k$ as a separate input. We eventually want to develop learning algorithms that work for arbitrary $k$, however. Thus it is convenient to make $k$ a part of our instances.)

An instance of the word problem is a finite set $S$ of words from $W_k$ for some $k$, plus one additional word $w \in W_k$. The words in the set $S$ form the presentation of a group $G$ that is unique (up to isomorphism). The problem is to determine whether the element of $G$ represented by $w$ is the identity element of $G$.

For those unfamiliar with presentation theory, we can make the preceding concrete. Intuitively the idea is that $G$ has $k$ generators, $g_1, \ldots, g_k$, and there is a map $\alpha : W_k \rightarrow G$ defined by

$$\begin{aligned} \alpha(\sigma_i) &= g_i \\ \alpha(\sigma_i^{-1}) &= g_i^{-1} \\ \alpha(w_1 w_w) &= \alpha(w_1) \cdot \alpha(w_2). \end{aligned} \tag{3.1}$$

The words in $S$ all map to $\bar{1}$, the identity element of $G$. The problem is to determine whether $\alpha(w) = \bar{1}$.

One way to obtain such a group $G$ starting with $W_k$ is as follows [35]. Throughout, if $w = s_1 s_2 \cdots s_n$ is a word in $W_k$, we denote by $w^{-1}$ the word $s_n^{-1} \cdots s_1^{-1}$ which we call the inverse of $w$. (If $s_i \in \Sigma^{-1}$, then define $s_i^{-1}$ to be the symbol in $\Sigma$ that $s_i$ is the inverse of.) We start by defining an equivalence relation $\sim$ on words in $W_k$. We say that $w_1 \sim w_2$ if applying the following operations a finite number of times to $w_1$ transform it to $w_2$.

1. Inserting any word in $S$, the inverse of any word in $S$, $\sigma_i \sigma_i^{-1}$, or $\sigma_i^{-1}\sigma$ between any two symbols of $w_1$, or before $w_1$, or after $w_1$.

2. Deleting any word in $S$, the inverse of any word in $S$, $\sigma_i \sigma_i^{-1}$, or $\sigma_i^{-1}\sigma$ if it forms a block of consecutive symbols in $w_1$.

The equivalence classes of $W_k$ induced by $\sim$ form the elements of our group. Multiplication is defined by $[w_1] \cdot [w_2] = [w_1 w_2]$. The mapping $\alpha$ is defined by $\alpha(\sigma_i) = [\sigma_i]$.

It is well known that the arbitrary word problem is undecidable. We, however, are only concerned with abelian groups.

To make the word problem into a learning problem, we define the instance space to be $\cup_k W_k$. A concept $c \subset W_k$ is in the concept class of interest to us if there is an abelian group $G$ generated by $k$ elements $g_1, \ldots, g_k$ and a map $\alpha$ defined as in (3.1), and $c = \alpha^{-1}(\bar{1})$.

We solve this learning problem by using our submodule algorithm. First we need to see how to write this problem as a submodule problem. Let $\rho_k : \mathbf{Z}^k \to W_k$ be the map defined by

$$\rho_k(z_1, \ldots, z_k) = \sigma_1^{z_1} \cdots \sigma_k^{z_k}$$

where $\sigma_i^0$ is defined to be $\lambda$ (the empty word), and $\sigma_i^{-n}$, is defined to be $(\sigma^{-1})^n$ for positive $n$.

In the other direction, let $\pi_k : W_k \to \mathbf{Z}^k$ be the mapping that counts letters,

$$\pi(w) = ((\text{number of } \sigma_1 \in w) - (\text{number of } \sigma_1^{-1}), \ldots,$$
$$(\text{number of } \sigma_k \in w) - (\text{number of } \sigma_k^{-1} \in w)). \tag{3.2}$$

We omit the subscript on $\rho$ and $\pi$ whenever it is clear. Notice that $\pi_k \circ \rho_k$ is the identity map for $\mathbf{Z}^k$.

Intuitively, if we restrict our attention to commutative groups (and languages), then $\rho$ is for practical purposes an inverse of $\pi$, since it is always the case that $w \sim \rho(\pi(w))$ where $\sim$ is the equivalence relation defined above. Thus we can use $\pi$ to map our instances to $\mathbf{Z}^k$. Throughout this section, we switch back and forth between thinking of our instances as coming from $W_k$ and $Z^k$.

Formally, our instances come from $W_k$, and we know there is a homomorphism $\alpha$ from the group on $W_k$with operation concatenation to a commutative group $G$. Instances are labeled positive if they are in the kernel of $\alpha$.

Since the composition $\nu = \alpha \circ \rho$ is a homomorphism from $\mathbf{Z}^k$ to the abelian group $G$, $\ker \nu$ must be a submodule of $\mathbf{Z}^k$. Hence if we can show that $\pi(w) \in \ker \nu$ if and only if $w \in \ker \alpha$, then all we need do is use $\pi$ to map all our instances to $\mathbf{Z}^k$ (keeping the label the same), and run our submodule algorithm.

We noted above that $\rho(\pi(w)) \sim w$, so $\alpha(\rho(\pi(w))) = \alpha(w)$. Thus, as desired, it is the case that $\pi(w) \in \ker \nu$ if and only if $w \in \ker \alpha$.

### 3.4.2 Some commutative languages

We can also use algorithm $SM$ to learn certain classes of commutative languages with a very good mistake bound. (We call a language $L$ *commutative* if $w \in L \Rightarrow$ all permutations of $w$ are in $L$.) Among these classes is the class of commutative zero-reversible languages. By way of contrast, Angluin [3] provides a family of algorithms

for learning the $k$-reversible languages from positive samples. Those algorithms achieve correct identification in the limit, but may make a very large number of mistakes.

Throughout this section, $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is the alphabet that regular languages are over, and $k$ is $|\Sigma|$. We define $\Sigma^{-1}$ to be $\{\sigma_1^{-1}, \ldots, \sigma_k^{-1}\}$, and let $\pi$ be the map $\pi_k$ defined above in equation (3.2).

Roughly speaking, we can learn any language $L$ whose image under $\pi$ is a coset of a submodule of $\mathbf{Z}^k$. To be more precise, however, we must take account of the fact that languages are defined only over $\Sigma$, so their images under $\pi$ are always be $k$-tuples of nonnegative numbers. If $S \subseteq \mathbf{Z}_k$, let us define

$$S^+ = \{(x_1, \ldots, x_k) \in S \mid x_i \geq 0, 1 \leq i \leq k\}.$$

We call $S+$ the *nonnegative restriction of* $S$.

**Theorem 3.8** *Let $\mathcal{C}$ be the class of all commutative languages $L$ such that $\pi(L) = N^+$ for some $N \in \mathbf{Z}^k$ that is the coset of a submodule of $\mathbf{Z}^k$. We can can learn $\mathcal{C}$ with a mistake bound of $1 + k + k\lceil \log n \rceil$ where $n$ is the length of the longest instance seen.*

*Proof* We simply apply the map $\pi$ to instances and then use the coset modification of algorithm $SM$. □

*Remark:* There are some elements $x \in N$ such that $\rho(x) \notin L$, and those elements are never seen by our algorithm. That does not cause problems, however, since an absolute mistake bound must hold no matter which subset of the instances is presented to the algorithm.

Notice that not all the languages in $\mathcal{C}$ are regular. For instance, $\mathcal{C}$ contains the language

$$L = \{w | w \text{ contains two times as many } \sigma_1 \text{ as } \sigma_2\}$$

which is not regular.

In the remainder of this subsection we show that the commutative zero-reversible languages are also in $\mathcal{C}$. For the sake of convenience, we repeat the definition of zero-reversible language [3] here:

DEFINITION. A DFA is formally a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$ where $q_0 \in Q$, $F \subset Q$, and $\delta$ is a partial function from $Q \times \Sigma$ to $Q$. The elements of $Q$ are called states; $\delta$ is called the transition function; $q_0$ is called the initial or start state; the states in $F$ are called accepting states. Such a DFA is *zero reversible* if (1) it has at most one accepting state, and (2) no state has more than one incoming transition for any letter $a \in \Sigma$. We say that a regular language $L$ is zero reversible if $L$ is accepted by some zero-reversible DFA.

Every regular language is accepted by a minimal state DFA that is unique up to the naming of the states. We call such a DFA *canonical.* Throughout the following discussion we restrict our attention to canonical DFAs. This restriction causes no loss of generality, since we are trying to devise learning algorithms which receive only words from $\Sigma^*$ as their inputs. We note some properties of canonical DFAs that we will use:

1. Every state is reachable from the start state.

2. If the language is commutative and word $v$ is any permutation of word $w$, then $\delta(q, w) = \delta(q, v)$ for every state $q$.

(Further discussion of canonical DFAs can be found in automata theory texts such as Hopcroft and Ullman [24].)

**Lemma 3.2** *Let $L$ be a commutative zero-reversible language which has as its canonical accepting DFA some $M = (Q, \Sigma, \delta, q_0, F)$ with $F = \{q_0\}$. Then $\pi(L)$ is the nonnegative restriction of some submodule of $\mathbf{Z}^k$.*

*Proof* Since $L$ is zero reversible, the behavior of the DFA for $L$ is well defined over $\Sigma \cup \Sigma^{-1}$. To make that notion precise, define the new DFA $M' = (Q, \Sigma \cup \Sigma^{-1}, \delta', q_0, \{q_0\})$,

where

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{if } a \in \Sigma \\ r \text{ such that } \delta(r,a^{-1}) = q & \text{if } a \in \Sigma^{-1} \text{ and such } r \text{ exists} \\ \text{undefined} & \text{otherwise} \end{cases}$$

We will show that $\pi(L(M'))$ is a submodule of $\mathbf{Z}^k$. Since $L = L(M') \cap \Sigma^*$, this is sufficient to complete the proof.

First we argue that $L(M')$ is commutative. It is sufficient to show that for all $q \in Q$, $\delta'(q,ab) = \delta'(q,ba)$ for arbitrary $a,b \in \Sigma \cup \Sigma^{-1}$. (Actually we will not consider any letter that occurs in no word in $L$, or the inverse of any such letter.) There are three cases:

1. Both $a,b \in \Sigma$. In this case, $\delta'$ is the same as $\delta$.

2. Both $a,b \in \Sigma^{-1}$. Say $a = \hat{a}^{-1}$ and $b = \hat{b}^{-1}$, where $\hat{a},\hat{b} \in \Sigma$. Now $\delta'(q,ab)$ must be some state $r$ such that $\delta(r,\hat{b}\hat{a}) = q$. Since $M$ is a canonical machine for a commutative language, $\delta(r,\hat{a}\hat{b}) = \delta(r,\hat{b}\hat{a}) = q$. Therefore $\delta'(q,ba) = r$ as well.

3. The letter $a \in \Sigma$, and the letter $b \in \Sigma^{-1}$. Again, say $b = \hat{b}^{-1}$ where $\hat{b} \in \Sigma$. Now let $r = \delta'(q,b)$, let $s = \delta'(r,a)$, and let $t = \delta'(q,a)$. $\delta'(q,ba)$ is obviously $s$. Now we need to calculate $\delta'(q,ab) = \delta'(t,b)$.

   It must be that $\delta(r,a\hat{b}) = \delta(r,b\hat{a})$. Now since $r = \delta'(q,b)$, it follows that $\delta(r,\hat{b}) = q$. Therefore $t = \delta(r,\hat{b}a) = \delta(r,a\hat{b}) = \delta(s,\hat{b})$. Thus, as desired, $s = \delta'(t,b)$. (See figure 3-1.)

Now we must show that $\pi(L(M'))$ meets the submodule properties: The all zero tuple is $\pi(\lambda)$, and $\lambda \in L(M')$, so $\pi(L(M'))$ contains the identity element.

Now $\pi(w_1) + \pi(w_2) = \pi(w_1 \cdot w_2)$, and if both $w_1$ and $w_2$ are in $L(M')$, then so is $w_1 \cdot w_2$. Thus $\pi(L(M'))$ is closed under addition.

**Figure 3-1**: Detail of proof of Lemma 3.2: $\delta(q, ab) = \delta(q, ba)$.

Multiplying the tuple for word $w$ by positive integer $n$ corresponds to taking the concatenation of $n$ copies of $w$, so $\pi(L(M'))$ is closed under nonnegative integer multiplication.

Finally, we must show that if $w \in L(M')$, then $-\pi(w) \in \pi(L(M'))$. Let $w = s_1 s_2 \cdots s_k$ and define $w^{-1} = s_1^{-1} s_2^{-1} \cdots s_k^{-1}$, (where if $s_i \in \Sigma^{-1}$, then $s_i^{-1}$ is the letter from $\Sigma$ that $s_i$ is the inverse of). Observe that $\pi(w^{-1}) = -\pi(w)$. It is clear that the path through the machine traveled on symbols $s_k^{-1} \cdots s_1^{-1}$ must be the reverse of the path traveled on the symbols in $w$; thus $w^{-1} \in L$. □

**Theorem 3.9** *Let $L$ be an arbitrary commutative zero-reversible language. The image of $L$ under $\pi$ is the nonnegative restriction of a coset of a submodule of $\mathbf{Z}^k$.*

*Proof* Let $M = (Q, \Sigma, \delta, q_0, F)$ be the canonical DFA accepting $L$. Since $L$ is zero reversible, $F$ consists of exactly one state. Call that state $q_f$. If $q_f = q_0$, then Lemma 3.2 applies and we are done.

Otherwise, let $M'$ be the extension of $M$ from alphabet $\Sigma$ to alphabet $\Sigma \cup \Sigma^{-1}$ in a manner analogous to the proof of Lemma 3.2. The image under $\pi$ of the set $X$ of strings $x$ such that $\delta'(q_0, x) = q_0$ is a submodule by Lemma 3.2.

Let $w$ be any string in $L$. The language accepted by $M'$ is just the set

$$\{xw \mid x \in X\}.$$

Hence $\pi(L(M')) = \pi(w)+\pi(X)$ which is a coset of the submodule $\pi(X)$. Since $L(M) = L(M')^+$, we are done. □

*Remark:* There is in fact a strong connection between the zero-reversible languages and the word problem for abelian groups, which we discussed in the previous subsection. One can show that the transition diagram of the canonical DFA for a zero-reversible language forms a Cayley graph, or graph of a group. (See such books as [12, 18, 35] for a discussion of Cayley graphs.) Furthermore, if the language is commutative, then the graph is the graph of an abelian group.

**Corollary 3.3** *We can learn the class of zero-reversible commutative languages with an absolute mistake bound of*

$$1 + |\Sigma| (1 + \lceil \log n \rceil)$$

*where $n$ is the length of the longest instance seen.*

*Proof* This follows immediately from Theorems 3.8 and 3.9. □

Corollary 3.3 is somewhat surprising in light of the results of Pitt and Warmuth [42]. They show that there is a particular subset of the class of commutative zero-reversible languages (the "counter languages") for which the minimum consistent DFA problem cannot be approximated within any polynomial.

### Vector subspaces

If we generalize from $\mathbf{Z}$-modules to $F$-modules for an arbitrary field $F$, we obtain the obvious algorithm for learning subspaces of a vector space with a mistake bound of the dimension of the subspace being learned. (This algorithm is also a special case of Shvayster [50].)

# Chapter 4

# Learning from noisy data

## 4.1 Introduction

In Chapters 2 and 3, we have discussed certain learning algorithms. Implicit in our analysis of those algorithms was the assumption that the examples input to the learner were correct. Thus if 17 was a positive instance of some concept, which was output by EXAMPLES, then we assumed that our learner received the example $(17, +)$ and not $(71, +)$ or $(17, -)$. In the real world, however, we are not always so fortunate. Often our data will be afflicted by noise. In this chapter we will examine what the effects are of different sorts of noise.

Some of the very first algorithms proposed for pac learning depended critically on the assumption of perfect, noiseless data. It was immediately recognized that this dependence is undesirable, and in several slightly later theoretical learning papers [51, 5, 25, 31], algorithms were presented that could withstand some amount of various sorts of noise.

In the theoretical literature, two models of noise have been studied so far to determine their effect on pac learning. (Quinlan [44] has done an interesting *empirical* study

of the effects of different sorts of noise on learning algorithms.) One is malicious noise [51]: when the learner requests an example, with some probability $\nu$, an omnipotent, omniscient adversary gets to replace the example with any example of the adversary's choice. The goal of this model is to capture the worst possible sort of noise. Kearns and Li [25] show that, given certain very minimal assumptions, in order to learn a concept class with this sort of noise model, the noise rate must be strictly less than $\epsilon/(1+\epsilon)$ where $\epsilon$ is as usual the accuracy parameter.

The other model of noise that has been studied is random classification noise. In this model, when the learner requests an example, with probability $1-\nu$ the learner gets the correct example; otherwise the learner receives the example with all its attributes unaltered by noise, but with the wrong label. Angluin and Laird [5] show that the information theoretic bound on learning with such noise is $\nu < 1/2$. They further show that any algorithm which works by choosing as its output concept some concept which minimizes disagreements with a polynomial size set of examples meets this bound, and also give an efficient algorithm for learning $k$DNF for any $\nu < 1/2$.

In this chapter we show that these two models of noise can be "pushed towards one another." We exhibit a more severe model of noise than random misclassification (malicious misclassification) for which we can still tolerate $\nu < 1/2$. On the other hand, we show that even with a much gentler model of noise than malicious noise (random attribute noise) any algorithm which works by minimizing disagreements can only tolerate $\nu < \epsilon$.

## 4.2 Notation

The definition of pac learning (from noiseless data) assumes that the learner trying to learn concept $c$ has available to it a black box or oracle called EXAMPLES, and that each call to EXAMPLES returns a labeled instance, $(x, s)$. We will introduce new

oracles which we will use to model the case where the labeled examples given to our learner are somehow corrupted by noise.

To obtain the malicious noise model we will use the *malicious error oracle*, $\mathrm{MAL}_\nu$. To obtain the most benign error model, the random misclassification model, we will use the *random misclassification oracle*, $\mathrm{RMC}_\nu$. In between we introduce two new oracles, the *malicious misclassification oracle*, $\mathrm{MMC}_\nu$, and the *random attribute error oracle*, $\mathrm{RAT}_\nu$. Each of these four oracles is defined with respect to a fixed target concept $c \in \mathcal{C}$ and a fixed probability distribution $\mathbf{P}$ on on the instance space.

Each of these oracles represents some sort of noisy version of EXAMPLES. The "desired," noiseless output of these oracles would thus be a correctly labeled point $(x, s)$, where $x$ is drawn according to $\mathbf{P}$. Now we describe the actual outputs of these oracles.

- When $\mathrm{MAL}_\nu$ is called, with probability $1 - \nu$, it does indeed return a correctly labeled $(x, s)$ where $x$ is drawn according to $\mathbf{P}$. With probability $\nu$ it returns an example $(x, s)$ about which no assumptions whatsoever may be made. In particular, this example may be maliciously selected by an adversary who has infinite computing power, and has knowledge of $c$, $\mathbf{P}$, $\nu$, and the internal state of the algorithm calling this oracle. This oracle is meant to model the situation where the learner usually gets a correct example, but some small fraction $\nu$ of the time the learner gets noisy examples and the nature of the noise is unknown or unpredictable.

- When $\mathrm{RMC}_\nu$ is called, it calls EXAMPLES to obtain some (noiseless) $(x, s)$, and with probability $1 - \nu$, $\mathrm{RMC}_\nu$ returns $(x, s)$. With probability $\nu$, $\mathrm{RMC}_\nu$ returns $(x, \bar{s})$ (i.e., $x$ with the wrong label). In this model the only source of noise is random misclassification.

- When $\mathrm{MMC}_\nu$ is called, it also calls EXAMPLES to obtain some (noiseless) $(x, s)$,

and, with probability $1-\nu$, $\mathrm{MMC}_\nu$ returns $(x, s)$. With probability $\nu$ $\mathrm{MMC}_\nu$ returns $(x, l)$ where $l$ is a label about which no assumptions whatsoever may be made. As with $\mathrm{MAL}_\nu$ we assume an omnipotent, omniscient adversary; but in this case the adversary only gets to choose the label of the example. This oracle is meant to model a situation where the only source of noise is misclassification, but the nature of the misclassification is unknown or unpredictable.

- We consider the oracle $\mathrm{RAT}_\nu$ only when we are learning boolean functions. $\mathrm{RAT}_\nu$ calls EXAMPLES and obtains some $(x_1 \cdots x_n; s)$. $\mathrm{RAT}_\nu$ then adds noise to this example by independently flipping each bit $x_i$ to $\bar{x}_i$ with probability $\nu$ for $1 \leq i \leq n$. Note that the label of the "true" example is never altered by $\mathrm{RAT}_\nu$. This oracle is meant to model a situation where the attributes of the examples are subject to noise, but that noise is as benign as possible.

Now we can define what it means to pac learn given noisy examples:

DEFINITION. We say that algorithm $A$ *pac learns* $C$ *from noisy examples of type* $O_\nu$ if and only if for every $c \in C$, for every probability distribution on $X$, for every positive $\epsilon$ and $\delta$, algorithm $A$, given only $\epsilon, \delta$ access to oracle $O_\nu$ for $c$, and an upper bound $\nu_b$ on $\nu$, meets the following two criteria.

**Learning criterion** Algorithm $A$ outputs some representation of a concept $c'$ such that

$$\Pr[c' \text{ is an } \epsilon\text{-approximation of } c] \geq 1 - \delta$$

where the probability is taken over the output of $O_\nu$ and any coin tosses $A$ may make.

**Efficiency criterion** The running time of $A$ is bounded by some polynomial function of $1/\epsilon$, $1/\delta$, $1/(1-2\nu_b)$, the length of an instance, and the length of the target concept.

We define s-pac learning from noisy examples in an analogous manner.

Notice that this definition only requires that the algorithm be given a bound on the noise rate. In fact the real source of examples for the algorithm may be $O_\nu$ for any $0 \leq \nu \leq \nu_b$.

DEFINITION. The *optimal error rate for concept class $\mathcal{C}$ given errors of type $O$*, $E(\mathcal{C}, O)$ is defined to be the largest $\nu$ such that there exists some algorithm $A$ which s-pac learns $\mathcal{C}$ from noisy examples of type $O_\nu$.

In some cases we will only consider those learning algorithms which work by drawing a set $S$ of (noisy) examples and outputting some $c \in \mathcal{C}$ such that the number of examples in $S$ which $c$ classifies differently from their labels is minimized.

DEFINITION. The *optimal error rate for algorithms which work by minimizing disagreements on concept class $\mathcal{C}$ given errors of type $O$*, $EMD(\mathcal{C}, O)$, is defined to be the largest $\nu$ such that there exists some algorithm $A$ which s-pac learns $\mathcal{C}$ from noisy examples of type $O_\nu$ by minimizing disagreements.

The above definitions don't mention computation at all. We'll say that the *optimal polynomial time error rate for concept class $\mathcal{C}$ given errors of type $O$*, $E^P(\mathcal{C}, O)$ is the largest $\nu$ such that there exists some algorithm $A$ which pac learns $\mathcal{C}$ from noisy examples of type $O_\nu$, and analogously for $EMD^P(\mathcal{C}, O)$.

## 4.3 Main results

### 4.3.1 Attribute noise

Let us define a concept class $\mathcal{C}$ to be *distinct* if there are concepts $c_1, c_2 \in \mathcal{C}$ and points $u$ and $v$ in the domain of $\mathcal{C}$ satisfying $u, v \in c_1, u \notin c_2$, and $v \in c_2$. Kearns and Li [25] show effectively:

**Theorem 4.1 (Kearns and Li)** *Let $C$ be a distinct concept class, and $\epsilon$ the accuracy parameter. Then $E(C, \mathrm{MAL}) < \epsilon(1+\epsilon)$.*

(Their definition of distinct is slightly different because they work with the two button model of learning.)

Now let us restrict our attention to concepts which are boolean functions. We'll say a concept class $C = \bigcup_{n=1}^{\infty} C_n$ is *instance distinct* if for each $n$ there are concepts $c_1, c_2 \in C_n$ and an instance $u$ in $\{0,1\}^n$ satisfying $c_1(u) \neq c_2(u)$ and for all $v \neq u$, $c_1(v) = c_2(v)$ Thus for a concept class to be instance distinct, all that is required is the existence of two concepts which agree on every instance in the domain save one. Most common boolean concept classes, including $k$DNF, DNF, CNF, etc., are not only distinct but also instance distinct.

For instance distinct concepts, restricting our attention to algorithms which work by minimizing disagreements, we get only slightly weaker results than Kearns and Li [25] when we substitute the much gentler oracle $\mathrm{RAT}_\nu$ for $\mathrm{MAL}_\nu$:

**Theorem 4.2** *Let $C$ be an instance distinct concept class, and $\epsilon$ the accuracy parameter. Then $EMD(C, \mathrm{RAT}) < \epsilon$.*

*Proof* We use the technique of induced distributions [25].

Fix $n$, and let $c_1, c_2$ be the two concepts which cause $C$ to be instance distinct. Let $u$ be the instance in the domain on which $c_1$ and $c_2$ differ, and let $v$ be any instance in the domain that differs from $u$ in only one bit position.

Now assume our algorithm is trying to pac learn to accuracy $\epsilon$, and fix the probability distribution $\mathbf{P}$ on the domain which assigns probability $\epsilon$ to $u$ and $1-\epsilon$ to $v$.

Say without loss of generality that $c_1(u) = c_1(v) = c_2(v) = +$, and $c_2(u) = -$. In the absence of noise, when our algorithm obtains examples from the oracle, it will see the following distribution:

| Concept | $(u,+)$ | $(u,-)$ | $(v,+)$ | $(v,-)$ |
|---|---|---|---|---|
| $c_1$ | $\epsilon$ | 0 | $1-\epsilon$ | 0 |
| $c_2$ | 0 | $\epsilon$ | $1-\epsilon$ | 0 |

If, however, the examples come not from EXAMPLES but from $\mathrm{RAT}_\nu$, the examples will have the following distribution (ignoring instances other than $u$ and $v$):

| Concept | $(u,+)$ | $(u,-)$ | $(v,+)$ | $(v,-)$ |
|---|---|---|---|---|
| $c_1$ | $\epsilon\bar{\nu}^n + (1-\epsilon)\nu\bar{\nu}^{n-1}$ | 0 | $(1-\epsilon)\bar{\nu}^n + \epsilon\nu\bar{\nu}^{n-1}$ | 0 |
| $c_2$ | $(1-\epsilon)\nu\bar{\nu}^{n-1}$ | $\epsilon\bar{\nu}^n$ | $(1-\epsilon)\bar{\nu}^n$ | $\epsilon\nu\bar{\nu}^{n-1}$ |

where $\bar{\nu} = 1-\nu$.

Now, in the case where $c_2$ is in fact the correct concept, we must have that the "observed mistake rate" of $c_2$ is strictly less than the observed mistake rate of $c_1$. Since $c_1$ and $c_2$ classify all instances other than $u$ identically, this amounts to:

$$(1-\epsilon)\nu(1-\nu)^{n-1} < \epsilon(1-\nu)^n \tag{4.1}$$

which simplifies to $\nu < \epsilon$ as desired. □

We can define an attribute noise oracle that is somewhat more gentle than even $\mathrm{RAT}_\nu$, and obtain a somewhat weaker bound. Oracle $\mathrm{RAT}_\nu'$ will obtain a correct example $(x_1 \cdots x_n; s)$ from EXAMPLES, and with probability $1-\nu$ output that example unaltered. Instead of altering each bit with probability $\nu$ like $\mathrm{RAT}_\nu$, $\mathrm{RAT}_\nu'$ will with probability $\nu$ pick some $1 \leq i \leq n$ randomly and uniformly, and output $(x_1, x_2, \ldots, \bar{x}_i, \ldots, x_{n-1}, x_n; s)$ (i.e., the correct example with exactly one of its attribute bits flipped).

**Corollary 4.1** *Let $\mathcal{C}$ be an instance distinct concept class, and $\epsilon$ the accuracy parameter. Then $EMD(\mathcal{C}, \mathrm{RAT}') < n\epsilon$.*

Theorem 4.2 and its corollary are somewhat surprising, especially since we will see below that much larger amounts of malicious classification noise can be tolerated. These results are sharply different from those of Quinlan [44], who found empirically that attribute noise was less harmful than classification noise. Perhaps the difference comes about because the definition of pac learning causes us to examine worst case probability distribution on the instance space.

We stated Theorem 4.2 and its corollary in the one-button model of learning. It is easy to convert these results to the two-button model of pac learning by using the techniques of Haussler et al. [21].

Note that it is crucial in obtaining the above results that one only consider algorithms that work by minimizing disagreements. In fact, one can pac learn $k$DNF getting examples from $\mathrm{RAT}_\nu$, for any $\nu < 1/2$, by using a different strategy [49]. That algorithm does, however, depend on the precise noise rate $\nu$—not merely a bound on $\nu$—being given as an input.

### 4.3.2 Misclassification noise

For the most benign errors, those generated by $\mathrm{RMC}_\nu$, Angluin and Laird have shown that s-pac learning is possible whenever $\nu < 1/2$. In particular, they showed that a modified version of the static consistent algorithm specified in Chapter 2 will work.

There are two key modifications. First, the sample size $m$ must depend on some bound $\nu_b$ on the noise rate $\nu$ in addition to the usual parameters.

Also, we cannot simply pick some concept $c \in C$ that agrees with all of the labels of the instances in the sample. Since those labels are noisy, there may not be any such $c$. For every concept $c \in C$, define $d_c$, the *disagreement number of* $c$, to be the number of labeled instances $(x, s)$ in the sample for which $c(x) \neq s$. The *disagreement rate* of $c$ is defined to be $d_c/m$. The output of the learning algorithm which tolerates

classification noise is any $c_*$ with a minimal disagreement number (or, equivalently, a minimal disagreement rate.)

The following theorem shows that this algorithm indeed pac learns:

**Theorem 4.3 (Angluin and Laird)** *Let $\mathcal{C}$ be any finite concept class.*[1] *If we draw a sample of size*

$$m = \frac{2}{\epsilon^2(1-2\nu_b)^2} \ln\left(\frac{2\,|\mathcal{C}|}{\delta}\right) \tag{4.2}$$

*from $RMC_\nu$ for any $\nu \leq \nu_b < 1/2$, and find any hypothesis $c_*$ with a minimal disagreement number, then*

$$\Pr\left[c_* \text{ is an } \epsilon\text{-approximation of the target concept}\right] \geq 1-\delta.$$

*Remark:* Laird [31] has shown that in fact $m = O(1/\epsilon)$ rather than $m = O(1/\epsilon^2)$ is sufficient.

Before proving Theorem 4.3, we first introduce a lemma from probability theory that we will need for this proof, as well as elsewhere throughout this thesis.

**Lemma 4.1 (Hoeffding's Inequality)** *Let $X_1, X_2, \cdots, X_m$ be independent 0-1 random variables each with probability $p$ of being 1. Let $S = \sum_1^m X_i$. Let $t$ and $\alpha$ be any positive constants.*

$$\Pr[S \geq pm + tm] \leq e^{-2mt^2} \tag{4.3}$$

$$\Pr[S \geq \alpha m] \leq e^{-2m(\alpha-p)^2} \tag{4.4}$$

$$\Pr[S \leq \beta m] \leq e^{-2m(\beta-p)^2} \tag{4.5}$$

*for $\alpha \geq p$ and $\beta \leq p$.*

[1] Recall from Chapter 2 that we consider a concept class $\mathcal{C} = \cup_{n=1}^{\infty} \mathcal{C}_n$ to be finite if $|\mathcal{C}_n|$ is finite for every $n$.

*Proof* See Hoeffding [23]. □

What Hoeffding's Inequality says intuitively is that if we run $m$ Bernoulli trials each with probability of success $p$, then the chance of getting a number of successes different from $pm$ by a constant fraction is exponentially vanishing.

*Proof of Theorem 4.3.*

Our examples have random misclassification noise with noise rate $\nu$. (The rate $\nu$ is fixed, but all the learning algorithm knows is that the noise rate is between 0 and $\nu_b$.) Consider a concept $c$ which disagrees with the target concept $c_t$ on probability weight $p$ of instances in the instance space. The expected disagreement rate of $c$ with a sample from $\mathrm{RMC}_\nu$ is

$$\mathrm{E}\,[\text{disagreement rate}] = (1-\nu)p + \nu(1-p). \tag{4.6}$$

For the target concept the expected disagreement rate is simply $\nu$. For any $\epsilon$-bad concept, the expected disagreement rate is at least $\nu+\epsilon(1-2\nu)$. The difference between those two rates, which we will denote by $g$ (for gap) is

$$g = \epsilon(1-2\nu) \tag{4.7}$$

$$\geq \epsilon(1-2\nu_b). \tag{4.8}$$

As long as the measured disagreement rate of the target concept is less than its expectation plus $g/2$, and the measured disagreement rate of every $\epsilon$-bad concept is greater than its expectation minus $g/2$, then every concept with a minimal disagreement rate must be $\epsilon$-good, which is what we need for pac learning.

By Hoeffding's Inequality we have that the probability that the disagreement rate of the target concept exceeds its expectation by as much as $g/2$ is at most

$$e^{-2(g/2)^2 m} \leq \frac{\delta}{2\,|\mathcal{C}|}.$$

Similarly, the probability that any one particular $\epsilon$-bad rule has a disagreement rate as much as $s/2$ less than its expectation is at most $\delta/2\,|\mathcal{C}|$, so the probability that any

$\epsilon$-bad rule does so is at most $\delta/2$. Thus, as desired the probability of outputting an $\epsilon$-bad rule is at most $\delta$. □

In fact, malicious misclassification is no more harmful:

**Theorem 4.4** *Theorem 4.3 holds with $RMC_\nu$ replaced by $MMC_\nu$.*

*Proof* The argument is similar to the proof of Theorem 4.3, except that now we must consider what happens if our examples are maliciously misclassified.

The difference between $\mathrm{RMC}_\nu$ and $\mathrm{MMC}_\nu$ is that in $\mathrm{MMC}_\nu$, the fraction $\nu$ of the time that "the coin toss comes up heads," the devil may choose *not* to misclassify the example. The devil may thus make some incorrect concept bad rule appear better to the learner than it would if the noise were purely random.

Let $\mathcal{E}$ be the event that the disagreement number of the target concept is less than the disagreement number of any $\epsilon$-bad concept.

In the proof of Theorem 4.3, the way we showed that we get an $\epsilon$-good rule with probability at least $1-\delta$ in the case of $\mathrm{RMC}_\nu$ was to show that $\Pr[\mathcal{E}] \geq 1-\delta$. If event $\mathcal{E}$ occurs when the examples are randomly misclassified, then it will still occur even if the devil chooses not to misclassify some (or all) of the examples misclassified by $\mathrm{RMC}_\nu$. Not mislabeling may cause the disagreement number of some $\epsilon$-bad concepts to decrease by 1 (or to increase by 1), but it definitely causes the disagreement number of the target concept by 1, so if event $\mathcal{E}$ occurs with mislabeling, then it still occurs without mislabeling.

In fact, this reasoning would hold even if the devil got to switch bad labels. That is to say, if the sample returned by $\mathrm{RMC}_\nu$ caused event $\mathcal{E}$ to occur, and that sample contained the two labeled instances ($x_1$,true label of $x_1$) and ($x_2$,wrong label for $x_2$), then if those two labeled instances were replaced in the sample by ($x_1$,wrong label for $x_1$) and ($x_2$,true label of $x_2$), then event $\mathcal{E}$ would still occur. □

**Corollary 4.2** *Theorem 4.4 still holds for static pac learning even if $MAL_\nu$ is replaced by a noise model where first the entire sample is drawn from* EXAMPLES *and then the adversary is allowed to pick any subset of the sample up to a fraction $\nu$ of the total sample to mislabel.*

Our techniques for handling malicious misclassification hold up computationally: Angluin and Laird [5] give an algorithm to polynomial time learn $k$DNF with in the presence of noise from $\mathrm{RMC}_\nu$ for any $\nu < 1/2$; we can strengthen that algorithm to tolerate noise from $\mathrm{MMC}_\nu$ for any $\nu < 1/2$, thus showing:

**Theorem 4.5** *For any $c_0 < 1/2$, $E^P(k\mathrm{DNF}, \mathrm{MMC}) \geq c_0$.*

The proof follows from the same sort of modification to the proof for the case of $\mathrm{RMC}_\nu$ as was used in the previous theorem. Again, the algorithm will in fact tolerate the somewhat worse sort of noise described in Corollary 4.2.

Note that $\mathrm{MMC}_\nu$, and even more so the oracle described in Corollary 4.2, are very strong models of misclassification noise. In particular, the latter could be used to well model the case where "borderline" instances from the domain are misclassified much more often than "obvious" instances—a case that is appealing to intuition, and that Amsterdam [2] suggests be studied as a more realistic alternative to purely random misclassification.

# Chapter 5

# Learning concepts reliably and usefully

## 5.1 Introduction

The pac learning model has, as we discussed in Chapter 2, many desirable features. However, there is one thing about it that certainly is *not* desirable: there are concept classes that we cannot learn within the pac learning model (given various cryptographic assumptions generally believed to be true) [52, 41, 42, 27].

The largest concept class that an optimist might hope to be able to learn efficiently in some model of concept learning is the set $\mathcal{C}^{\text{poly}}$ of all concepts that can be represented by polynomial size circuits. For any larger class, just deciding whether a given instance is in a particular concept may be computationally intractable. The infeasibility of learning $\mathcal{C}^{\text{poly}}$ in the pac learning model was shown in Valiant's original pac learning paper using cryptographic tools from Goldreich, Goldwasser, and Micali [52, 15].

In this chapter we examine a model of learning with a more powerful teacher than the EXAMPLES oracle of the pac learning model that allows us to learn this "most

optimistic" concept class.

### 5.1.1 Hierarchical learning

The way we escape the infeasibility of learning arbitrary concepts is by first learning relevant subconcepts of the target concept, and then learning the target concept itself.

Learning by first learning relevant subconcepts has been a useful technique elsewhere in the field of learning:

- Cognitive psychologists believe that one way humans learn is by first organizing simple knowledge into "chunks," and then using these chunks as subconcepts in later learning [36].

- In the artificial intelligence community, the builders of the *Soar* computer learning system have built a system that saves useful "chunks" of knowledge acquired in the current learning task for use as subconcepts in future learning tasks [30, 29]. Also, the SIERRA system learns how to do arithmetic in a manner broadly similar to what we suggest; it learns "one subprocedure per lesson." [53]

- Within the framework of theoretical inductive inference, Angluin, Gasarch, and Smith [4] recently showed how to learn certain otherwise unlearnable recursive functions by first learning relevant subconcepts.

### 5.1.2 A new variation on the Valiant model

We introduce here a new definition of learning which is very similar to but more stringent than pac learning. In pac learning, the learner must give as output a concept in whatever representation is being worked with—say circuits. Our learner is instead supposed to give a (polynomial time) program taking instances as input, and having three possible outputs: "Yes," "No," and "I don't know."

DEFINITION. We call learning algorithm $A$ *reliable* if the program output by $A$ says "Yes" only on positive instances, and says "No" only on negative instances of the target concept.

Of course, given that definition of reliable, it is very easy to design a reliable learning algorithm: Have the learning algorithm look at no examples, and output the program which just gives the useless answer "I don't know" on all instances. Informally we call a learning algorithm output useful if the program it outputs says "I don't know" on at most a fraction $\epsilon$ of all instances, where $\epsilon$ is an input to the learning algorithm. Formally, we make the following definition, analogous to pac learning:

DEFINITION. Let $\mathcal{C}$ be a class of concepts on domain $X = \cup_{n=1}^{\infty} X_n$. We say algorithm $A$ *reliably and probably usefully learns* $\mathcal{C}$ if and only if for every positive $n$, for every $c \in \mathcal{C}_n$, for every probability distribution $\mathbf{P}$ on $X$, for every positive $\epsilon$ and $\delta$, algorithm $A$, given only $\epsilon, \delta$ and access to EXAMPLES($c$) halts in time polynomial in $n$, $|c|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, and outputs a program $Q$ that satisfies the following conditions.

1. For every $x \in X_n$, $Q(x) = \text{Yes} \Rightarrow c(x) = 1$, and $Q(x) = \text{No} \Rightarrow c(x) = 0$. ($A$ is reliable.)

2. With probability at least $1 - \delta$,

$$\sum_{Q(x)=\text{ I don't know}} \mathbf{P}(x) < \epsilon.$$

   (A is probably useful.)

The above definition is similar to the definition of pac learning, in that both definitions require the learner to find some concept that probably agrees with the target concept most of the time. Our new definition is stronger than pac learning in that we require, in addition, that the output of learner must *never* misclassify an instance. It must somehow "know enough" to say "I don't know," rather than to misclassify.

In this chapter we present an algorithm that reliably and probably usefully learns the concept class $\mathcal{C}^{\text{poly}}$.

## 5.2 How to learn: sketch

The original definition of pac learning has many desirable features. Not least among them is that efficient algorithms for pac learning a number of interesting concept classes are now known. Of course, we do not always know a priori that the concept we want to learn is going to be in 3CNF or 7DNF or some other given class. We would like to have an algorithm that can pac learn regardless of what class the target concept is drawn from. More precisely, we would like to have an algorithm that could pac learn the class of all functions that can be represented by a polynomial-size boolean formula (or, similarly, a polynomial-size boolean circuit).

Unfortunately, this goal is unlikely to be attainable. As Valiant explains [52], assuming that one-way functions exist (an assumption which we feel is likely to be correct), the class of polynomial-size boolean formula is not pac learnable.

Thus we are driven to look for some way of learning arbitrary boolean formulas. Our solution is to learn in a hierarchical manner. First we pac learn some important subconcepts of the target concept, and then we pac learn the final concept as a function of these subconcepts.

To be more precise, our method is as follows: We learn our first subconcept knowing that it must be some simple boolean function of the instance attributes. We learn each following subconcept knowing that it must be some some simple boolean function of the instance attributes *and previously learned subconcepts.* Ultimately we learn the original target concept as some simple boolean function of the instance attributes and all of the previously learned subconcepts.

Consider, for instance, the concept of one's *dependents*, as defined by the IRS.[1]

$$\begin{aligned} \textit{dependent} = (> \frac{1}{2}\text{SupportFromMe}) \wedge \neg\text{FiledJointReturn} \\ \wedge[(\text{Income} < 1900 \wedge (\text{MyChild} \vee \text{MyParent}) \\ \vee(\text{MyChild} \wedge (\text{Age} < 19 \vee \text{IsInCollege}))]. \end{aligned} \tag{5.1}$$

One can readily imagine such a complicated definition being too hard to learn from examples. On the other hand, if we first teach some simple subconcepts, such as "MyChild $\vee$ MyParent," and "Age<19 $\vee$ IsInCollege," and next teach some harder subconcepts as functions of those, and then finally the *dependent* concept as a function of all previously learned subconcepts, then the learning task becomes easier.

Moreover, because we break the target concept into very simple subconcepts, we can develop a learning protocol that has one very nice feature absent from ordinary pac learning—our learner *knows* when it is confused. (Formally, we achieve reliable and probably useful learning.) Continuing with the above example, we probably do not need to force our learner/taxpayer to learn the concept *dependent* perfectly. It is acceptable if the learner is unable to correctly classify certain unusual, very low probability instances such as, say, the case of "your underage great-great-great-granddaughter when all intervening generations are deceased." The probability of such an instance occurring is extremely low. Nevertheless, it would be desirable, if one ever did encounter such an "ever so great" grandchild, to be able to say, "I don't know if she is an instance of a dependent," rather than to misclassify her.

Our learner can, if desired, do precisely that—output a short fast program taking instances as its input and having the three outputs, "Yes" (*dependent*), "No" (not a *dependent*), and "I don't know." This program is guaranteed to be correct whenever it gives a "Yes" or "No" classification, and moreover, with probability at least $1 - \delta$

[1]What follows is, in fact, a great oversimplification of the IRS definition.

it says "I don't know" about at most a fraction $\epsilon$ of all people. In short, it meets our definition of reliable and probably useful learning.

### 5.2.1 Notation

Before showing how to break our target concept, $t$, into pieces, we must first specify the problem more precisely. For convenience' sake only, we assume that $t$ is represented as a straight-line program: Let the inputs to $t$ be $x_1, \ldots, x_n$, and call the output $y_l$. The $i$-th line of the program for $t$, for $1 \leq i \leq l$ is of the form:

$$y_i = z_{i,1} \circ z_{i,2} \tag{5.2}$$

where $\circ$ is one of the two boolean operators $\vee$ and $\wedge$, and every $z_{i,k}$ is either a literal, or else $y_j$ or $\bar{y}_j$ for some previously computed $y_j$ (i.e., $j < i$).[2] We say $l$ is the *size* of such a straight line program.

We will use EB($h$) (standing for "Easy Boolean") to denote the set of all boolean formula that are the conjunction or disjunction of two literals chosen from a set of at most $h$. We will often write simply EB when the value of $h$ is clear from context.

Figure 5-1 shows a straight line program for the *dependent* concept defined in equation (5.1) above. The expression for every $y_i$ comes from the class EB(13). (More precisely, the expression for $y_i$ comes from EB($6 + i$).)

### 5.2.2 An easy but trivial way to learn

As a first attempt to develop a protocol for learning our arbitrary $t$ piece by piece, we might try the following: Have the teacher supply not only examples, but also the pieces—the $y_i$. In particular, let the $y_i$ be rearranged in some arbitrary order,

[2]Note that straight line programs are equivalent to circuits, with lines being equivalent to the gates of the circuit, topologically sorted.

Input variables: FiledJointReturn, $>\frac{1}{2}$SupportFromMe, MyChild, MyParent, Age<19, IsInCollege, Income<1900.
Output: *dependent* $= y_7$.

$$
\begin{aligned}
y_1 &= (> \frac{1}{2}\text{SupportFromMe}) \wedge \neg\text{FiledJointReturn} \\
y_2 &= \text{MyChild} \vee \text{MyParent} \\
y_3 &= \text{Age} < 19 \vee \text{IsInCollege} \\
y_4 &= \text{Income} < 1900 \wedge y_2 \\
y_5 &= \text{MyChild} \wedge y_3 \\
y_6 &= y_4 \vee y_5 \\
y_7 &= y_1 \wedge y_6
\end{aligned}
$$

**Figure 5-1**: A straight line program for *dependent*

$y_{j_1}, \ldots, y_{j_l}$. Now, each time the learner requests an example, he gets more than just a labeled example drawn according to $\mathbf{P}$. The learner receives $x_1 \cdots x_n \# y_{j_1}, \ldots, y_{j_l}$ (and its label). Given all this help, it turns out to be easy to learn. It is not hard for the learning algorithm to determine which of the other variables a given variable depends on.

This solution is not very satisfying, however, since it requires that the learner receive a large amount of "extra help" with each and every example. In essence, it would mean that every time our learner was given an example while learning *dependent*, he would have to be told whether it was a child in college, whether it was a relative, and so on. Our approach is to first teach the learner about $y_1$ for a while, assume the learner has learned $y_1$, then move on to $y_2$, never to return to $y_1$, and so on, $y_i$ by $y_i$.

### 5.2.3 High level view of our solution

The learning proceeds as follows: As in regular pac learning, there is one fixed probability distribution, **P**, on examples throughout; the teacher is *not* allowed to help the student by altering the probability distribution.

There are $l$ rounds. The teacher moves from round $i$ to round $i+1$ when the learner tells him to do so. In round $i$, the learner is going to learn an approximation to $y_i$.

When our learner requests an example during round $i$, the teacher gives the learner a pair, $(x_1 \cdots x_n, s)$, where $x_1 \cdots x_n$ is drawn according to **P**, and $s$ tells whether $x_1 \cdots x_n$ is a positive or negative instance *of* $y_i$. In other words, in round $i$, $s$ gives the truth value of $y_i(x_1 \cdots x_n)$ (rather than the truth value of $t(x_1 \cdots x_n)$).

During each round $i$, the learner tries to $(\epsilon', \delta')$-learn $y_i$ where $\epsilon' = \epsilon/p_1(n)$ and $\delta' = \delta/p_2(n)$ and where $p_1$ and $p_2$ are polynomials to be specified below. This learning task at first glance appears to be extremely simple, because $y_i$ must be a simple conjunction or disjunction of $x_1 \cdots x_n$ and $y_1, \ldots, y_{i-1}$ (and perhaps their negations). The catch is that while the learner gets the true values for $x_1 \cdots x_n$ he only gets his computed values for $y_1, \ldots, y_{i-1}$.

For instance, it might be that the true formula for $y_3$ is $y_1 \wedge y_2$. However, the values of $y_1$ and $y_2$ are not inputs to the learning algorithm. Suppose the learner has pac learned formulas $\hat{y_1}$ and $\hat{y_2}$ for $y_1$ and $y_2$. It may well be that the learner calls EXAMPLES and gets back a particular $x_1 \cdots x_n$ and the information that $y_3(x_1 \cdots x_n)$ is true, but both $\hat{y}_1(x_1 \cdots x_n)$ and $\hat{y}_2(x_1 \cdots x_n)$ evaluate to false when $y_1(x_1 \cdots x_n)$ and $y_2(x_1 \cdots x_n)$ are both true.

Our job is to show how to do this learning in such a manner that at the end, when we have a representation for $y_l$ in terms of all the $x_i$ and $y_i$, and we substitute the $x_i$ back in for the $y_i$, the final expression, $y_l(x_1 \cdots x_n)$ $\epsilon$-approximates the target concept with probability $1 - \delta$.

In fact, as we said above, we do something stronger. Our learner not merely pac learns, but reliably and probably usefully learns.

### A key technique

The key technique we use is to have the learner learn and maintain a list of all possible candidates for a given $y_i$. For each subconcept $y_i$ we explicitly maintain the "most specific" list of the version space representation [37].

The reason the learner can maintain this list is that the set of all the possible candidates for any particular $y_i$ is of polynomial size. Recall that the target function $t$ is specified by a straight line program. Let $K$ be the total number of possible distinct lines, $z_{i,1} \circ z_{i,2}$.

$$K = 8\binom{n+l}{2}.$$

The important thing to notice is that $K$ is polynomial in $n$ and $l$, the size of (the representation of) the target concept. The exact value of $K$ comes about because each $y_i$ is in the class EB, but the only thing special about EB is that it is of polynomial size. Our technique will work equally well using any polynomial-size concept class.

We exploit this technique by designing an algorithm with three fundamental parts:

1. In round $i$ we get various examples of $y_i$. We say that an example, $(x_1 \cdots x_n, s)$, is "good" if for every previously learned $y_j$, $1 \leq j < i$, all the formulas in the list for $y_j$ take on the same truth value on $x_1 \cdots x_n$. Since one of the formulas in the list for $y_j$ is the correct one, in every good example all the $y_j$'s are computed correctly. We begin by filtering our examples to obtain a set of good examples.

2. Given good examples, we can be certain of the values of the $y_j$, so we can proceed to learn $y_i$ as a function of the attributes $x_1 \cdots x_n, y_1, \ldots, y_{i-1}$.

3. Finally, we need something to specify the algorithm that we output at the end of round $l$.

## 5.3 Detailed specification of our learning protocol

We assume to begin with that the learner is given $l$, the length of the straight line program, at the beginning of the learning protocol. This assumption makes the presentation simpler and clearer. We show later that the learner need not be given $l$.

### 5.3.1 Learning $y_i$

During each round $i$, the learner simply needs to learn $y_i$ as a function of the input literals and previous $y_j$ and $\bar{y}_j$. Moreover, the formula for $y_i$ is in the class EB. There are at most $K$ candidates for the the formula for $y_i$.

In Chapter 2 we saw how to ($\epsilon'$, $\delta'$)-pac learn concept $y_i$ that is one of at most $K$ concepts by using the static consistent algorithm.

The learner chooses

$$m \geq \frac{1}{\epsilon'}\left(\ln(K) + \ln\left(\frac{1}{\delta'}\right)\right) \tag{5.3}$$

and obtains $m$ labeled instances from EXAMPLES. The learner then checks the (at most $K$) candidates for the formula for $y_i$ one by one until one is found that is consistent with all $m$ examples, and outputs that candidate. Of course, our proof of the correctness of the static consistent algorithm depends on the assumption that the instances output by EXAMPLES are not missing any bits.

We could use the static consistent algorithm for our $\epsilon', \delta'$ learning of $y_1$, since we do always get the correct values of the instance attributes when we request a labeled example in round 1 of our learning. In fact, we use something much like that algorithm, except that we check all the possible formulas for $y_1$, and "output" the set of all the

formulas that are consistent with all $m$ examples. (Learning $y_1$ is merely an internal subroutine used in the the first stage of a multi-stage learning protocol; we don't really output anything at this point.)

The idea of using a set here is that if the set is guaranteed to contain the correct function, then when all functions in the set agree, we know we have the correct value of $y_1$. Otherwise we know we "don't know" $y_1$.

DEFINITION. Let $F = \{f_1, \ldots, f_s\}$ be a set of boolean formulas, each of the same number of variables, say $n$. We say $F$ *is coherent on* $x_1 \cdots x_n$ if $f_1(x_1 \cdots x_n) = f_2(x_1 \cdots x_n) = \cdots = f_s(x_1 \cdots x_n)$.

Let $F_1 = \{f_{1,1}, f_{1,2}, \ldots f_{1,i_1}\}$ be the set of formulas we learned for $y_1$. Notice that for an arbitrary example, $x_1 \cdots x_n$, if $F_1$ is coherent on $x_1 \cdots x_n$, then the common value of the formulas must be the true value for $y_1(x_1 \cdots x_n)$. The reason is that we know the true formula for $y_1$ is contained in $F_1$.

Thus, in order to learn an arbitrary $y_i$, we are led to use Procedure CSL, which is specified in Figure 5-2. The key thing to notice in Procedure CSL is that once an example has been "filtered," (in Subroutine GetSample) then—for that example—we know not only the values of the instance attributes $x_1 \cdots x_n$, but also the values of $y_1, \ldots y_{i-1}$.

### 5.3.2 Learning the target concept

Procedure CSL does indeed give us a way to learn our target concept $t$ once we calculate appropriate values for $\epsilon'$ and $\delta'$.

**Theorem 5.1** *Fix a target concept $t$ that is a function of $n$ variables, and some straight line program for $t$ of length $l$. Let $K$ be the number of possibilities for the $l$-th line of the program. Take any $0 < \epsilon, \delta \leq 1$. Let $\epsilon' = \epsilon/lK$. Let $\delta' = \delta/l$. Call* CSL(1), CSL(2), ..., CSL($l$). *Then,*

**Procedure** ConsistentSetLearner (hereinafter CSL)

Inputs: $i$; $F_1, \ldots, F_{i-1}$ (previously learned formula sets for $y_1, \ldots, y_{i-1}$);
$n, \epsilon'$, and $\delta'$.
Output: $F_i$, a set of formulas for $y_i$, or "Fail."

Pick $m$ according to equation (5.3).
Call GetSample$(m, i)$.
If GetSample did not return "Fail"
  then set $F_i := \{f \in EB(i) \mid f$ is consistent with all $m$ filtered examples$\}$.
  else output "Fail."

Subroutine GetSample$(m, i)$
Repeat until either $m$ "filtered" examples are obtained, or $2m$ attempts are made:
    Obtain an example, $x = x_1 \cdots x_n$, by calling EXAMPLES.
    For $j := 1$ to $i - 1$ do
        If $F_j$ is coherent for $x_1 \cdots x_n, y_1, \ldots, y_{j-1}$
          then label the common value $y_j$
          else break the For $j$ loop.
    If every $F_j$, $1 \leq j \leq i - 1$, was coherent
        then consider $x$ to be "filtered," and save it.
        else discard $x$.
If $m$ filtered examples were obtained
    then return those $m$ filtered examples
    else return "Fail".

**Figure 5-2**: Procedure CSL

1. *with probability at least* $1 - \delta$,

   - *no call ever returns "Fail," and,*
   - *with probability at least* $1 - \epsilon$ *every* $F_i$ *is coherent on a randomly drawn instance,* $x_1 \cdots x_n$ *and,*

2. *if every* $F_i$ *is coherent on* $x_1, \ldots, x_n$, *then* $y_l(x_1 \cdots x_n)$ *(making the appropriate substitutions for intermediate* $y_i$*) correctly classifies* $x_1 \cdots x_n$.

*Proof* Our instance space, $X$, is $\{0,1\}^n$. Let $X_i(F_1, \ldots, F_{i-1})$ be all instances $x$ such that all of $F_1, \ldots, F_{i-1}$ are coherent on $x$. The set $X_i(F_1, \ldots, F_{i-1})$ contains exactly those instances that will be successfully filtered by CSL($i$) (instead of being discarded by CSL($i$)).

For $f \in F_i$, define

$$err_i(f) = \{x \in X_i(F_1, \ldots, F_{i-1}) \mid f(x) \neq y_i(x)\}.$$

The set $err_i(f)$ consists of those instances that might be seen as training instances for $y_i$ that $f$ misclassifies. Let $\mathbf{P}$ be the probability distribution on the instance space.

Let us call $f \in F_i$ *good* if

$$\mathbf{P}(err_i(f)) \leq \frac{\epsilon}{lK}.$$

We say that $F_i$ is good if every $f \in F_i$ is good, and otherwise we say that $F_i$ is bad. Finally, let us say that CSL($i$) *wins* if the output is a good $F_i$, and neither a bad $F_i$ nor "Fail."

In the case of CSL(1) we can never fail. It follows from the discussion in Section 2.6 of Chapter 2 that

$$\Pr[F_1 \text{ is bad}] \leq \delta/l. \tag{5.4}$$

Hence the probability that CSL(1) wins is at least $1 - \delta/l$.

For CSL(2), there are two ways we could end with "Fail." The first is that we could fail because $F_1$ is bad. We ignore this possibility, because what we ultimately want to calculate is

$$\Pr[F_1 \text{ and } F_2 \text{ win}] = \Pr[F_1 \text{ wins}] \Pr[F_2 \text{ wins} \mid F_1 \text{ wins}].$$

Thus we need only worry about whether CSL(2) outputs "Fail" in the case where $F_1$ is good.

If $F_1$ is good, then

$$\Pr[\text{one fixed } f \in F_1 \text{ disagrees with } y_1 \text{ on a random } x] \leq \frac{\epsilon}{lK} \tag{5.5}$$

$$\Pr[\text{any } f \in F_1 \text{ disagrees with } y_1 \text{ on a random } x] \leq \frac{\epsilon}{l} \tag{5.6}$$

since there are at most $K$ formulas in $F_1$. Hence, the probability (given that $F_1$ is good) that $F_1$ is not coherent for a random instance is at most $1 - \epsilon/l$. We can use Hoeffding's Inequality (Lemma 4.1 above) to show that the probability of that happening on more than $m$ out of $2m$ trials is at most

$$\begin{aligned} e^{-(1-\frac{2\epsilon}{l})m} &= \left(\frac{lk}{\delta}\right)^{-(1-\frac{2\epsilon}{l})\frac{lk}{\epsilon}} \\ &= l/\delta^{-\omega(1)} \end{aligned}$$

which for our purposes is vanishingly small.

Thus the probability that CSL(2) fails given that $F_1$ is good is vanishingly small, so to determine whether CSL(2) wins we need only determine whether the output $F_2$ is good given that the sample determining $F_2$ was filtered by a good $F_1$. This probability is at least $1 - \delta/l$, and the reason is almost the same as the same as the reason that the probability that $F_1$ is good is at least $1 - \delta/l$. If we do not fail when we call CSL(2), then the probability that $F_2$ is bad is again at most $\delta/l$. The difference is that now the probability distribution on instances is not $\mathbf{P}$ but the conditional probability distribution $\mathbf{P}'$ defined by $\mathbf{P}'(x) = \mathbf{P}(x \mid x \in X_1)$. However, since $\mathbf{P}'(x) \geq \mathbf{P}(x)$ for

all $x$ in the sample used to pick the formulas in $F_2$, this difference can only cause the probability that $F_2$ is good to be larger.

The argument for an arbitrary call to CSL($i$) is similar to that for CSL(2) and it shows that

$$\Pr[\text{CSL}(i) \text{ outputs "Fail"} \mid \text{CSL}(1),\ldots,\text{CSL}(i-1) \text{ win}] \leq e^{-(1-\frac{2\epsilon}{l})m}.$$

and that assuming CSL($i$) does not output "Fail"

$$\Pr[F_i \text{ is bad} \mid \text{CSL}(1),\ldots,\text{CSL}(i-1) \text{ win}] \leq \frac{\delta}{l}. \tag{5.7}$$

Thus we have that

$$\Pr[\text{CSL}(i) \text{ wins} \mid \text{CSL}(1),\ldots,\text{CSL}(i-1) \text{ win}] \geq 1 - \left(\frac{\delta}{l} + \left(\frac{l}{\delta}\right)^{-\omega(1)}\right). \tag{5.8}$$

Now we compute the probability that all calls to CSL win using what is sometimes called the law of successive conditioning. Let $\mathcal{E}_i$ be the event that CSL($i$) wins.

$$\begin{aligned}
\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \cdots \wedge \mathcal{E}_l] &= \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdots \Pr[\mathcal{E}_l \mid \mathcal{E}_1 \wedge \cdots \wedge \mathcal{E}_{l-1}] \\
&= \left(1 - \left(\frac{\delta}{l} + \left(\frac{l}{\delta}\right)^{-\omega(1)}\right)\right)^l \\
&> 1 - \delta.
\end{aligned} \tag{5.9}$$

Thus the probability that all calls to CSL win is at least $1-\delta$. By definition, if all calls to CSL win, no call outputs "Fail," and all the $F_i$ are good. Now we must show that if every $F_i$ is good, then the total probability weight assigned by $\mathbf{P}$ to instances $x$ for which some $F_i$ is not coherent is at most $\epsilon$.

We showed above in inequality (5.6) that the probability weight $\mathbf{P}$ assigns to instances on for which $F_1$ is not coherent (given that $F_1$ is good) is at most $\epsilon/l$. Similar reasoning shows that the probability weight $\mathbf{P}$ assigns to instances in $X_i(F_1,\ldots,F_{i-1})$ for which $F_i$ is not coherent is at most $\epsilon/l$. Hence the total probability weight assigned to instances for which some $F_i$ is not coherent is at most $l(\epsilon/l) = \epsilon$ as desired. □

Algorithm Reliable Learner

Inputs: $n, \epsilon, \delta$, $l$, teacher for gate by gate learning of unknown concept $c$.
Output: Program to classify instances.

For $i := 1$ to $l$ do
    Call CSL($i$).
    If the call fails
      then halt and output a program that classifies all examples "I don't know."
      else save the set $F_i$ output by CSL($i$).
Output the program that classifies an instance $x = x_1 \cdots x_n$ as follows:

    For $j := 1$ to $l$ do
      If $F_j$ is coherent for $x_1 \cdots x_n, y_1, \ldots, y_{j-1}$
        then label the common value of the functions in $F_j$ as $y_j$
        else halt and output "I don't know."
    Output $y_l$.

**Figure 5-3**: Algorithm Reliable Learner

**Corollary 5.1** *There is an algorithm for reliably and probably usefully learning any concept represented by a polynomial-size circuit gate by gate.*

*Proof* We exhibit the algorithm in Figure 5-3. It is immediate from Theorem 5.1 that this algorithm has the desired properties. □

Thus we have a simple program that with probability $1 - \delta$ classifies most examples correctly, and "knows," because it found some incoherent $F_i$, when it is given one of the rare examples it can't classify.

On the other hand, if we really want to simply pac learn, and output a boolean circuit, we can do that as well by doing the following: Pick any formula for $y_1$ from $F_1$ to obtain a gate computing $y_1$. Use this gate wherever $y_1$ is called for later. In the same manner, pick any formula from $F_2$ to be a gate for computing $y_2$. Continue in this

fashion until we finally have a circuit for $y_l$ taking only variables $x_1 \cdots x_n$ as inputs.

**Corollary 5.2** *If we run the process described in Theorem 5.1, and then convert to a boolean circuit as described above, this process pac learns.*

### 5.3.3 Removing the circuit size as an input

In this section we show that we can still use Procedure CSL as a subroutine for learning reliably and probably usefully even if the length of the target concept is not known by the learner.

The idea of the method is that when we know $l$ we spread out our tolerance $\epsilon$ by "using up" $\epsilon/l$ per line for a total of $\epsilon$, and similarly for $\delta$. Now, not knowing $l$, we "use up" $\frac{6\epsilon}{\pi^2 i^2}$ for line $i$, for a total of

$$\begin{aligned} \sum_{i=1}^{l} \frac{6\epsilon}{\pi^2 i^2} &< \frac{6\epsilon}{\pi^2} \sum_{i=1}^{\infty} \frac{1}{i^2} \\ &= \epsilon, \end{aligned}$$

and similarly for $\delta$. Of course that is merely the intuition; we now proceed to the proof.

**Theorem 5.2** *Call CSL(1), CSL(2), ..., CSL(l) using the values* $\epsilon' = \frac{6\epsilon}{K\pi^2 i^2}$ *and* $\delta' = \frac{6\delta}{\pi^2 i^2}$ *in the call to CSL(i). Then*

1. *with probability at least* $1 - \delta$,
   - *no call ever returns "Fail," and,*
   - *with probability at least* $1 - \epsilon$ *every* $F_i$ *is coherent on a randomly drawn instance,* $x_1 \cdots x_n$ *and,*
2. *if every* $F_i$ *is coherent on* $x_1, \ldots, x_n$, *then* $y_l(x_1 \cdots x_n)$ *(making the appropriate substitutions for intermediate* $y_i$*) correctly classifies* $x_1 \cdots x_n$.

*Proof* We sill simply give the changes that need to be made to the proof of Theorem 5.1. We now define $f \in F_i$ to be good only if

$$\sum_{x \in err_i(f)} \mathbf{P}(x) \leq \frac{6\epsilon}{K\pi^2 i^2}.$$

We continue to say that $F_i$ is good if every $f \in F_i$ is good, and that CSL($i$) wins if its output is a good $F_i$ as opposed to either a bad $F_i$ or "Fail".

Equation (5.4) becomes

$$\Pr[F_1 \text{ is bad}] \leq \frac{6\delta}{\pi^2}. \tag{5.10}$$

Its generalization, equation (5.7), now states that assuming that the call to CSL($i$) does not output "Fail" we have

$$\Pr[F_i \text{ is bad} \mid \text{CSL}(1),\ldots,\text{CSL}(i-1) \text{ won}] \leq \frac{6\delta}{\pi^2 i^2}. \tag{5.11}$$

The chance that a call to CSL($i$) outputs "Fail" if all the $F_j$ for $1 \leq j \leq i-1$ are good is still exponentially vanishing. Thus equation (5.8) becomes

$$\Pr[\text{CSL}(i) \text{ wins} \mid \text{CSL}(1),\ldots,\text{CSL}(i-1) \text{ won}] \geq 1 - \left(\frac{6\delta}{\pi^2 i^2} + \left(\frac{l}{\delta}\right)^{-\omega(1)}\right). \tag{5.12}$$

If we again let $\mathcal{E}_i$ be the event that CSL($i$) wins, equation (5.9) now becomes

$$\begin{aligned}
\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \cdots \wedge \mathcal{E}_l] &= \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdots \Pr[\mathcal{E}_l \mid \mathcal{E}_1 \wedge \cdots \wedge \mathcal{E}_{l-1}] \\
&= \prod_{i=1}^{l} \left(1 - \left(\frac{6\delta}{\pi^2 i^2} + \left(\frac{l}{\delta}\right)^{-\omega(1)}\right)\right)^l \\
&> 1 - \sum_{i=1}^{l} \frac{6\delta}{\pi^2 i^2} \\
&> 1 - \delta.
\end{aligned}$$

The argument for the parameter $\epsilon$ is similar to the argument for the parameter $\delta$.

□

## 5.4 Noise

In the previous section we showed how to reliably and probably usefully learn any concept that can be represented by a polynomial-size circuit. However, the algorithm given there depends on the fact that the examples given by the teacher were completely free of noise. In this section we show how that algorithm can be modified to tolerate noise in the data, albeit with a slight degradation in the learning.

From perfect data we saw that we could achieve

$$\text{Guaranteed reliable and } \Pr[\text{useful}] \geq 1 - \delta.$$

In the presence of noise we achieve

$$\Pr[\text{Reliable and useful}] \geq 1 - \delta.$$

Note that for the purpose of applications, this degradation is not as bad as it seems at first. All our algorithms have sample complexity and running time polynomial in $\log(1/\delta)$, so we can afford to make $\delta$ very small.

Our results for noise meet the limit for classification noise suggested by Chapter 4. We do not know how to do as well as the limit suggested there for malicious noise. To be specific, we show how to tolerate classification noise with a noise rate of up to $1/2$, or how to tolerate malicious noise with a noise rate of up to $\epsilon/4lK$.

### 5.4.1 Classification noise

We begin by showing how to modify Procedure CSL to tolerate noise coming from $\text{RMC}_\nu$, and then go on to argue that this modification in fact tolerates noise coming from $\text{MMC}_\nu$. Call this new procedure CSL2.

In the case of classification noise, the learner receives one additional input, $\nu_b$, a bound on the noise rate. The learner knows that all the examples come from $\text{RMC}_\nu$

for some fixed $0 \leq \nu \leq \nu_b$. Of course the learner is now allowed time polynomial in $1(1-2\nu_b)$ in addition to the other parameters.

There are two differences between Procedure CSL and Procedure CSL2. The first is that the value of $m$ is changed to

$$m = \frac{8}{\epsilon'^2(1-2\nu_b)^2} \ln\left(\frac{3K}{\delta'}\right). \tag{5.13}$$

The second difference between CSL and CSL2 is the way that the formulas for $F_i$ are chosen once a sample of $m$ "good" examples have been obtained. In Procedure CSL2 for every candidate formula $f$ for $y_i$ we calculate $d_f$, the number of disagreements between $f$ and the labels of the teacher for the sample. Clearly $0 \leq d_f \leq m$. In Procedure CSL we simply set $F_i$ to be the set of all $f$ such that $d_f = 0$. We knew there would be at least one such $f$, because in the noise-free case, $d_{y_i} = 0$. Now that there is classification noise, however, there may not be any $f$ for which $d_f$ is equal to 0.

Now instead we find a formula $f_*$ such that $d_{f_*}$ is minimal, and set

$$F_i = \{f \mid d_f \leq d_{f_*} + ms/4\} \tag{5.14}$$

where

$$s = \epsilon'(1-2\nu_b). \tag{5.15}$$

We formally specify CSL2 in Figure 5-4.

**Theorem 5.3** *Let $\epsilon' = \epsilon/lK$. Let $\delta' = \delta/l$. Call* CSL2(1), CSL2(2), ..., CSL2($l$). *Assume that the examples come from $RMC_\nu$ for some $\nu \leq \nu_b$, where $\nu_b < \frac{1}{2}$. Then, with probability at least $1-\delta$,*

- *no call ever returns "Fail," and,*
- *with probability at least $1-\epsilon$ every $F_i$ is coherent on a randomly drawn instance, $x_1 \cdots x_n$ and,*

**Procedure** CSL2

Inputs: $i$; $F_1, \ldots, F_{i-1}$ (previously learned formula sets); $\nu_b, n, \epsilon'$, and $\delta'$.
Output: $F_i$, a set of formulas for $y_i$, or "Fail."

Pick $m$ according to equation (5.13).
Call GetSample$(m, i)$. (Specified above in Figure 5-2.)
If GetSample returned "Fail"
  then output "Fail"
  else for every $f \in EB(i)$ compute mistake number $d_f$.
    Set $dmin := \min\{d_f\}$.
    for every $f \in EB(i)$
      If $d_f < dmin + m\epsilon'(1 - 2\nu_b)/4$
        then put $f$ in $F_i$.

**Figure 5-4**: Procedure CSL2

- *if every $F_i$ is coherent on $x_1, \ldots, x_n$, then $y_l(x_1, \ldots, x_n)$ (making the appropriate substitutions for intermediate $y_i$) correctly classifies $x_1 \cdots x_n$.*

*Proof* We basically want to show that the proof of Theorem 5.1 still goes through. We proved two key facts about each $F_i$ in that proof. The first was that the true formula for $y_i$ was guaranteed to be in $F_i$. The second was that with high probability each $F_i$ was good, where good was defined to mean that every formula in $F_i$ agreed with $y_i$ on at least probability weight $1 - \epsilon/l$ of the instances. Let us say that an $F_i$ is "very good" if it has both of those two properties. We now show that with high probability every $F_i$ is very good. (Because of the noise, we can no longer hope to *guarantee* that the true formula for $y_i$ is in $F_i$.) In particular we show that

$$\Pr[y_i \in F_i \text{ and } F_i \text{ is good} \mid F_1, \ldots F_{i-1} \text{ are all very good}] \geq 1 - \delta/l. \qquad (5.16)$$

Assume now that $F_1, F_2, \ldots, F_{i-1}$ are all very good. That means that we have the full set of correct attribute bits in the $m$ examples we use to determine $F_i$ (although,

of course, some of the labels may be wrong).

We noted in the proof of Theorem 4.3 of Chapter 4 that $s$ specified in equation (5.15) is the expected gap in disagreement rate between the true formula for $y_i$ and any formula that is not an $\epsilon'$-approximation of the true formula. Thus if we were to make $F_i$ the set of all $f$ such that $d_f \leq \mathrm{E}[d_{y_i} + ms/2]$, then we would have that with high probability, $F_i$ is very good. Now $\mathrm{E}[d_{y_i}] = \nu m$, but unfortunately we do not know the value $\nu$.

Nevertheless, to prove that equation (5.16) holds it suffices to show that the sum of the probability of each of the following three events is at most $\delta/l$.

1. No formula has a number of disagreements less than $\mathrm{E}[d_{y_i}] - ms/4$.

2. For the true formula $y_i$ we have $d_{y_i} \leq \mathrm{E}[d_{y_i}] + ms/4$.

3. Every $\epsilon'$-bad formula has at least $\mathrm{E}[d_{y_i}] + 3ms/4$ disagreements with the sample.

If both of the first two events occur, then we have that $d_{y_i} \leq d_{f_*} + ms/2$, so $y_i$ is placed in $F_i$.

If both event 2 occurs, then we have that

$$d_{y_i} \leq \mathrm{E}[d_{y_i}] + ms/4,$$

and thus that

$$d_{f_*} \leq \mathrm{E}[d_{y_i}] + ms/4.$$

Together with the occurrence of event 3 this guarantees that no $\epsilon'$-bad formula is put in $F_i$.

For every formula $f$ we have the $\mathrm{E}[d_f] \geq \mathrm{E}[d_{y_i}]$. (The argument here is somewhat similar to the proof of Theorem 4.3.) Thus we get from Hoeffding's Inequality that

$$\begin{aligned} \Pr[\text{fixed } f \text{ has } d_f \leq \mathrm{E}[d_{y_i}] - ms/4] &\leq e^{-2(s/4)^2 m} \\ &\leq \frac{\delta}{3Kl.} \end{aligned}$$

Thus the probability of event 1, that any $f$ has a $d_f$ that is too small, is at most $\delta/3l$.

A similar argument shows that the probability of event 2 is at most $\delta/3Kl$, and that the probability of event 3 is at most $\delta/3l$.

Given this, the rest of the proof is completely analogous to the proof of Theorem 5.1. The only difference is that in this case the true formula for $y_i$ is in $F_i$ only with high probability, not with certainty. □

**Theorem 5.4** *In fact, Procedure CSL2 has the performance specified in Theorem 5.3 even if the examples come from $MMC_\nu$.*

*Proof* We want to show that the condition specified by inequality (5.16) still holds in this case. The only difference between $\text{MMC}_\nu$ and $\text{RMC}_\nu$ is that there may be some examples that were labeled incorrectly by $\text{RMC}_\nu$ that now the devil decides to label correctly. We show that in spite of any such change of label, if $y_i$ was in $F_i$ when the examples came from $\text{RMC}_\nu$ then it still is in $F_i$, and that no $\epsilon'$-bad formula that was not placed in $F_i$ before is now.

The only way for $y_i$ not to be in $F_i$ is if some other formula has $ms/2$ fewer disagreements with the sample than $y_i$ has. Changing incorrect labels to correct labels cannot affect whether that is true. Changing an incorrect label to a correct label may or may not decrease the number of disagreements for any formula but $y_i$ by 1; such a change must decrease $d_{y_i}$ by 1.

The way we showed that no $\epsilon'$-bad formula was placed in $F_i$ was by showing that every such formula had a number of disagreements at least $ms/2$ greater than $d_{y_i}$. Again changing incorrect labels to correct labels cannot eliminate such a gap. □

### 5.4.2 Malicious noise

The strategy for learning in the presence of a small amount of malicious noise is similar to the strategy for learning in the presence of classification noise. We only consider

**Procedure** CSL3

Inputs: $i$; $F_1, \ldots, F_{i-1}$ (previously learned formula sets); $n, \epsilon'$, and $\delta'$.
Output: $F_i$, a set of formulas for $y_i$, or "Fail."

Pick $m_{\mathrm{MAL}}$ according to equation (5.17).
Call GetSample($m_{\mathrm{mal}}$). (Specified in Figure 5-2.)
If GetSample returned "Fail"
  then output "Fail"
  else for every $f \in EB(i)$
    compute the mistake rate for $f$
    If it is less than $\frac{\epsilon}{2lK}$ then put $f$ in $F_i$.

**Figure 5-5**: Procedure CSL3

malicious noise rates $\nu \leq \epsilon/4lK$. Given a sample from $\mathrm{MAL}_\nu$, the expected mistake rate of any must be at most $(\epsilon/lK) - \nu$. Therefore, there must be a gap $g$ in expected mistake rates between the true rule and any $(\epsilon/lk)$-bad rule of at least

$$\begin{aligned} g &= \frac{\epsilon}{lK} - 2\nu \\ &\geq \frac{\epsilon}{2lK} \end{aligned}$$

since $\nu < \epsilon/4lK$.

We exploit this gap by getting a sample of size

$$m_{\mathrm{MAL}} = \frac{8l^2K^2}{\epsilon^2}\left(\ln K + \ln\left(\frac{3l}{\delta}\right)\right), \tag{5.17}$$

and putting into our set any rule with a disagreement rate less than $\nu + g/2$. The precise algorithm, CSL3, is specified in Figure 5-5.

**Theorem 5.5** *Let* $\epsilon' = \epsilon/lK$. *Let* $\delta' = \delta/l$. *Call* CSL3(1), CSL3(2), ..., CSL3($l$). *Assume that the examples come from* $MAL_\nu$ *for some* $\nu \leq \epsilon/4lK$. *Then, with probability at least* $1 - \delta$,

- *no call ever returns "Fail," and,*

- *with probability at least* $1-\epsilon$ *every* $F_i$ *is coherent on a randomly drawn instance,* $x_1 \cdots x_n$ *and,*

- *if every* $F_i$ *is coherent on* $x_1, \ldots, x_n$*, then* $y_l(x_1, \ldots, x_n)$ *(making the appropriate substitutions for intermediate* $y_i$*) correctly classifies* $x_1 \cdots x_n$*.*

*Proof sketch:* The proof is similar to the proof of Theorem 5.3. We still say the $F_i$ is very good if $y_i \in F_i$ and if the set of instances for which any $f \in F_i$ disagrees with $y_i$ has probability weight at most $\epsilon/l$.

Assume now that $F_1, F_2, \ldots, F_{i-1}$ are all very good, and that in CSL3($i$) the call to GetSample did not return "Fail". In order to show that $F_i$ is very good we need the following to events to occur.

1. The mistake rate of $y_i$ must be less than $\epsilon/2lK$.

2. Every $(\epsilon/lK)$-bad formula $f \in EB(i)$ must have a mistake rate of at least $\epsilon/2lK$.

Since the samples come from the malicious error oracle, we cannot simply say that $d_{y_i}$, the mistake number of $y_i$, is the sum of Bernoulli trials. Nevertheless, the worst thing that the malicious error oracle can do to $d_{y_i}$ is add one to it every time it affects the examples. Thus, $d_{y_i}$ has expectation no greater than the sum of $m_{\mathrm{MAL}}$ Bernoulli random variables each with probability $\epsilon/4lK$ of being 1. As usual we use Hoeffding's Inequality to show that the probability of getting a value as high as $2(\epsilon/2lK)m_{\mathrm{MAL}}$ is at most $(\delta/2K)$.

The argument for the second event, an $(\epsilon/lK)$-bad formula having too low a mistake rate, is similar. □

## 5.5 Summary and conclusions

In this chapter, we have shown how to learn complicated concepts by breaking them into subconcepts. The key idea we used was maintaining a list of all possible candidates (the "version spaces") for the correct subconcept, instead of simply picking some one candidate. For the purposes of this chapter, we were concerned with the class EB, but our method is applicable to any polynomial-size class. We expect that this particular method will prove to have other applications.

We believe this general approach is the philosophically sound way to do inductive inference, since what distinguishes induction from deduction is that in induction one can never be completely certain that one has learned correctly. (Kugel [28] contains an interesting discussion of this point.) It is always possible that one will see a counterexample to one's current favorite theory. This idea of maintaining a list of all the candidates for the correct "answer" has recently born fruit elsewhere in the field of inductive inference as well, in a new model of recursion theoretic inductive inference [47], and in a method for inference of simple assignment automata [48].

Another contribution of this chapter has been to introduce the notion of learning that is reliable and probably useful, and to give a learning procedure that achieves such learning.

In fact, our learning procedure is in one sense not merely reliable, but even better; because it has maintained candidate sets for all subconcepts, it need not simply output "I don't know," on difficult instances. It has maintained enough information to be able to know which subconcept is causing it to output "I don't know." Thus, in a learning environment where it is appropriate to do so, our learning procedure can go back and request more help from the teacher on that particular subconcept.

# Chapter 6

# A different model of learning

## 6.1 Introduction

In this chapter we present a new model for the the process of "inductive inference"—the process of drawing inferences from data. Angluin and Smith [6] provide an excellent introduction and overview of previous work in the field. Our work is distinguished by the following features:

- Our inference procedure begins with an a priori probability associated with each possible theory, and updates these probabilities in a Bayesian manner as evidence is gathered.
- Our inference procedure has two primitive actions available to it for gathering evidence, each of which has a cost (in terms of time taken):
  1. Using a theory to predict the result of a particular experiment.
  2. Running an experiment.
- Our inference procedure attempts to maximize the expected "rate of return", for example, in terms of the total probability of theories eliminated per unit time.

Osherstraub, Stob, and Weinstein [38] have examined the issue of Bayesianism within a standard model of inductive infrerence. Their work is rather different from the approach we take in this chapter, however, because their definition of efficient computation is effectively any recursive function.

Our approach addresses the following three issues, which we feel are not always well handled by previous models.

**(1) Induction is fundamentally different from deduction.** Much previous work has tried to cast induction into the same mold as deduction: given some data (premises) to infer the correct theory (conclusion). This approach is philosophically wrong, since experimental data can only eliminate theories, not prove them. (See Feyerabend [13] and Kugel [28].) For similar reasons, we feel it is better to study inference procedures which represent the *set* of remaining theories (and perhaps their probabilities), rather than inference procedures which are constrained to return a *single* answer.

**(2) The difficulty of making predictions is overemphasized.** Much of the previous theoretical work in this area has been recursion-theoretic in nature, and the richness of the results obtained has been in large part due to the richness of the theories allowed; allowing partial recursive functions as theories makes inference very difficult. The resulting theory probably overemphasizes this recursion-theoretic aspect, compared to the ordinary practice of science. In this paper, all theories will be total (they predict a result for every experiment), and we assume that the cost of making such a prediction from a theory is a fixed constant $c$ (time units), independent of the theory or the proposed experiment. This is obviously an oversimplification, but serves our purposes well.

**(3) Experiments take time, and should be carefully chosen.** Much of the previous work on inductive inference has assumed that the data (i.e., the list of all possible experimental results) is presented to the learner in some order (cf. [14, 8]).

However, the rate of progress in science clearly depends on which

experiments are run next. (Consider experimental particle physics today.) Part of doing science well is choosing the right experiments to do.

A good scientist must decide how to allocate his time most effectively—should he next run some experiment (if so, which one?), or should he work with one of the more promising theories, computing what it would predict for some experiment (if so, which theory and which experiment?). These "natural" questions are not particularly well handled by previous models of the inductive inference problem, but our model will allow us to answer such questions. Our results also shed some interesting light on related questions, such as when to run "crucial" experiments that distinguish between competing hypotheses.

Our model can perhaps be viewed as well as a contribution to the theory of subjective probability [16], which has traditionally had a problem with the fact that subjective probabilities can change as a result of "pure thinking." Various proposals, such as "evolving probabilities" [17] have been proposed, but these do not deal with the "thinking" aspect in a clean way.

## 6.2 Subjective probabilities

Throughout this chapter, when we speak of the probability that our scientist assigns to some event, we are speaking of the scientist's *subjective probability.* Our view of subjective probability has been heavily influenced by I. J. Good's article, "Kinds of Probability [16]."

The reason for dealing with subjective probability is that "true" or frequentist or "physical probability" may not be available. Our scientist does not necessarily have some oracle available to provide him with "true probabilities;" he must take some action based on his current set of beliefs.

## 6.3 The Model

### 6.3.1 Basic Notation and Assumptions

We assume the existence of some scientific domain of interest, defined by an (infinite) set of possible experiments. Performing the $j$-th experiment yields a datum $\chi_j$; in this paper we assume for convenience that $\chi_j \in \{0,1\}$. We make the simplistic assumption that doing an experiment always takes precisely $d$ units of time (independent of which experiment is performed).

We assume that there are an infinite (but enumerable) set of theories available about the given domain; we denote them as $\varphi_0, \varphi_1, \ldots$. Each theory is understood to be a total function from $\mathbf{N}$ into $\{0,1\}$; the value $\varphi_i(j) = \varphi_{ij}$ is the "prediction" theory $\varphi_i$ makes about the result of experiment $j$. We assume there exists a *correct* theory, $\varphi_r$, such that $(\forall j)\varphi_{rj} = \chi_j$. We make the simplistic assumption that computing $\varphi_{ij}$ from $i$ and $j$ always takes precisely $c$ units of time (independent of $i$ and $j$).

We assume that other operations, such as planning, take no time.

Our scientist begins with two kinds of initial or a priori subjective probabilities:

- The a priori probability that $\varphi_{ij} = 1$, for any $i$ and $j$. We assume that $\Pr(\varphi_{ij} = 0) = \Pr(\varphi_{ij} = 1) = \frac{1}{2}$ a priori, for all $i$ and $j$; the scientist has no reason to expect his theory to predict one way or the other, until he actually does the computation.
- The a priori probability $p_i$ that theory $\varphi_i = \varphi_r$, (i.e. that $\varphi_i$ is correct). We assume that the $p_i$'s are computable, that $(\forall i)p_i > 0$ (all theories are possible at first), and that that $p_0 \geq p_1 \geq \ldots$.

### 6.3.2 The Scientist Makes Progress

Our scientist begins in a state of total ignorance, and proceeds to enlighten himself by taking steps consisting of either doing an experiment (determining some $\chi_j$) or making

a prediction (computing some $\varphi_{ij}$). The scientist may choose which experiments and predictions he wishes to do or not to do, and can do these in any order (predictions may precede or follow corresponding experiments, for example).

We need notation to denote the scientist's state of knowledge at time $t$ (after $t$ steps have been taken).

- Let "$\perp$" denote "unknown".
- Let $\varphi_{ij}^t \in \{0, 1, \perp\}$ denote the scientist's knowledge of $\varphi_{ij}$ at time $t$.
- Let $\chi_j^t \in \{0, 1, \perp\}$ denote the scientist's knowledge of $\chi_j$ at time $t$.

If at time $t$ both $\varphi_{ij}^t = \varphi_{ij}$ and $\chi_j^t = \chi_j$ (i.e. both are known at time $t$), then there are two possibilities. Either $\varphi_{ij} \neq \chi_j$, in which case theory $\varphi_i$ is *refuted*, or $\varphi_{ij} = \chi_j$, in which case theory $\varphi_i$ is (to some extent) *confirmed.*

### 6.3.3 How Long Will Science Take?

Obviously, after a finite number of steps, our scientist will be able to refute only a finite number of theories, so at no point will he be able claim that he has discovered the complete "truth".

More realistically, he may ask "How long will it be before I have eliminated all theories with higher a priori probability than the correct theory?" The answer here depends on the set of a priori probabilities. A realistic "non-informative prior" attempts to have $p_i$ decrease to zero as slowly as possible; for example we might have $p_i = C \cdot (i \ln(i) \ln \ln(i) ...)^{-1}$, where $C$ is a normalizing constant and only the positive terms in the series of logarithms are included [45].

Note that at least one step is required to eliminate a theory, so that the expected number of steps required to eliminate all theories with higher a priori probability than

the true one is at least equal to the expected number of such theories, i.e.,

$$\sum_{r=0}^{\infty} r \cdot p_r \ ,$$

which is infinite. This result holds for many similar probability distributions which do not go to zero too quickly.

In fact, for a typical set of initial probabilities, our scientist expects to have an infinite amount of work to do before the true theory is even considered!

For this reason, among others, we will concentrate on the rate at which the scientist can refute false theories, rather than on the expected time taken before the scientist would assert that, on the basis of the evidence available to him, $\varphi_r$ is the best available theory.

### 6.3.4 How the Scientist Updates His Knowledge

To model the evolution of the scientist's knowledge more carefully, we show how his subjective probabilities associated with the various theories change as a result of the steps he has taken, using Bayes' Rule.

What happens to the probabilities maintained by the scientist after step $t$ is performed? Let $p_i^t$ denote the probabilities after step $t$ (here $p_i^0 = p_i$). We consider the effect of step $t$ on the probability that theory $\varphi_i$ is correct. That is, we look at how $p_i^{t-1}$ is updated to become $p_i^t$.

The process of updating these probabilities according to the result of the last step, can be performed by executing the following operations in order:

1. For all $i$,

   - Set $p_i^t$ to 0 if $\varphi_i$ has just been refuted.
   - Set $p_i^t$ to $2 * p_i^{t-1}$ if $\varphi_i$ has just been confirmed.

- Otherwise set $p_i^t$ to $p_i^{t-1}$.

2. Normalize the $p_i^t$'s so that they add up to 1.

The above procedure follows directly from Bayes' Rule, since it is judged a priori to be equally likely for a prediction to be a 0 or a 1.

We note that if the scientist just sits and "thinks" about an experiment (i.e., he just computes the predictions of various theories for this experiment), his subjective probability that $\Pr(\chi_j^t = 0)$ will *evolve*, since

$$\Pr(\chi_j^t = 0) = \sum_{\varphi_{ij}^t = 0} p_i^t + \frac{1}{2} \sum_{\varphi_{ij}^t = \perp} p_i^t.$$

It would also not be unreasonable to treat this probability as an interval, since one knows the upper and lower limits that it could evolve to.

### 6.3.5 An Example

Consider Table 6.1, which illustrates a portion of a particular scientist's knowledge at some point in time. (Here unknown values are shown as blanks, and only a portion of the actual infinite table is shown.)

The second row of the table shows which experiments he has run. Here he knows only $\chi_0 \ldots \chi_4$. The second column gives his current probabilities $p_i^t$.

The second part shows what predictions he has made. Each row of this table corresponds to one theory. Theories which have been refuted have current probability zero and are not shown here; *it is convenient from here on to assume that $\varphi_0$ is the most probable theory, $\varphi_1$ is the next most probable theory, and so on.* In this example, the scientist has found out what his most probable theory predicts for experiments 0–5, and so on.

Running experiment 5 next has the potential of refuting $\varphi_0$. (It will either refute $\varphi_0$ or $\varphi_3$.) Making the prediction $\varphi_{1,5}$ can not (immediately) refute $\varphi_1$, but would

| | | | j | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| $i$ | $p_i$ | $\chi_j \rightarrow$ | 0 | 1 | 1 | 0 | 0 | | |
| 0 | 0.60 | | 0 | 1 | 1 | 0 | 0 | 1 | |
| 1 | 0.10 | | 0 | 1 | 1 | 0 | | | |
| 2 | 0.05 | | 0 | 1 | 1 | | | | |
| 3 | 0.04 | $\varphi_{ij} \rightarrow$ | 0 | | | | | 0 | |
| 4 | 0.03 | | | 1 | 1 | | | | |
| 5 | 0.02 | | 0 | | | | | | |
| 6 | 0.01 | | | | | | | | |

**Table 6.1**: Partial View of Scientist's State of Knowledge

affect the scientist's estimate of the likelihood that $\chi_5 = 0$. With the current state of knowledge, the scientist would estimate that

$$\Pr(\chi_5 = 0) = 0.04 + \frac{1}{2}(1 - 0.60 - 0.04) = 0.22.$$

Note, however, that $\Pr(\varphi_{1,5} = 0)$ remains $1/2$, independent of anything else, until it is computed.

## 6.4 Our Inference Procedures

The approach taken by a scientist will depend upon the relative costs of making predictions versus doing experiments, his initial probabilities for the theories, and exactly how he wishes to "optimize" his rate of progress.

### 6.4.1 General Assumptions

At each step, the scientist must decide what to do next. Although this choice is, and always remains, a choice among an infinite number of alternatives, it is reasonable to restrict this to a finite set by adopting the following rules:

- When running or predicting the result of an experiment which has neither been previously run nor had predictions made for it, without loss of generality choose the least-numbered such experiment available.

- When making a prediction for a theory for which *no* previous predictions have been made, choose the most probable such theory (in the case of ties, choose the least-numbered such theory).

### 6.4.2 Optimization Criteria

The scientist will choose what actions to take according to some optimization criteria. For example, he may wish to:

1. Maximize the expected total probability currently associated with theories which are refuted by the action chosen.

2. Minimize the entropy $-\sum_{i=0}^{\infty} p_i^t \log(p_i^t)$ of his assignment of probabilities to theories.

3. Maximize the probability assigned to the theory he currently believes to be the most likely.

4. Maximize the highest probability assigned to any theory.

5. Minimize the expected total probability assigned to *incorrect theories.*

More generally, he may wish to maximize his "rate of progress" by dividing his progress (measured by the change in one of the above criteria) by the time taken by the action chosen.

In this paper we will discuss all of the above optimization criteria; some very briefly, and some at length. In the remainder of this section we discuss the general form that all our inference procedures take, regardless of the particular optimization criterion they use.

### 6.4.3 Menus of Options

We propose that the scientist organize his strategy as a "greedy" strategy of the following form:

- He organizes his decision at each step into a finite number of options. Each such option is a *program* specifying a sequence of predictions and/or experiments to run, which terminates with probability 1.

- At a given step, for each available option, the scientist computes the expected "rate of return" of that option, defined as the expected total gain of that option (where gain is measured by some optimization criterion) divided by the expected cost of that option.

- The scientist then chooses to execute an option having highest expected rate of return, breaking ties arbitrarily.

The reason for introducing the notion of an "option", rather than just concentrating on the elementary possibilities for a given step, is that certain steps have *no* expected rate of return in and of themselves. For example, making a prediction when the corresponding experiment has not yet been run has zero expected rate of return, as does

running an experiment when no prediction regarding that experiment has yet been made.

From now on, we let $q_i^t$ denote $1 - p_i^t$. We also observe that if our set of probabilities satisfies $p_0 \geq p_1 \geq \ldots$ then it also satisfies $p_0 q_0 \geq p_1 q_1 \geq \ldots$, since $p_0$ is no further from $\frac{1}{2}$ than $p_1$ is and $\frac{1}{2} \geq p_1 \geq p_2 \geq \ldots$.

## 6.5 Inference procedure 1: Maximizing the weight of refuted theories

We begin by studying an inference procedure which tries to refute wrong theories as quickly as possible. Specifically, the scientist will choose an action which maximizes the quotient of the expected total probability of theories eliminated by that action, divided by the cost of that action. The reason for this choice is its simplicity, and the ease with which the scientist can implement such a strategy. Furthermore, if our a priori probability happens to be one of the ones for which infinite expected time is required simply to eliminate all wrong theories (see section 6.3.3.), then this measure probably makes the most sense.

### 6.5.1 A Simple Menu of Options

In this subsection and the following subsection, we will spell out a particular menu of options and analyze our scientist's strategy when he uses this menu and the "maximizing the weight of refuted theories" optimization criterion. In later sections we will analyze our scientist's strategy when he uses the same menu but different optimization criteria.

We first consider the following two options, each of which will always have non-zero expected rate of return:

- *Prediction/Experiment Pair:* Make a prediction $\varphi_{0j}$ for the least $j$ for which no predictions yet exist, and then run the corresponding experiment. Here, as usual, $\varphi_0$ denotes the theory which is currently most probable. The exepected "reward" for this action is $p_0^t$ times the probability that $\varphi_0$ will, in fact, be refuted. Theory $\varphi_0$ will never be refuted if it is the true theory, and will be refuted with probability $1/2$ otherwise; therefore the probability tht $\varphi_0$ will be refuted is $q_0^t/2$. Our expected rate of return is thus

$$\frac{p_0^t q_0^t}{2(c+d)}.$$

  We are not compelled to restrict the prediction/experiment pairs to using the most probable theory, but do so because it is convenient to limit our options, and also because the expected return from other theories will not be as good.

- *Prediction:* Compute a prediction $\varphi_{ij}$, given that the corresponding experiment determining $\chi_j$ has already been run. Thus, the expected rate of return for this prediction is

$$\frac{p_i^t q_i^t}{2c}.$$

  Here again it is clear that we should choose the least $i$ possible, so as to maximize the rate of return.

If we stick to options in this simple menu, then the opportunity to make a prediction only arises after the simple prediction/experiment pair has already been run for that experiment.

### 6.5.2 An Expanded Menu of Options

An expanded menu can be obtained by adding the following two options to the simple menu:

- *Simple Experiment:* Run experiment $j$, given that at least one prediction has been made for this experiment. The expected rate of return is

$$\frac{p_0^t q_0^t}{2d},$$

since the probability that "truth" differs from $\varphi_0$ is $q_0^t/2$, and (as argued below), in this case we must have only the prediction $\varphi_{0j}$.

- *Crucial Two-Way Experiment:* Determine the least $j$ such that the two most probable theories make differing predictions for $\chi_j$. Then run experiment $j$. To calculate the expected reward, we must consider three cases. (1) If $\varphi_0$ is the true theory, then we will refute $\varphi_1$, for a reward of $p_1^t$. The probability that $\varphi_0$ is the true theory is $p_0^t$, so this case contributes $p_0^t p_1^t$ to the total expected reward. (2) Similarly, the case where $\varphi_1$ is the true theory contributes $p_0^t p_1^t$ to the total final reward. (3) If neither $\varphi_0$ nor $\varphi_1$ is the true theory, then it is equally likely that $\varphi_0$ or $\varphi_1$ will be refuted. Since this case has probability $1 - p_0^t - p_1^t$, its contribution to the expected reward is $(1 - p_0^t - p_1^t)(p_0^t + p_1^t)/2$. Thus the expected reward is

$$\begin{aligned} 2p_0^t p_1^t + \frac{(1 - p_0^t - p_1^t)(p_0^t + p_1^t)}{2} &= \frac{p_0^t q_0^t + p_1^t q_1^t + 2p_0^t p_1^t}{2} \\ &= \frac{p_0^t + p_1^t - (p_0^t - p_1^t)^2}{2} \end{aligned}$$

Note that the expected cost of *finding* a crucial experiment is exactly $4c$, since if we pick a $j$ and compute $\varphi_{0j}$ and $\varphi_{1j}$, we have a $1/2$ chance of finding $j$ to be crucial.[1] The expected rate of return is

$$\frac{p_0^t + p_1^t - (p_0^t - p_1^t)^2}{2(4c + d)}. \tag{6.1}$$

[1]Note also, that there is no special reason to restrict ourselves to crucial two-way experiments. We could also run crucial $n$-way experiments, where we find the least $j$ such that the $n$ most probable theories split as evenly as possible (in terms of probability weight). Now the expected cost of finding such a $j$ increases from $4c$ to $(2^n + 2^{n-1} - 2)c$.

We note that in the expanded menu, the only way an opportunity can arise to run a simple experiment is by having the search for a crucial experiment generate predictions for the first two theories, without running the corresponding experiment since the predictions were identical. This is the only way we can obtain a situation where predictions have been made for experiments that haven't been run. Furthermore, additional predictions won't be made for this experiment until after this experiment has been run. Since the crucial experiment will eliminate one of the top two theories, we will be left in a situation where (after renumbering of theories as usual) there is a $j$ for which we know $\varphi_{0j}$ but have not yet run experiment $j$.

We claim that, using either the simple or expanded menu, the *relative* order of two theories will not change, except when a theory is refuted, if an optimal greedy strategy is used. This follows since it is always preferable to work with the more probable theories, given a particular option, and this work will tend to enhance the probability of that theory if it is not refuted.

Having given our menu of options, we can now make one simple definition. When we speak of *checking* or *testing* $\varphi_i$, we are talking about either doing a prediction/experiment pair involving $\varphi_i$ or doing simple experiment $j$ for some $j$ for which $\varphi_i$ has already made a prediction. In short, testing $\varphi_i$ means to take some action that could potentially refute $\varphi_i$.

### 6.5.3 Behavior of this Inference Procedure

#### For the Simple Menu

For the simple menu, clearly we begin with a prediction/experiment pair. After that, the scientist will oscillate between further testing of his best theory (using prediction/experiment pairs), and testing of his other theories (using predictions).

The ratio $c/(c+d)$ will affect the relative amount of time spent on prediction/

experiment pairs. We will typically see all theories down to some probability threshold (depending on $c$, $d$, and $p_0$) fully checked out against existing experimental data, before proceeding with the next prediction/experiment pair.

### For the Expanded Menu

If it is more expensive to perform an experiment than to compute a theory's prediction, then our scientist will at least want to consider whether he should get his experimental data from crucial experiments rather than from prediction/experiment pairs.

Let's consider whether at the beginning of time, the scientist is better off running a prediction/experiment pair, or running a crucial two-way experiment. The crucial experiment will have a higher expected rate of return if

$$\frac{p_0 + p_1 - (p_0 - p_1)^2}{2(4c + d)} > \frac{p_0 q_0}{2(c + d)} \tag{6.2}$$

or

$$\frac{d}{c} \geq \frac{3p_0 q_0}{p_1(2p_0 + q_1)} - 1.$$

It is sufficient for equation 6.2 to hold if

$$\frac{c + d}{3c} > \frac{p_0 q_0}{p_1 q_1}.$$

We see that for any ratio $d/c$, it is possible to have a crucial experiment be advantageous over a prediction/experiment pair; consider what happens when $p_0 = p_1 = 1/2$.

*No matter how cheap experiments get, relative to the cost of making predictions, it is possible to find a probability distribution where it is advantageous to find an experiment which will be crucial, before running any experiments.*

Thus in general, it may pay to use the expanded menu, for any values of $d$ and $c$.

## 6.6 Inference procedure 2: A minimum entropy approach

The entropy of a probability distribution $P$,

$$H(P) = \sum_{i=1}^{\infty} -p_i \log_2 p_i \tag{6.3}$$

is considered to be a good measure of the information contained in that probability distribution. Maximizing entropy corresponds to maximizing uncertainty; minimizing entropy corresponds to minimizing uncertainty. Thus a reasonable optimization criterion for our scientist would be minimizing the entropy of the a posteriori probability distribution.

Unfortunately, for some probability distributions, the entropy will be infinite. Consider, for instance, the previously mentioned distribution due to Rissanen [45],

$$p_i = C \cdot (i \ln(i) \ln \ln(i) ...)^{-1} , \tag{6.4}$$

where $C$ is a normalizing constant and only the positive terms in the series of logarithms are included. Wyner [56] shows that the entropy series, equation (6.3), converges only if the series $\sum_{i=1}^{\infty} p_i \log i$ is convergent, but this series is clearly diverges for the distribution given in equation (6.4).

However, any particular experiment or prediction made by our scientist only causes him to alter a finite number of his a posteriori probabilities for theories, excluding the effect of renormalizing. It happens, as we shall see below, that even with renormalization, the expected *change* in entropy for any action from the expanded menu is finite.

The above discussion leads us to a precise description of the optimization criterion for our second inference procedure. The scientist chooses an action which maximizes the quotient of the expected decrease in the entropy of the probability distribution resulting from that action, divided by the cost of that action.

### 6.6.1 Behavior of this Inference Procedure

We need to calculate the expected change in entropy for each of our action in our (expanded) menu. To begin with, we calculate the change in entropy caused by refuting $\varphi_0$. (The case for $\varphi_i$ is similar, but the notation is simpler for $i = 0$.) Let $P^{t_0}$ be the initial probability distribution , and let $\Delta(H(P))$ denote the change in entropy that occurs when $\varphi_0$ is refuted. Then

$$\begin{aligned}
\Delta(H(P)) &= -\sum_{i=1}^{\infty} \frac{p_i}{1-p_0} \log\left(\frac{p_i}{1-p_0}\right) + \sum_{i=0}^{\infty} p_i \log p_i \\
&= \frac{1}{1-p_0}\left[(1-p_0)\log(1-p_0) - \sum_{i=1}^{\infty} p_i \log p_i\right] + \sum_{i=0}^{\infty} p_i \log p_i \\
&= \log(1-p_0) + \frac{1}{1-p_0}\left(p_0 \log p_0 + \sum_{i=0}^{\infty} p_i \log p_i\right) + \sum_{i=0}^{\infty} p_i \log p_i \\
&= \log(1-p_0) + \frac{p_0 \log p_0}{1-p_0} + \frac{p_0}{1-p_0} H(P^{t_0}). \qquad (6.5)
\end{aligned}$$

Now we calculate the change in entropy that occurs if $\varphi_0$ is instead partially confirmed. In the case we have

$$\begin{aligned}
\Delta(H(P)) &= -\frac{2p_0}{1+p_0} \log\left(\frac{2p_0}{1+p_0}\right) - \sum_{i=1}^{\infty} \frac{p_i}{1+p_0} \log\left(\frac{p_i}{1+p_0}\right) + \sum_{i=0}^{\infty} p_i \log p_i \\
&= \frac{1}{1+p_0}\left[-2p_0 \log\left(\frac{2p_0}{1+p_0}\right) + p_0 \log p_0 - \sum_{i=0}^{\infty} p_i \log p_i\right] + \sum_{i=0}^{\infty} p_i \log p_i \\
&= \log(1+p_0) - \frac{p_0}{1+p_0}\left(2 + \log p_0 + H(P^{t_0})\right). \qquad (6.6)
\end{aligned}$$

Now we are ready to compute the expected change in entropy for each item in our expanded menu.

- For computing the prediction $\varphi_{;j}$ (assuming that $\chi_j$ is already known), we get

$$\mathrm{E}\left[\Delta(H(P))\right] = -p_i + .5(1-p_i)\log(1-p_i) + .5(1+p_i)\log(1+p_i). \qquad (6.7)$$

  Equation (6.7) comes from taking $(1-p_i)/2$ (the probability that $\varphi_i$ is refuted) times the quantity specified by equation (6.5) plus $(1+p_i)/2$ (the probability that $\varphi_i$ is confirmed) times the quantity specified by equation (6.6).

- For running a two way experiment between $\varphi_0$ and $\varphi_1$ we get

$$\mathrm{E}[\Delta(H(P))] = -p_0-p_1+.5(1+p_0-p_1)\log(1+p_0-p_1)+.5(1-p_0+p_1)\log(1-p_0+p_1). \tag{6.8}$$

- In fact, in general, for running $\chi_j$ where the total probability weight of theories which predict that $\chi_j$ will be zero is $r_0$ and the total probability weight of theories which predict that $\chi_j$ will be one is $r_1$ we get

$$\mathrm{E}[\Delta(H(P))] = -r_0-r_1+.5(1+r_0-r_1)\log(1+r_0-r_1)+.5(1-r_0+r_1)\log(1-r_0+r_1). \tag{6.9}$$

Consider the probability distribution, $R$, that has only two outcomes, one with probability $r_0 + .5(1 - r_0 - r_1)$, the other with probability $r_1 + .5(1 - r_0 - r_1)$. We can rewrite equation (6.9) in terms of the entropy of $R$,

$$\mathrm{E}[\Delta(H(P))] = -r_0 - r_1 + H(R). \tag{6.10}$$

Equations (6.7) and (6.8) can be rewritten in a similar manner (since really they're just special cases of equation (6.9)).

In fact, the calculations for this entropy driven inference procedure and the previous, "Kill wrong theories" driven procedure yield very similar results. Equation (6.10) and equation (6.1) could both be written as

$$\text{PROGRESS} = k(r_0 + r_1 - \text{penalty}(|r_0 - r_1|). \tag{6.11}$$

(The difference in signs between equation (6.10) and equation (6.11) arises because in equation (6.10) we're trying to *minimize* entropy, so our progress is negative, and our penalty is positive.)

Let $\delta = |r_0 - r_1|$. For the entropy approach, $k = 1$ in equation (6.11), and $\text{penalty}(\delta) = H(.5+\delta/2, .5-\delta/2)$. (In terms of $r_0$ and $r_1$ that probability distribution is

$r_0 + u/2, r_1 + u/2$, where $u = 1 - r_0 - r_1$ is the undecided probability weight—the total probability weight of those theories $i$ such that $\varphi_i(j) = \perp$.) For the kill wrong theories approach of the previous section, $k = 1/2$ in equation (6.11), and penalty$(\delta) = \delta^2$.

As one might expect given this strong similarity between the two optimization criteria, the inference procedures behave in a roughly similar manner.

## 6.7 Inference procedure 3: Making the best theory good

Our scientist might decide that he would like to at all times have a theory that's "pretty good." There are several approaches he might take.

In the extreme, he might simply decide that his goal would be to always increase the a posteriori probability assigned to the current best theory. Such a cynical strategy turns out to be impossible. No actions lead to an *expected* increase in the probability assigned to the best theory. If we check the best theory with any kind of action, then with probability $p_0 + .5(1 - p_0)$ it is confirmed, and its probability goes up to $2p_0/(1 + p_0)$. However, with probability $1 - p_0$ it is refuted and its probability goes to zero. Thus its expected probability after any action is $[(p_0 + 1)/2] \cdot 2p_0/(1 + p_0) = p_0$. If we check other theories, they may be either refuted, which would increase the probability assigned to $\varphi_0$, or confirmed, which would decrease the probability assigned to $\varphi_0$, and it again works out that the expected value of the a posteriori probability weight assigned to $\varphi_0$ is $p_0$.

Since our scientist cannot steadily increase the probability assigned to the best theory, he might settle for a strategy which always keeps the current best theory best. To accomplish this goal, the scientist should never test $\varphi_0$ against any theory. He should simply test the other theories, making sure to stop testing $\varphi_i$ as soon as $p_i \geq .5p_0$

(otherwise $\varphi_i$ might replace $\varphi_0$ as best). This procedure is obviously uninteresting.

There is, however, at least one interesting way for the scientist to always have a "pretty good" best theory. The scientist chooses an action to maximize the quotient of the expected value of the probability weight assigned to the best theory not yet refuted after that action, and the cost of that action.

### 6.7.1 Behavior of this Inference Procedure

The first thing we do is calculate the expected value of the weight assigned to the best theory for each action from the expanded menu.

- If we test $\varphi_0$ (with any kind of action), then with probability $p_0 + .5(1 - p_0)$ it will be confirmed, and the probability weight for the best theory will become $2p_0/(1 + p_0)$. With probability $.5(1 - p_0)$, $\varphi_0$ will be refuted, and the probability weight for the best theory will become $p_1/(1 - p_0)$. The expected value of the probability weight for the best theory is therefore $p_0 + p_1/2$.

- If $p_i \leq .5p_0$ (so if even we test and confirm $\varphi_i$ it will still have a lower a posteriori probability weight than $\varphi_0$), then testing $\varphi_i$ does not lead to an increase in the expected value of the probability weight of the best theory.

- If $p_i > .5p_0$, and we test $\varphi_i$, then the expected value of the probability weight of the best theory after the test is $p_i + p_0/2$.

  Note however, that this situation is of no practical importance. If $\chi_j$ is known and both $\varphi_{0j}$ and $\varphi_{ij}$ are unknown, then it will be more profitable to compute $\varphi_{0j}$ than to compute $\varphi_{ij}$. Consider now the case where there is some $j$ such that $\chi_j^t = \varphi_{0j}^t$ but $\varphi_{ij}^t = \perp$. Whichever theory is now numbered zero began with an initial probability weight greater than or equal to the initial probability weight of

of the theory now numbered $i$. Moreover, since at time $t$ $\varphi_0$ has been confirmed more than $\varphi_i$, it must be that $p_0^t \geq 2p_i^t$.

- If we run a crucial experiment for the two best theories, then the expected value of the probability weight of the best theory is $p_0 + p_1$.[2]

Having listed the payoffs for each action, we can now give the payoff/cost ratios for the actions we might take:

- A simple pair with the best theory: $(p_0 + .5p_1)/(c + d)$.
- Prediction for $\varphi_{0j}$ if $\chi_j$ known: $(p_0 + .5p_1)/d$
- Simple experiment $\chi_j$ where $\varphi_{0j}$ is known: $(p_0 + .5p_1)/d$.
- Crucial two way experiment: $(p_0 + p_1)/(4c + d)$.
- We might consider running a two way experiment when we have some leftover predictions (say from an earlier two way experiment) for one of the two theories. If we have $k$ such predictions, then the expected cost decreases from $d + 4c$ to $d + (3 - \sum_{i=1}^{k-1} 2^{-i})c$.

All our scientist needs to do is pick the maximum reward/cost action from the above list, but we'll make a few qualitative observations here: If there is a $j$ for which $\chi_j$ is known but $\varphi_{0j}$ is not, then it's always best to compute $\varphi_{0j}$. It's better to do a crucial experiment instead of a simple pair if $d/c > 6p_0 + 2p_1$; otherwise it is better to do the simple pair.

[2]In this case there we gain nothing by running a crucial experiment for the best $n$ theories for $n > 2$.

## 6.8 An optimality result

There are a number of ways one might measure the efficiency of our inference procedures. Here we consider the question, "How efficiently do these procedures eliminate wrong theories?" This measure seems especially appropriate since all of these inference procedures have the qualitative behavior that early on they are busy refuting lots of wrong theories. It turns out that all our procedures do this refuting of wrong theories well; we will show that all of our procedures perform within a constant factor of the optimum.

We begin by calculating the best possible refutation rate.

### 6.8.1 The optimal refutation rate

Assume that the right theory has index at least $r$. Define $f(c, d, r)$ to be the expected cost of refuting $\varphi_0, \varphi_1, \ldots, \varphi_{r-1}$.

**Theorem:** *For any inference procedure,* $f(c, d, r) \geq 2cr + d\Theta(\log r)$.

Proof: To refute $\varphi_i$ we must keep on computing values of $\varphi_{ij}$ until we get one where $\varphi_{ij} = 0$ and $\chi_j = 1$ or vice versa. Given that $\varphi_i$ is not the right theory, we expect we will on average have to try two $\varphi_{ij}$ until we get one that is refuted by $\chi$. Hence our expected computation cost for eliminating $r$ theories must be at least $2cr$.

Now for the cost of doing experiments. Since for wrong theories the $\varphi_{ij}$ are all independent, we might as well reuse the same experimental $\chi_j$'s in refuting each $\varphi_i$. However, we have $r$ such $\varphi_i$'s to refute. What is the expected maximum number of agreements between any $\varphi_i$ and $\chi$ over all $r$ $\varphi_i$'s? Equivalently, if we play a game where we toss a coin until we've seen a total of $r$ heads, what is the expected length of the longest consecutive run of tails? We will show that the answer is $\Theta(\log r)$.

More formally, let $X_i$ be the number of experiments required to refute (wrong) $\varphi_i$;

it is easy to check that $\Pr[X_i = j] = 2^{-j}$ for $j = 1, 2, \ldots$. Let $X = \max_{i=1}^{r} X_i$. We want to show $\mathrm{E}[X] = \Theta(\log r)$.

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{k=1}^{\infty} k \Pr[X = k] \\
&= \sum_{k=1}^{\infty} k(\Pr[X \geq k] - \Pr[X \geq k+1]) \\
&= \sum_{k=1}^{\infty} \Pr[\exists i : X_i \geq k] \\
&\leq \sum_{k=1}^{\lfloor \log r \rfloor} \Pr[\exists i : X_i \geq k] + \sum_{k=\lfloor \log r \rfloor}^{\infty} r 2^{-k+1} \\
&\leq \log r + 1. \qquad (6.12)
\end{aligned}$$

In the other direction we have

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{k=1}^{\infty} \Pr[\exists i : X_i \geq k] \\
&= \sum_{k=1}^{\infty} 1 - (1 - 2^{-k+1})^r \\
&\geq \sum_{k=1}^{\lfloor .5 \log r \rfloor} 1 - (1 - 2/\sqrt{r})^r \\
&\geq (1 - (1 - 2/\sqrt{r})^r) \frac{1}{2} \log r \\
&\sim (1 - e^{-2\sqrt{r}}) \frac{1}{2} \log r. \qquad (6.13)
\end{aligned}$$

### 6.8.2 How our procedures compare to the optimum

The three inference procedures we discussed in the preceding three sections all perform within a constant factor of the optimum in refuting wrong theories.

None of them ever actually does an experiment when there are known experimental values against which the best theory has not yet been tested. Thus, until the right theory has become $\varphi_0$, we never do any more experiments than the optimum theory refutation strategy.

We do sometimes perform more computations than the optimum theory refutation strategy. In particular, we sometimes perform "wasted" computations as part of a crucial two way experiment. In such an experiment we might compute $\varphi_{0j}$ and $\varphi_{1j}$ for some $j$ and find them to be equal. By the definition of a crucial experiment, we will refute one of those two theories before ever doing experiment $\chi_j$; hence one of those computations was "wasted." However, we only perform crucial experiments when we're going to do an experiment, and we only do $O(\log r)$ experiments, so we only miss the optimum of $2cr$ by $cO(\log r)$.

## 6.9 Conclusions for Chapter 6

We have introduced a new model for the process of inductive inference, which

1. is relatively simple, yet
2. captures a number of the qualitative characteristics of "real" science,
3. provides a crisp model for evolving or dynamic subjective probabilities, and
4. demonstrates that crucial experiments are of interest for *any* relative cost of experiments and making predictions.

# Chapter 7

# Final remarks

This thesis has primarily explored two formal models of learning—the new model introduced in this chapter, and the pac learning model in earlier chapters. The hope is that the presentation of two such different models in the same place highlights the features that any such model must have, and illustrates the contributions that computational learning theory can make to the study of learning in general.

Good models will on the should capture some qualitative features of real world problems, and yet pose problems simple enough that one can make progress in finding provably good algorithms.

# Bibliography


[1] Naoki Abe. Polynomial learnability of semilinear sets (extended abstract). In *Second Workshop on Computational Learning Theory*, Santa Cruz, Cal. 1989.

[2] Jonathan Amsterdam. *The Valiant Learning Model: Extensions and Assessment.* Master's thesis, MIT Department of Electrical Engineering and Computer Science, January 1988.

[3] Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, July 1982.

[4] Dana Angluin, William I. Gasarch, and Carl H. Smith. *Training Sequences.* Technical Report UMIACS-TR-87-37, University of Maryland Institute for Advanced Computer Studies, August 1987.

[5] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[6] Dana Angluin and Carl H. Smith. Inductive inference: theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[7] J. M. Barzdin and R. V. Frievald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

[8] Lenore Blum and Manuel Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28(2):125–155, June 1975.

[9] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 273–282, Berkeley, California, May 1986.

[10] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. *Learnability and the Vapnik-Chervonenkis Dimension.* Technical Report USCS-

CRL-87-20, U.C. Santa Cruz Computer Science Laboratory, November 1987. To appear in *J. ACM.*

[11] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, April 1987.

[12] H. S. M. Coxeter and W. O. J. Moser. *Generators and Relations for Discrete Groups.* Springer-Verlag, New York, third edition, 1972.

[13] P. K. Feyerabend. *Philosophical Papers: Realism, Rationalism, & Scientific Method.* Volume 1, Cambridge University Press, 1981.

[14] E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[15] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[16] I. J. Good. Kinds of probability. *Science*, 129(3347):443–447, February 1959.

[17] I. J. Good. The probabilistic explication of information, evidence, surprise, causality, explanation, and utility. In V. P. Godame and D. A. Sprott, editors, *Foundations of Statistical Inference*, pages 108–141, Holt, Reinhart, and Winston, 1971.

[18] Israel Grossman and Wilhelm Magnus. *Groups and Their Graphs.* Volume 14 of *New Mathematical Library*, Mathematical Association of America, Washington, 1964.

[19] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, 4th edition, 1960.

[20] David Haussler. Bias, version spaces and Valiant's learning framework. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 324–336, University of California, Irvine, June 1987.

[21] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. In *First Workshop on Computational Learning Theory*, pages 42–55, Morgan-Kaufmann, August 1988.

[22] David Haussler, Nick Littlestone, and Manfred K. Warmuth. Predicting $\{0,1\}$-functions on randomly drawn points. In *29th Annual Symposium on Foundations of Computer Science*, pages 100–109, IEEE, White Plains, NY, 1988. Tech. Report, U. C. Santa Cruz, to appear (longer version).

[23] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[24] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass. 1979.

[25] Michael Kearns and Ming Li. Learning in the presence of malicious errors. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, May 1988.

[26] Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. Recent results on boolean concept learning. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 337–352, University of California, Irvine, June 1987.

[27] Michael Kearns and Leslie G. Valiant. *Learning Boolean Formulae or Finite Automata is as Hard as Factoring.* Technical Report TR 14-88, Harvard University Aiken Computation Laboratory, 1988.

[28] Peter Kugel. Induction, pure and simple. *Information and Control*, 35:276–336, 1977.

[29] John Laird, Paul Rosenbloom, and Allen Newell. Chunking in Soar: the anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 1986.

[30] John Laird, Paul Rosenbloom, and Allen Newell. Towards chunking as a general learning mechanism. In *Proceedings AAAI-84*, pages 188–192, August 1984.

[31] Philip D. Laird. *Learning from Good and Bad Data. Kluwer international series in engineering and computer science*, Kluwer Academic Publishers, Boston, 1988.

[32] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[33] Nathan Linial, Yishay Mansour, and Ronald L. Rivest. Results on learnability and the Vapnik-Chervonenkis dimension. In *Proceedings of the Twenth-Ninth Annual Symposium on Foundations of Computer Science*, pages 120–129, October 1988.

[34] Nick Littlestone. Learning when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[35] Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial Group Theory: Presentation of Groups in Terms of Generators and Relations.* John Wiley & Sons, New York, 1966.

[36] G. Miller. The magic number seven, plus or minus two: some limits on our capacity for processing information. *Psychology Review*, 63:81–97, 1956.

[37] 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, Cambridge, Mass., August 1977.

[38] Daniel N. Osherson, Michael Stob, and Scott Weinstein. Mathematical learners pay a price for Bayesianism. 1986. (MIT Dept. of Brain and Cognitive Science).

[39] Daniel N. Osherson, Michael Stob, and Scott Weinstein. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists.* MIT Press, 1986.

[40] Judea Pearl. On the connection between the complexity and credibility of inferred models. *Journal of General Systems*, 4:255–264, 1978.

[41] Leonard Pitt and Leslie G. Valiant. *Computational Limitations on Learning from Examples.* Technical Report, Harvard University Aiken Computation Laboratory, July 1986.

[42] Leonard Pitt and Manfred K. Warmuth. The minimum DFA consistency problem cannot be approximated within any polynomial. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, Seattle, Washington, May 1989.

[43] David Pollard. *Convergence of Stochastic Processes.* Springer-Verlag, 1984.

[44] J. Ross Quinlan. The effect of noise on concept learning. In *Machine Learning, An Artificial Intelligence Approach (Volume II)*, chapter 6, pages 149–166, Morgan Kaufmann, 1986.

[45] Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

[46] Ronald L. Rivest. Personal communication.

[47] Ronald L. Rivest and Robert Sloan. A new model for inductive inference. In Moshe Vardi, editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 13–27, Morgan Kaufmann, March 1988.

[48] Robert E. Schapire. *Diversity-Based Inference of Finite Automata.* Master's thesis, MIT Lab. for Computer Science, May 1988. Technical Report MIT/LCS/TR-413.

[49] George Shackelford and Dennis Volper. Learning k-DNF with noise in the attributes. In *First Workshop on Computatinal Learning Theory*, pages 97–103, Morgan Kaufmann, Cambridge, Mass. August 1988.

[50] Haim Shvaytser. Linear manifolds are learnable from positive examples. April 1988. Unpublished manuscript.

[51] Leslie G. Valiant. Learning disjunctions of conjunctions. In *Proceedings IJCAI-85*, pages 560–566, International Joint Committee for Artificial Intelligence, Morgan Kaufmann, August 1985.

[52] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[53] Kurt VanLehn. Learning one subprocedure per lesson. *Artificial Intelligence*, 31(1):1–40, January 1987.

[54] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New York, 1982.

[55] 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.

[56] A. D. Wyner. An upper bound on the entropy series. *Information and Control*, 20:176–181, 1972.

[57] Andrew C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, 1982.