COMPUTATIONAL LEARNING THEORY

Sally A. Goldman, Washington University, St. Louis Missouri

1 Introduction

Since the late fifties, computer scientists (particularly those working in the area of artificial intelligence) have been trying to understand how to construct computer programs that perform tasks we normally think of as requiring human intelligence, and which can improve their performance over time by modifying their behavior in response to experience. In other words, one objective has been to design computer programs that can learn. For example, Samuels designed a program to play checkers in the early sixties that could improve its performance as it gained experience playing against human opponents. More recently, research on artificial neural networks has stimulated interest in the design of systems capable of performing tasks that are difficult to describe algorithmically (such as recognizing a spoken word or identifying an object in a complex scene), by exposure to many examples.

As a concrete example consider the task of hand-written character recognition. A learning algorithm is given a set of examples where each contains a hand-written character as specified by a set of attributes (e.g. the height of the letter) along with the name (label) for the intended character. This set of examples is often called the training data. The goal of the learner is to efficiently construct a rule (often referred to as a hypothesis or classifier) that can be used to take some previously unseen character and with high accuracy determine the proper label.

Computational learning theory is a branch of theoretical computer science that formally studies how to design computer programs that are capable of learning, and identifies the computational limits of learning by machines. Historically, researchers in the artificial intelligence community have judged learning algorithms empirically, according to their performance on sample problems. While such evaluations provide much useful information and insight, often it is hard using such evaluations to make meaningful comparisons among competing learning algorithms.

Computational learning theory provides a formal framework in which to precisely formulate

and address questions regarding the performance of different learning algorithms so that careful comparisons of both the predictive power and the computational efficiency of alternative learning algorithms can be made. Three key aspects that must be formalized are the way in which the learner interacts with its environment, the definition of successfully completing the learning task, and a formal definition of efficiency of both data usage (sample complexity) and processing time (time complexity). It is important to remember that the theoretical learning models studied are abstractions from real-life problems. Thus close connections with experimentalists are useful to help validate or modify these abstractions so that the theoretical results help to explain or predict empirical performance. In this direction, computational learning theory research has close connections to machine learning research. In addition to its predictive capability, some other important features of a good theoretical model are simplicity, robustness to variations in the learning scenario, and an ability to create insights to empirically observed phenomena.

The first theoretical studies of machine learning were performed by inductive inference researchers (see [Angluin and Smith, 1987]) beginning with the introduction of the first formal definition of learning given by [Gold, 1967]. In Gold's model, the learner receives a sequence of examples and is required to make a sequence of guesses as to the underlying rule (concept) such that this sequence converges at some finite point to a single guess that correctly names the unknown rule. A key distinguishing characteristic is that Gold's model does not attempt to capture any notion of the efficiency of the learning process whereas the field of computational learning theory emphasizes the computational feasibility of the learning algorithm.

Another closely related field is that of pattern recognition (see [Duda and Hart, 1973] and [Devroye, Györfi, and Lugosi, 1996]). Much of the theory developed by learning theory researchers to evaluate the amount of data needed to learn directly adapts results from the fields of statistics and pattern recognition. A key distinguishing factor is that learning theory researchers study both the data (information) requirements for learning and the time complexity of the learning algorithm. (In contrast, pattern recognition researchers tend to focus on issues related to the data requirements.) Finally, there are a lot of close relations to work on artificial neural networks. While much of neural network research is empirical, there is also a good amount of theoretical

work (see [Devroye, Györfi, and Lugosi, 1996]).

This chapter is structured as follows. In Section 2 we describe the basic framework of concept learning and give notation that we use throughout the chapter. Next, in Section 3 we describe the PAC (distribution-free) model that began the field of computational learning theory. An important early result in the field is the demonstration of the relationships between the VC-dimension, a combinatorial measure, and the data requirements for PAC learning. We then discuss some commonly studied noise models and general techniques for PAC learning from noisy data.

In Section 4 we cover some of the other commonly studied formal learning models. In Section 4.1 we study an on-line learning model. Unlike the PAC model in which there is a training period, in the on-line learning model the learner must improve the quality of its predictions as it functions in the world. Next, in Section 4.2, we study the query model which is very similar to the on-line model except that the learner plays a more active role.

As in other theoretical fields, along with having techniques to prove positive results, another important component of learning theory research is to develop and apply methods to prove when a learning problem is hard. In Section 5 we describe some techniques used to show that learning problems are hard. Next, in Section 6, we explore a variation of the PAC model called the weak learning model, and study techniques for boosting the performance of a mediocre learning algorithm. Finally, we close with a brief overview of some of the many current research issues being studied within the field of computational learning theory.

2 General Framework

For ease of exposition, we initially focus on concept learning in which the learner's goal is to infer how an unknown target function classifies (as positive or negative) examples from a given domain. In the character recognition example, one possible task of the learner would be to classify each character as a numeral or non-numeral. Most of the definitions given here naturally extend to the general setting of learning functions with multiple-valued or real-valued outputs. Later in Section 4.1 we briefly discuss the more general problem of learning a real-valued function.

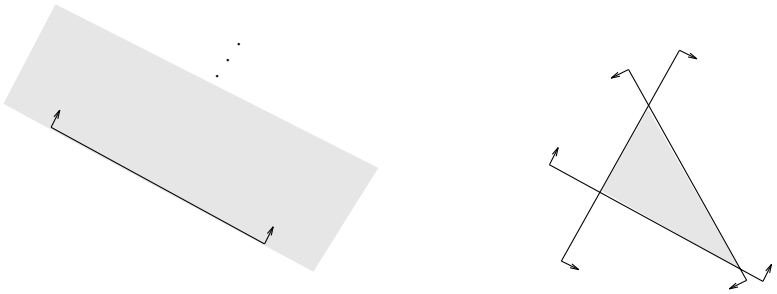

Figure 1: The left figure shows (in $\Re^2$) a concept from $\mathcal{C}_{\text{halfspace}}$ and the right figure shows a concept for $\mathcal{C}_{\text{halfspace}}^{\cap_s}$ for $s = 3$. The points from $\Re^2$ that are classified as positive are shaded. The unshaded points are classified as negative.

2.1 Notation

The instance space (domain) $\mathcal{X}$ is the set of all possible objects (instances) to be classified. Two very common domains are the boolean hypercube $\{0,1\}^n$, and the continuous n-dimensional space $\Re^n$. For example, if there are n boolean attributes then each example can be expressed as an element of $\{0,1\}^n$. Likewise, if there are n real-valued attributes, then $\Re^n$ can be used. A concept f is a boolean function over domain $\mathcal{X}$. Each $x \in \mathcal{X}$ is referred to as an example (point).

A **concept class** $\mathcal{C}$ is a collection of subsets of $\mathcal{X}$. That is $\mathcal{C} \subseteq 2^{\mathcal{X}}$. Typically (but not always), it is assumed that the learner has prior knowledge of $\mathcal{C}$. Examples are classified according to membership of a target concept $f \in \mathcal{C}$. Often, instead of using this set theoretic view of $\mathcal{C}$, a functional view is used in which $f(x)$ gives the classification of concept $f \in \mathcal{C}$ for each example $x \in \mathcal{X}$. An example $x \in \mathcal{X}$ is a positive example of f if $f(x) = 1$ (equivalently $x \in f$), or a negative example of f if $f(x) = 0$ (or equivalently $x \notin f$). Often $\mathcal{C}$ is decomposed into subclasses $\mathcal{C}_n$ according to some natural size measure n for encoding an example. For example, in the boolean domain, n is the number of boolean attributes. Let $\mathcal{X}_n$ denote the set of examples to be classified for each problem of size n, $\mathcal{X} = \bigcup_{n \geq 1} \mathcal{X}_n$.

We use the class $\mathcal{C}_{\text{halfspace}}$, the set of halfspaces in $\Re^n$, and the class $\mathcal{C}_{\text{halfspace}}^{\cap_s}$, the set of all the intersections of up to s halfspaces in $\Re^d$, to illustrate some of the basic techniques for designing

learning algorithms[1] (see Figure 1). We now formally define a halfspace in $\Re^n$ and describe how the points in $\Re^n$ are classified by it. Let $\vec{x} = (x_1, \ldots, x_n)$ denote an element of $\Re^n$. A halfspace defined by $\vec{a} \in \Re^n$ and $b \in \Re$ classifies as positive the set of points $\{\vec{x} \mid \vec{a} \cdot \vec{x} \geq b\}$. For example, in two-dimensions (i.e. $n = 2$), $5x_1 - 2x_2 \geq -3$ defines a halfspace. The point $(0, 0)$ is a positive example and $(2, 10)$ is a negative example. For $\mathcal{C}^{\cap_s}_{\text{halfspace}}$, an example is positive exactly when it is classified as positive by each of the halfspaces forming the intersection. Thus the set of positive points forms a (possibly open) convex polytope in $\Re^n$.

We also study some boolean concepts. For these concepts the domain is $\{0, 1\}^n$. Let $x_1, \ldots, x_n$ denote the n boolean attributes. A literal is either x_i or $\overline{x_i}$ where $i = 1, \ldots, n$. A term is a conjunction of literals. Finally, a DNF formula is a disjunction of terms. One of the biggest open problems of computational learning theory is whether or not the concept class of DNF formulas is efficiently PAC learnable. Since the problem of learning general DNF formulas is a long-standing open problem, several subclasses have been studied. A monotone DNF formula is a DNF formula in which there are no negated variables. A read-once DNF formula is a DNF formula in which each variable appears at most once. In addition to adding a restriction that the formula be monotone and/or read-once, one can also limit either the size of each term or the number of terms. A k-term DNF formula is a DNF formula in which there are at most k terms. Finally, a k-DNF formula is a DNF formula in which at most k literals are used by each term.

As one would expect, the time and sample complexity of the learning algorithm depends on the complexity of the underlying target concept. For example, the complexity for learning a monotone DNF formula is likely to depend on the number of terms (conjuncts) in the formula. To give a careful definition of the size of a concept $f \in \mathcal{C}$, we associate a language $\mathcal{R}_{\mathcal{C}}$ with each concept class $\mathcal{C}$ that is used for representing concepts in $\mathcal{C}$. Each $r \in \mathcal{R}_{\mathcal{C}}$ denotes some $f \in \mathcal{C}$, and every $f \in \mathcal{C}$ has at least one representation $r \in \mathcal{R}_{\mathcal{C}}$. Each concept $f \in \mathcal{C}_n$ has a size denoted by $|f|$, which is the representation length of the shortest $r \in \mathcal{R}_{\mathcal{C}}$ that denotes f. For ease of exposition, in the remainder of this chapter we use $\mathcal{C}$ and $\mathcal{R}_{\mathcal{C}}$ interchangeably.

To appreciate the significance of the choice of the representation class, consider the problem of

[1] As a convention, when the number of dimensions can be arbitrary we use n, and when the number of dimensions must be a constant we use d.

learning a regular language[2]. The question as to whether an algorithm is efficient depends heavily on the representation class. As defined more formally below, an efficient learning algorithm must have time and sample complexity polynomial in $|f|$ where f is the target concept. The target regular language could be represented as a deterministic finite-state automaton (DFA) or as a non-deterministic finite-state automaton (NFA). However, the length of the representation as a NFA can be exponentially smaller than the shortest representation as a DFA. Thus, the learner may be allowed exponentially more time when learning the class of regular languages as represented by DFAs versus when learning the class of regular languages as represented by NFAs. Often to make the representation class clear, the concept class names the representation. Thus instead of talking about learning a regular language, we talk about learning a DFA or an NFA. Thus the problem of learning a DFA is easier than learning an NFA. Similar issues arise when learning boolean functions. For example, whether the function is represented as a decision tree, a DNF formula, or a boolean circuit greatly affects the representation size.

3 PAC Learning Model

The field of computational learning theory began with Valiant's seminal work [Valiant, 1984] in which he defined the PAC[3](distribution-free) learning model. In the PAC model examples are generated according to an unknown probability distribution $\mathcal{D}$, and the goal of a learning algorithm is to classify with high accuracy (with respect to the distribution $\mathcal{D}$) any further (unclassified) examples.

We now formally define the PAC model. To obtain information about an unknown target function $f \in \mathcal{C}_n$, the learner is provided access to labeled (positive and negative) examples of f, drawn randomly according to some unknown target distribution $\mathcal{D}$ over $\mathcal{X}_n$. The learner is also given as input ϵ and δ such that $0 < \epsilon, \delta < 1$, and an upper bound k on $|f|$. The learner's goal is to output, with probability at least $1 - \delta$, a hypothesis $h \in \mathcal{C}_n$ that has probability at

[2]See Section 2.1 of Chapter 30 and Section 2 of Chapter 31 in this handbook for background on regular languages, DFAs and NFAs.

[3]PAC is an acronym coined by Dana Angluin for probably approximately correct.

most ϵ of disagreeing with f on a randomly drawn example from $\mathcal{D}$ (thus, h has error at most ϵ). If such a learning algorithm $\mathcal{A}$ exists (that is, an algorithm $\mathcal{A}$ meeting the goal for any $n \geq 1$, any target concept $f \in \mathcal{C}_n$, any target distribution $\mathcal{D}$, any $\epsilon, \delta > 0$, and any $k \geq |f|$), then $\mathcal{C}$ is **PAC learnable**. A PAC learning algorithm is a polynomial-time (efficient) algorithm if the number of examples drawn and the computation time are polynomial in n, k, $1/\epsilon$, and $1/\delta$. We note that most learning algorithms are really functions from samples to hypotheses (i.e. given a sample S the learning algorithm produces a hypothesis h). It is only for the analysis in which we say that for a given ϵ and δ, the learning algorithm is guaranteed with probability at least $1 - \delta$ to output a hypothesis with error at most ϵ given a sample whose size is a function of ϵ, δ, n and k. Thus, empirically, one can generally run a PAC algorithm on provided data and then empirically measure the error of the final hypothesis. One exception is when trying to empirically use statistical query algorithms since, for most of these algorithms, the algorithm uses ϵ for more than just determining the desired sample size. (See Section 3.3 for a discussion of statistical query algorithms, and [Goldman and Scott, 1996] for a discussion of their empirical use).

As originally formulated, PAC learnability also required the hypothesis to be a member of the concept class $\mathcal{C}_n$. We refer to this more stringent learning model as proper PAC-learnability. The work of [Pitt and Valiant, 1988] shows that a prerequisite for proper PAC-learning is the ability to solve the consistent hypothesis problem, which is the problem of finding a concept $f \in \mathcal{C}$ that is consistent with a provided sample. Their result implies that if the consistent hypothesis problem is NP-hard for a given concept class (as they show is the case for the class of k-term DNF formulas) and NP $\neq$ RP, then the learning problem is hard[4]. A more general form of learning in which the goal is to find any polynomial-time algorithm that classifies instances accurately in the PAC sense is commonly called prediction. In this less stringent variation of the PAC model, the algorithm need not output a hypothesis from the concept class $\mathcal{C}$ but instead is just required to make its prediction in polynomial time. This idea of prediction in the PAC model originated in the paper of [Haussler, Littlestone, and Warmuth, 1994], and is discussed in [Pitt and Warmuth, 1990]. Throughout the remainder of this chapter, when referring to the

[4]For background on Complexity Classes see Chapter 33 (NP defined) and Chapter 35, Section 3 (RP defined) of this handbook.

PAC learning model we allow the learner to output any hypothesis that can be evaluated in polynomial time. That is, given a hypothesis h and an example x, we require that $h(x)$ can be computed in polynomial time. We refer to the model in which the learner is required to output a hypothesis $h \in \mathcal{C}_n$ as the proper PAC learning model.

Many variations of the PAC model are known to be equivalent (in terms of what concept classes are efficiently learnable) to the model defined above. We now briefly discuss one of these variations. In the above definition of the PAC model, along with receiving ϵ, δ, and n as input, the learner also receives k, an upperbound on $|f|$. The learner must be given ϵ and δ. Further, by looking at just one example, the value of n is known. Yet giving the learner knowledge of k may appear to make the problem easier. However, if the demand of polynomial-time computation is replaced with expected polynomial-time computation, then the learning algorithm need not be given the parameter k, but could "guess" it instead. We now briefly review the standard doubling technique used to convert an algorithm A designed to have k as input to an algorithm B that has no prior knowledge of k. Algorithm B begins with an estimate, say 1, for its upperbound on k and runs algorithm A using this estimate to obtain hypothesis h. Then algorithm B uses a hypothesis testing procedure to determine if the error of h is at most ϵ. Since the learner can only gather a random sample of examples, it is not possible to distinguish a hypothesis with error ϵ from one with error just greater than ϵ. However, by drawing a sufficiently large sample and looking at the empirical performance of h on that sample, we can distinguish a hypothesis with error at most $\epsilon/2$ from one with error more than ϵ. In particular, given a sample[5] of size $m = \left\lceil \frac{32}{\epsilon} \ln \frac{2}{\delta} \right\rceil$, if h misclassifies at most $\frac{3\epsilon}{4} \cdot m$ examples then B accepts h. Otherwise, algorithm B doubles its estimate for k and repeats the process. For the technical details of the above argument as well as for a discussion the other equivalences, see [Haussler, Kearns, Littlestone, and Warmuth, 1991].

[5]Chernoff bounds are used to compute the sample size so that the hypothesis testing procedure gives the desired output with high probability.

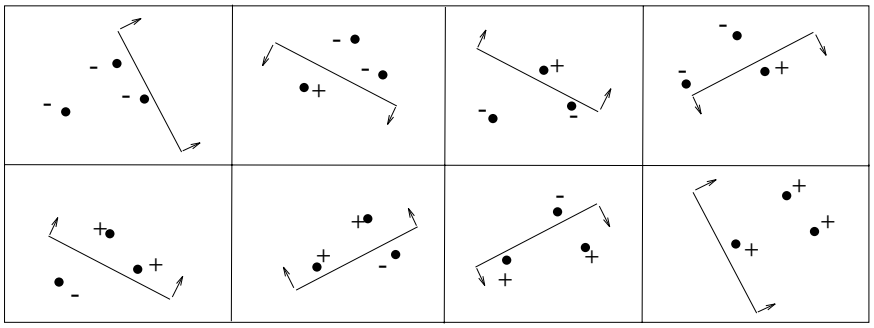

Figure 2: A demonstration that there are 3 points that are shattered by the class of two-dimensional halfspaces (i.e. $\mathcal{C}_{\text{halfspace}}$ with $n = 2$). Notice that all 8 possible ways that the points can be classified as positive or negative can be realized.

3.1 Sample Complexity Bounds, the VC-Dimension, and Occam's Razor

Although we are concerned with the time complexity of the learning algorithm, a fundamental question to first answer is the sample complexity (data) requirements. [Blumer, Ehrenfeucht, Haussler, and Warmuth, 1989] identified a combinatorial parameter of a class of functions defined by [Vapnik and Chervonenkis, 1971]. They give strong results relating this parameter, called the **VC-dimension**, to information-theoretic bounds on the sample size needed to have accurate generalization. Note that given a sufficiently large sample there is still the computational problem of finding a "good" hypothesis.

We now define the VC-dimension. We say that a finite set $S \subseteq \mathcal{X}$ is shattered by the concept class $\mathcal{C}$ if for each of the $2^{|S|}$ subsets $S' \subseteq S$, there is a concept $f \in \mathcal{C}$ that contains all of S' and none of $S - S'$. In other words, for any of the $2^{|S|}$ possible labelings of S (where each example $s \in S$ is either positive or negative), there is some $f \in \mathcal{C}$ that realizes the desired labeling (see Figure 2). We can now define the VC-dimension of a concept class $\mathcal{C}$. The VC-dimension of $\mathcal{C}$, denoted $\text{VCD}(\mathcal{C})$, is the smallest d for which no set of $d + 1$ examples is shattered by $\mathcal{C}$. Equivalently, $\text{VCD}(\mathcal{C})$ is the cardinality of the largest finite set of points $S \subseteq X$ that is shattered by $\mathcal{C}$.

So to prove that $\text{VCD}(\mathcal{C}) \geq d$ it suffices to give d examples that can be shattered. However, to prove $\text{VCD}(\mathcal{C}) \leq d$ one must show that no set of $d + 1$ examples can be shattered. Since the VC-

dimension is so fundamental in determining the sample complexity required for PAC learning, we now go through several sample computations of the VC-dimension.

Axis-parallel rectangles in $\Re^2$. For this concept class the points lying on or inside the target rectangle are positive, and points lying outside the target rectangle are negative. First, it is easily seen that there is a set of four points (e.g. $\{(0,1),(0,-1),(1,0),(-1,0)\}$) that can be shattered. Thus $\text{VCD}(\mathcal{C}) \geq 4$. We now argue that no set of five points can be shattered. The smallest bounding axis-parallel rectangle defined by the five points is in fact defined by at most four of the points. For p a non-defining point in the set, we see that the set cannot be shattered since it is not possible for p to be classified as negative while also classifying the others as positive. Thus $\text{VCD}(\mathcal{C}) = 4$.

Halfspaces in $\Re^2$. Points lying in or on the halfspace are positive, and the remaining points are negative. It is easily shown that any three non-collinear points (e.g. $(0,1),(0,0),(1,0)$) are shattered by $\mathcal{C}$ (recall Figure 2). Thus $\text{VCD}(\mathcal{C}) \geq 3$. We now show that no set of size four can be shattered by $\mathcal{C}$. If at least three of the points are collinear then there is no halfspace that contains the two extreme points but does not contain the middle points. Thus the four points cannot be shattered if any three are collinear. Next, suppose that the points form a quadrilateral. There is no halfspace which labels one pair of diagonally opposite points positive and the other pair of diagonally opposite points negative. The final case is that one point p is in the triangle defined by the other three. In this case there is no halfspace which labels p differently from the other three. Thus clearly the four points cannot be shattered. Therefore we have demonstrated that $\text{VCD}(\mathcal{C}) = 3$. Generalizing to halfspaces in $\Re^n$, it can be shown that $\text{VCD}(\mathcal{C}_{\text{halfspace}}) = n + 1$ (see [Blumer, Ehrenfeucht, Haussler, and Warmuth, 1989]).

Closed sets in $\Re^2$. All points lying in the set or on the boundary of the set are positive, and all points lying outside the set are negative. Any set can be shattered by $\mathcal{C}$, since a closed set can assume any shape in $\Re^2$. Thus, the largest set that can be shattered by $\mathcal{C}$ is infinite, and hence $\text{VCD}(\mathcal{C}) = \infty$.

We now briefly discuss techniques that aid in computing the VC-dimension of more complex concept classes. Suppose we wanted to compute the VC-dimension of $\mathcal{C}_{\text{halfspace}}^{\cap_s}$, the class of intersections of up to s halfspaces over $\Re^d$. We would like to make use of our knowledge of the VC-dimension of $\mathcal{C}_{\text{halfspace}}$, the class of halfspaces over $\Re^d$. [Blumer, Ehrenfeucht, Haussler, and Warmuth, 1989] gave the following result: let $\mathcal{C}$ be a concept class with $\text{VCD}(\mathcal{C}) \leq D$. Then the class defined by the intersection of up to s concepts from $\mathcal{C}$ has VC-dimension[6] at most $2Ds \lg(3s)$. In fact, the above result applies when replacing intersection with any boolean function. Thus the concept class $\mathcal{C}_{\text{halfspace}}^{\cap_s}$ (where each halfspace is defined over $\Re^d$) has VC-dimension at most $2(d+1)s \lg(3s)$.

We now discuss the significance of the VC-dimension to the PAC model. One important property is that for $D = \text{VCD}(\mathcal{C})$, the number of different labelings that can be realized (also referred to as behaviors) using $\mathcal{C}$ for a set S is at most $\left(\frac{e|S|}{D}\right)^D$. Thus for a constant D we have polynomial growth in the number of labelings versus the exponential growth of $2^{|S|}$. A key result in the PAC model is that of [Blumer, Ehrenfeucht, Haussler, and Warmuth, 1989] that gives an upperbound on the sample complexity needed to PAC learn in terms of ϵ, δ, and the VC-dimension of the hypothesis class. They proved that one can design a learning algorithm A for concept class $\mathcal{C}$ using hypothesis space $\mathcal{H}$ in the following manner. Any concept $h \in \mathcal{H}$ consistent with a sample of size $\max\left(\frac{4}{\epsilon}\lg\frac{2}{\delta}, \frac{8\,\text{VCD}(\mathcal{H})}{\epsilon}\lg\frac{13}{\epsilon}\right)$ has error at most ϵ with probability at least $1-\delta$. To obtain a polynomial-time PAC learning algorithm what remains is to solve the algorithmic problem of finding a hypothesis from $\mathcal{H}$ consistent with the labeled sample. Furthermore, [Ehrenfeucht and Haussler, 1989] proved an information-theoretic lower bound that learning any concept class $\mathcal{C}$ requires $\Omega\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{\text{VCD}(\mathcal{C})}{\epsilon}\right)$ examples in the worst case. As an example application, see Figure 3.

A key limitation of this technique to design a PAC learning algorithm is that the hypothesis must be drawn from some fixed hypothesis class $\mathcal{H}$. In particular, the complexity of the hypothesis class must be independent of the sample size. However, often the algorithmic problem of finding such a hypothesis from the desired class is NP-hard.

[6] Note that throughout this chapter, lg is used for the base-2 logarithm. When the base of the logarithm is not significant (such as when using asymptotic notation), we use log.

PAC-learn-halfspace(n, ϵ, δ)

Draw a labeled sample S of size $\max\left(\frac{4}{\epsilon}\lg\frac{2}{\delta}, \frac{8(n+1)}{\epsilon}\lg\frac{13}{\epsilon}\right)$

Use a polynomial-time linear programming algorithm to find a halfspace h consistent with S

Output h

Figure 3: A PAC algorithm to learn a concept from $\mathcal{C}_{\text{halfspace}}$. Recall that $\text{VCD}(\mathcal{C}_{\text{halfspace}}) = n+1$.

As an example suppose we tried to modify PAC-learn-halfspace from Figure 3 to efficiently PAC learn the class of intersections of at most s halfspaces in $\Re^d$ for d the number of dimensions a constant. Suppose we are given a sample S of m example points labeled according to some $f \in \mathcal{C}^{\cap_s}_{\text{halfspace}}$. The algorithmic problem of finding a hypothesis from $\mathcal{C}^{\cap_s}_{\text{halfspace}}$ can be formulated as a set covering problem in the following manner. An instance of the set covering problem is a set $\mathcal{U}$ and a family $\mathcal{T}$ of subsets of $\mathcal{U}$ such that $(\bigcup_{t\in\mathcal{T}}\ t) = \mathcal{U}$. A solution is a smallest cardinality subset $\mathcal{G}$ of $\mathcal{T}$ such that $\left(\bigcup_{g\in\mathcal{G}}\ g\right) = \mathcal{U}$. Consider any halfspace g that correctly classifies all positive examples from S. We say that g covers all of the negative examples from S that are also classified as negative by g. Thus $\mathcal{U}$ is the set of negative examples from S, and $\mathcal{T}$ is the set of representative halfspaces (one for each behavior with respect to S) that correctly classify all positive examples from S.

So finding a minimum sized hypothesis from $\mathcal{C}^{\cap_s}_{\text{halfspace}}$ consistent with the sample S is exactly the problem of finding the minimum number of halfspaces that are required to cover all negative points in S. Given that the set covering problem is NP-complete, how can we proceed? We can apply the greedy approximation algorithm that has a ratio bound of $\ln|\mathcal{U}| + 1$ to find a hypothesis consistent with S that is the intersection of at most $\ln|S| + 1$ halfspaces. However, since the VC-dimension of the hypothesis grows with the size of the sample, the basic technique described above cannot be applied. In general, when using a set covering approach, the size of the hypothesis often depends on the size of the sample.

[Blumer, Ehrenfeucht, Haussler, and Warmuth, 1987 and 1989] extended this basic technique by showing that finding a hypothesis h consistent with a sample S for which the size of h is sub-linear in $|S|$ is sufficient to guarantee PAC learnability. In other words, by obtaining sufficient data compression one obtains good generalization. More formally, let $\mathcal{H}^A_{k,n,m}$

PAC-learn-intersection-of-halfspaces(d, s, ϵ, δ)

Draw a labeled sample S of size $m = \max\left(\frac{4}{\epsilon}\lg\frac{2}{\delta}, \frac{192ds}{\epsilon}\left(6 + 4\lg 3 + \lg\frac{ds}{\epsilon}\right)^3\right)$
Form a set $\mathcal{T}$ of halfspaces that are consistent with the positive examples
Let N be the negative examples from S
Let h be the always true hypothesis
Repeat until $N = \emptyset$
    Find a halfspace $t \in \mathcal{T}$ consistent with the max. number of exs. from N
    Let $h = h \cap t$
    Remove the examples from N that were covered by t
Output h

Figure 4: An Occam algorithm to PAC learn a concept from $\mathcal{C}^{\cap_s}_{\text{halfspace}}$.

be the hypothesis space used by algorithm A when each example has size n, the target concept has size k, and the sample size is m. We say that algorithm A is an **Occam algorithm** for concept class $\mathcal{C}$ if there exists a polynomial $p(k,n)$ and a constant β, $0 \leq \beta < 1$, such that for any sample S with $|S| = m \geq 1$ and any k, n, A outputs a hypothesis[7] h consistent with S such that $|h| \leq p(k,n)m^\beta$. Let A be an Occam algorithm for concept class $\mathcal{C}$ that has hypothesis space $\mathcal{H}^A_{k,n,m}$. If $\text{VCD}(\mathcal{H}^A_{k,n,m}) \leq p(k,n)(\lg m)^\ell$ (so $|h| \leq p(k,n)(\lg m)^\ell$) for some polynomial $p(k,n) \geq 2$ and $\ell \geq 1$, then A is a PAC learning algorithm for $\mathcal{C}$ using sample size $m = \max\left(\frac{4}{\epsilon}\lg\frac{2}{\delta}, \frac{2^{\ell+4}p(k,n)}{\epsilon}\left(\lg\frac{8(2\ell+2)^{\ell+1}p(k,n)}{\epsilon}\right)^{\ell+1}\right)$.

Figure 4 presents an Occam algorithm to PAC learn a concept from $\mathcal{C}^{\cap_s}_{\text{halfspace}}$ (i.e. the intersection of at most s halfspaces over $\Re^d$). Since $\text{VCD}(\mathcal{C}^{\cap_s}_{\text{halfspace}}) \leq 2(d+1)s\lg(3s)$, and the hypothesis consists of the intersection of at most $s(1+\ln m)$ halfspaces, it follows that $\text{VCD}(\mathcal{H}) \leq 2(d+1)s(1+\ln m)\lg(3s(1+\ln m)) \leq 3ds\ln^2 m$. The size of the sample S then follows by noting that $p(s,d) = 3ds$ and $\ell = 3$. By combining the fact that $\text{VCD}(\mathcal{C}_{\text{halfspace}}) = d+1$ and the general upper bound of $\left(\frac{e|S|}{D}\right)^D$ on the number of behaviors on a sample of size $|S|$ for a class of VC-dimension D, we get that $|\mathcal{T}| \leq \left(\frac{em}{d+1}\right)^{d+1} = O(m^{d+1})$. (For d constant this quantity is polynomial in the size of the sample.) We can construct $\mathcal{T}$ in polynomial time by either using simple geometric arguments or the more general technique of [Blumer, Ehrenfeucht, Haussler, and Warmuth, 1989]. Finally, the time complexity of the greedy set covering algorithm is $O(\sum_{t\in\mathcal{T}}|t|) = O(m \cdot m^{d+1})$

[7] Recall that we use $|h|$ to denote the number of bits required to represent h.

which is polynomial in s (the number of halfspaces), ϵ, and δ for d (the number of dimensions) constant.

For the problem of learning the intersection of halfspaces the greedy covering technique provided substantial data compression. Namely, the size of our hypothesis only had a logarithmic dependence on the size of the sample. In general, only a sub-linear dependence is required as given by the following result of [Blumer, Ehrenfeucht, Haussler, and Warmuth, 1987]. Let A be an Occam algorithm for concept class $\mathcal{C}$ that has hypothesis space $\mathcal{H}^A_{k,n,m}$. If $\text{VCD}(Hyp^A_{k,n,m}) \leq p(k,n)m^\beta$ (so $|h| \leq p(k,n)m^\beta$) for some polynomial $p(k,n) \geq 2$ and $\beta < 1$, then A is a PAC learning algorithm for $\mathcal{C}$ using sample size $m = \max\left(\frac{2}{\epsilon}\ln\frac{1}{\delta}, \left(\frac{2\ln 2}{\epsilon}\cdot p(k,n)\right)^{\frac{1}{1-\beta}}\right)$.

3.2 Models of Noise

The basic definition of PAC learning assumes that the data received is drawn randomly from $\mathcal{D}$ and properly labeled according to the target concept. Clearly, for learning algorithms to be of practical use they most be robust to noise in the training data. In order to theoretically study an algorithm's tolerance to noise, several formal models of noise have been studied. In the model of Random **Classification Noise**([Angluin and Laird, 1988]) with probability $1-\eta$, the learner receives the uncorrupted example (x,ℓ). However, with probability η, the learner receives the example $(x,\bar{\ell})$. So in this noise model, learner usually gets a correct example, but some small fraction η of the time the learner receives an example in which the label has been inverted. In the model of Malicious Classification Noise([Sloan, 1988]) with probability $1-\eta$, the learner receives the uncorrupted example (x,ℓ). However, with probability η, the learner receives the example (x,ℓ') in which x is unchanged, but the label ℓ' is selected by an adversary who has infinite computing power and has knowledge of the learning algorithm, the target concept, and the distribution $\mathcal{D}$. In the previous two models, only the labels are corrupted. Another noise model is that of Malicious Noise([Valiant, 1985]). In this model, with probability $1-\eta$, the learner receives the uncorrupted example (x,ℓ). However, with probability η, the learner receives an example (x',ℓ') about which no assumptions whatsoever may be made. In particular, this example (and its label) may be maliciously selected by an adversary. Thus in this model, the learner usually gets a correct example, but some small fraction η of the time the learner gets

noisy examples and the nature of the noise is unknown.

We now define two noise models that are only defined when the instance space is $\{0,1\}^n$. In the model of Uniform Random Attribute Noise ([Sloan, 1988]), the example $(b_1 \cdots b_n, \ell)$ is corrupted by a random process that independently flips each bit b_i to $\bar{b}_i$ with probability η for $1 \leq i \leq n$. Note that the label of the "true" example is never altered. In this noise model, the attributes' values are subject to noise, but that noise is as benign as possible. For example, the attributes' values might be sent over a noisy channel, but the label is not. Finally, in the model of Product Random Attribute Noise([Goldman and Sloan, 1995]), the example $(b_1 \cdots b_n, \ell)$ is corrupted by a random process of independently flipping each bit b_i to $\bar{b}_i$ with some fixed probability $\eta_i \leq \eta$ for each $1 \leq i \leq n$. Thus unlike the model of uniform random attribute noise, the noise rate associated with each bit of the example may be different.

3.3 Gaining Noise Tolerance in the PAC Model

Some of the first work on designing noise-tolerant PAC algorithms was done by [Angluin and Laird, 1988]. They gave an algorithm for learning boolean conjunctions that tolerates random classification noise of a rate approaching the information-theoretic barrier of 1/2. Furthermore, they proved that the general technique of finding a hypothesis that minimizes disagreements with a sufficiently large sample allows one to handle random classification noise of any rate approaching 1/2. However, they showed that even the very simple problem of minimizing disagreements (when there are no assumptions about the noise) is NP-hard. Until recently, there have been a small number of efficient noise-tolerant PAC algorithms, but no general techniques were available to design such algorithms, and there was little work to characterize which concept classes could be efficiently learned in the presence of noise.

The first (computationally feasible) tool to design noise-tolerant PAC algorithms was provided by the **statistical query model**, first introduced by [Kearns, 1993], and since improved by [Aslam and Decatur, 1997]. In this model, rather than sampling labeled examples, the learner requests the value of various statistics. A relative-error statistical query [Aslam and Decatur, 1997] takes the form $\mathbf{SQ}(\chi, \mu, \theta)$ where χ is a predicate over labeled examples, μ is a relative error bound, and θ is a threshold. As an example, let χ to be "$(h(x) = 1) \wedge (\ell = 0)$"

which is true when x is a negative example but the hypothesis classifies x as positive. So the probability that χ is true for a random example is the false positive error of hypothesis h. For target concept f, we define $P_\chi = \Pr_{\mathcal{D}}[\chi(x, f(x))) = 1]$ where $\Pr_{\mathcal{D}}$ is used to denote that x is drawn at random from distribution $\mathcal{D}$. If $P_\chi < \theta$ then $\mathbf{SQ}(\chi, \mu, \theta)$ may return $\perp$. If $\perp$ is not returned, then $\mathbf{SQ}(\chi, \mu, \theta)$ must return an estimate $\hat{P}_\chi$ such that $P_\chi(1-\mu) \leq \hat{P}_\chi \leq P_\chi(1+\mu)$. The learner may also request unlabeled examples (since we are only concerned about classification noise).

Let's take our algorithm, PAC-learn-intersection-of-halfspaces and reformulate it as a relative-error statistical query algorithm. First we draw an unlabeled sample S_u that we use to generate the set $\mathcal{T}$ of halfspaces to use for our covering. Similar to before, we place a halfspace in $\mathcal{T}$ corresponding to each possible way in which the points of S_u can be divided. Recall that before, we only place a halfspace in $\mathcal{T}$ if it properly labeled the positive examples. Since we have an unlabeled sample we cannot use such an approach. Instead, we use a statistical query (for each potential halfspace) to check if a given halfspace is consistent with most of the positive examples. Finally, when performing the greedy covering step we cannot pick the halfspace that covers the most negative examples, but rather the one that covers the "most" (based on our empirical estimate) negative weight. Figure 5 shows our algorithm in more depth.

We now cover the key ideas in proving that SQ-learn-intersection-of-halfspaces is correct. First, we pick S_u so that any hypothesis consistent with S_u (if we knew the labels) would have error at most $\frac{\epsilon}{6r}$ with probability $1 - \frac{\delta}{2}$. Since our hypothesis class is $\mathcal{C}^{\cap_r}_{\text{halfspace}}$, and $\text{VCD}(\mathcal{C}^{\cap_r}_{\text{halfspace}}) \leq 2(d+1)r\lg(3r)$, we obtain the sample size for S_u.

For each of the s halfspaces that form the target concept, there is some halfspace in $\mathcal{T}$ consistent with that halfspace over S_u, and thus that has error at most $\epsilon/(6r)$ on the positive region. For each such halfspace t our estimate $\hat{P}_\chi$ (for the probability that $t(x) = 0$ and $\ell = 1$) is at most $\frac{\epsilon}{6r} \cdot \frac{3}{2} = \frac{\epsilon}{4r}$. Thus, we place s halfspaces in $\mathcal{T}_{good}$ that produce a hypothesis with false negative error $\leq \epsilon/(6r)$. Let $e_i = \Pr[h(x) = 1 \wedge \ell = 0]$ (i.e. the false positive error) after i rounds of the while loop. Since e_i is our current error and $\epsilon/(6r)$ is a lower bound on the error we could

SQ-learn-intersection-of-halfspaces(d, s, ϵ)

Let $r = 2s \ln(3/\epsilon)$

Draw an unlabeled sample $S_u = \max\left\{\frac{24r}{\epsilon}\lg\frac{4}{\delta}, \frac{96(d+1)r^2\lg(3r)}{\epsilon}\lg\frac{78r}{\epsilon}\right\}$

Form the set $\mathcal{T}$ of halfspaces - one for each linear separation of S_u into two sets

$\mathcal{T}_{good} = \emptyset$

For each $t \in \mathcal{T}$

$\hat{P_\chi} = \mathbf{SQ}\left(t(x) = 0 \wedge \ell = 1, \frac{1}{2}, \frac{\epsilon}{2r}\right)$

If $\hat{P_\chi} \leq \frac{\epsilon}{4r}$ or $\perp$ Then $\mathcal{T}_{good} = \mathcal{T}_{good} \cup t$

Let h be the always true hypothesis

While $\mathbf{SQ}\left(h(x) = 1 \wedge \ell = 0, \frac{1}{9}, \frac{\epsilon}{2}\right) > \frac{4\epsilon}{9}$ and the loop has been executed $< r$ times

$t_{max} = \text{argmax}_{t \in \mathcal{T}_{good}} \mathbf{SQ}\left(h(x) = 1 \wedge \ell = 0 \wedge t(x) = 0, \frac{1}{3}, \frac{7\epsilon}{30s}\right)$

$h = h \cap t_{max}$

Output h

Figure 5: A relative-error statistical query algorithm to learn a concept from $\mathcal{C}^{\cap_s}_{\text{halfspace}}$ over $\Re^d$.

achieve, by an averaging argument it follows that there is a $t \in \mathcal{T}_{good}$ for which

$$P = \Pr[(h(x) = 1) \wedge (\ell = 0) \wedge (t(x) = 0)] \geq \left(e_i - \frac{\epsilon}{6r}\right) \cdot \frac{1}{s} \geq \left(e_i - \frac{\epsilon}{6}\right) \cdot \frac{1}{s}.$$

Thus for the halfspace t_{max} selected to add to h we know that the statistical query returns an estimate $\hat{P} \geq \frac{2}{3} \cdot P \geq \frac{2}{3s}(e_i - \frac{\epsilon}{6})$. Thus for t_{max} we know that $P \geq \frac{3}{4} \cdot \frac{2}{3s}(e_i - \frac{\epsilon}{6}) = \frac{1}{2s}(e_i - \frac{\epsilon}{6})$. Solving the recurrence $e_{i+1} \leq e_i - \frac{1}{2s}(e_i - \frac{\epsilon}{6})$ (with $e_i \leq 1$) yields that $e_i \leq \frac{\epsilon}{6} + \left(1 - \frac{1}{2s}\right)^i$. Picking $r = i = 2s \ln\frac{3}{\epsilon}$ suffices to ensure that $e_i \leq \epsilon/2$ as desired.

Once $e_i \leq 2\epsilon/5$, we exit the while loop (since $\frac{2\epsilon}{5} \cdot \frac{10}{9} = \frac{4\epsilon}{9}$). Thus we enter the loop only when $e_i > 2\epsilon/5$, and hence for t_{max}, $P \geq \frac{1}{s}(e_i - \frac{\epsilon}{6})$. So we choose $\theta = \frac{1}{s}(\frac{2\epsilon}{5} - \frac{\epsilon}{6}) = \frac{7\epsilon}{30s}$. Finally, when we exit the loop $\Pr[h(x) = 1 \wedge \ell = 0] \leq \frac{4\epsilon}{9} \cdot \frac{9}{8} = \frac{\epsilon}{2}$. Thus the total error is at most ϵ as desired.

Notice that in the statistical query model there is not a confidence parameter δ. This is because the SQ oracle guarantees that its estimates meet the given requirements. However, when we use random labelled examples to simulate the statistical queries, we can only guarantee that with high probability the estimates meet their requirements. Thus the results on converting an SQ algorithm into a PAC algorithm re-introduce the confidence parameter δ.

By applying uniform convergence results and Chernoff bounds it can be shown that if one draws a sufficiently large sample then a statistical query can be estimated by the fraction of the sample that satisfies the predicate. For example, in the relative-error SQ model with a set $\mathcal{Q}$ of

possible queries, [Aslam and Decatur, 1997] show that a sample of size $O\left(\frac{1}{\mu^2\theta}\log\frac{|\mathcal{Q}|}{\delta}\right)$ suffices to appropriately answer $[\chi,\mu,\theta]$ for every $\chi\in\mathcal{Q}$ with probability at least $1-\delta$. (If $\text{VCD}(\mathcal{Q})=q$ then a sample of size $O\left(\frac{q}{\mu^2\theta}\log\frac{1}{\mu\theta}+\frac{1}{\mu^2\theta}\log\frac{1}{\delta}\right)$ suffices.)

To handle random classification noise of any rate approaching 1/2 more complex methods are used for answering the statistical queries. Roughly, using knowledge of the noise process and a sufficiently accurate estimate of the noise rate (which must itself be determined by the algorithm), the noise process can be "inverted." The total sample complexity required to simulate an SQ algorithm in the presence of classification noise of rate $\eta\leq\eta_b$ is $\tilde{O}\left(\frac{\log(|\mathcal{Q}|/\delta)}{\mu_*^2\theta_*\rho(1-2\eta_b)^2}\right)$ where μ_* (respectively, θ_*) is the minimum value of μ (respectively, θ) across all queries and $\rho\in[\theta_*,1]$, The soft-oh notation ($\tilde{O}$) is similar to the standard big-oh notation except dominate log factors are also removed. Alternatively, for the query space $\mathcal{Q}$, the sample complexity is $O\left(\frac{\text{VCD}(\mathcal{Q})}{\mu_*^2\theta_*\rho_*(1-2\eta_b)^2}\left[\log\left(\frac{1}{\mu_*\theta_*\rho_*(1-2\eta_b)}\right)+\log\frac{1}{\delta}\right]\right)$. Notice that the amount by which η_b is less than 1/2 is just $\frac{1}{1/2-\eta_b}=\frac{2}{1-2\eta_b}$. Thus the above are polynomial as long as $\frac{1}{2}-\eta_b$ is at least one over a polynomial.

4 Exact and Mistake Bounded Learning Models

The PAC learning model is a batch model—there is a separation between the training phase and the performance phase. In the training phase the learner is presented with labeled examples — no predictions are expected. Then at the end of the training phase the learner must output a hypothesis h to classify unseen examples. Also, since the learner never finds out the true classification for the unlabeled instances, all learning occurs in the training phase. In many settings, the learner does not have the luxury of a training phase but rather must learn as it performs. We now study two closely related learning models designed for such a setting.

4.1 On-line Learning Model

To motivate the **on-line learning model** (also known as the mistake-bounded learning model), suppose that when arriving at work (in Boston) you may either park in the street or park in a garage. In fact, between your office building and the garage there is a street on which you

can always find a spot. On most days, street parking is preferable since you avoid paying the \$15 garage fee. Unfortunately, when parking on the street you risk being towed (\$75) due to street cleaning, snow emergency, special events, etc. When calling the city to find out when they tow, you are unable to get any reasonable guidance and decide the best thing to do is just learn from experience. There are many pieces of information that you might consider in making your prediction: e.g. the date, the day of the week, the weather. We make the following assumption: after you commit yourself to one choice or the other you learn of the right decision. In this example, the city has rules dictating when they tow; you just don't know them. If you park on the street, at the end of the day you know if your car was towed; otherwise when walking to the garage you see if the street is clear (i.e. you learn if you would have been towed). The on-line model is designed to study algorithms for learning to make accurate predictions in circumstances such as these. Note that unlike the problems addressed by many techniques from reinforcement learning, here there is immediate (versus delayed) feedback.

We now define the on-line learning model for the general setting in which the target function has a real-valued output (without loss of generality, scaled to be between 0 and 1). An on-line learning algorithm for $\mathcal{C}$ is an algorithm that runs under the following scenario. A learning session consists of a set of trials. In each trial, the learner is given an unlabeled instance $x \in \mathcal{X}$. The learner uses its current hypothesis to predict a value $p(x)$ for the unknown (real-valued) target concept $f \in \mathcal{C}$ and then the learner is told the correct value for $f(x)$. Several loss functions have been considered to measure the quality of the learner's predictions. Three commonly used loss functions are the following: the square loss defined by $\ell_2(p(x), f(x)) = (f(x) - p(x))^2$, the log loss defined by $\ell_{\log}(p(x), f(x)) = -f(x) \log p(x) - (1 - f(x)) \log(1 - p(x))$, and the absolute loss defined by $\ell_1(p(x), f(x)) = |f(x) - p(x)|$.

The goal of the learner is to make predictions so that total loss over all predictions is minimized. In this learning model, most often a worst-case model for the environment is assumed. There is some known concept class from which the target concept is selected. An adversary (with unlimited computing power and knowledge of the learner's algorithm) then selects both the target function and the presentation order for the instances. In this model there is no training

phase — Instead, the learner receives unlabeled instances throughout the entire learning session. However, after each prediction the learner "discovers" the correct value. This feedback can then be used by the learner to improve its hypothesis.

We now discuss the special case when the target function is boolean and correspondingly, predictions must be either 0 or 1. In this special case the loss function most commonly used is the absolute loss. Notice that if the prediction is correct then the value of the loss function is 0, and if the prediction is incorrect then the value of the loss function is 1. Thus the total loss of the learner is exactly the number of prediction mistakes. Thus, in the worst-case model we assume that an adversary selects the order in which the instances are presented to the learner and we evaluate the learner by the maximum number of mistakes made during the learning session. Our goal is to minimize the worst-case number of mistakes using an efficient learning algorithm (i.e. each prediction is made in polynomial time). Observe that such mistake bounds are quite strong in that the order in which examples are presented does not matter; however, it is impossible to tell how early the mistakes will occur. [Littlestone, 1988] has shown that in this learning model $\text{VCD}(\mathcal{C})$ is a lower bound on the number of prediction mistakes.

4.1.1 Handling Irrelevant Attributes

Here we consider the common scenario in which there are many attributes the learner could consider, yet the target concept depends on a small number of them. Thus most of the attributes are irrelevant to the target concept. We now briefly describe one early algorithm, Winnow of [Littlestone, 1988], that handles a large number of irrelevant attributes. More specifically, for Winnow, the number of mistakes only grows logarithmically with the number of irrelevant attributes. Winnow (or modifications of it) have many nice features such as noise tolerance and the ability to handle the situation in which the target concept is changing. Also, Winnow can directly be applied to the agnostic learning model [Kearns, Schapire, and Sellie, 1994]. In the agnostic learning model no assumptions are made about the target concept. In particular, the learner is unaware of any concept class that contains the target concept. Instead, we compare the performance of an agnostic algorithm (typically in terms of the number of mistakes or based on some other loss function) to the performance of the best hypothesis selected from a comparison

Winnow(θ, α)

For $i = 1, \ldots, n$, initialize $w_i = 1$

To predict the value of $x = (x_1, \ldots, x_n) \in X$:
If $\sum_{i=1}^{n} w_i x_i > \theta$, then predict 1
Else predict 0

Let ρ be the correct prediction
If the prediction was 1 and $\rho = 0$ then
If $x_i = 1$ then let $w_i = w_i / \alpha$
If the prediction was 0 and $\rho = 1$ then
If $x_i = 1$ then let $w_i = w_i \cdot \alpha$

Figure 6: The algorithm Winnow.

or "touchstone" class where the best hypothesis from the touchstone class is the one that incurs the minimum loss over all functions in the touchstone class.

Winnow is similar to the classical perceptron algorithm[Rosenblatt, 1958], except that it uses a multiplicative weight-update scheme that permits it to perform much better than classical perceptron training algorithms when many attributes are irrelevant. The basic version of Winnow is designed to learn the concept class of a linearly separable boolean function which is a map $f : \{0,1\}^n \to \{0,1\}$ such that there exists a hyperplane in $\Re^n$ that separates the inverse images $f^{-1}(0)$ and $f^{-1}(1)$ (i.e. the hyperplane separates the points on which the function is 1 from those on which it is 0). An example of a linearly separable function is any monotone disjunction: if $f(x_1,\ldots,x_n) = x_{i_1} \vee \cdots \vee x_{i_k}$, then the hyperplane $x_{i_1} + \cdots + x_{i_k} = 1/2$ is a separating hyperplane. For each attribute x_i there is an associated weight w_i. There are two parameters, θ which determines the threshold for predicting 1 (positive), and α which determines the adjustment made to the weight of an attribute that was partly responsible for a wrong prediction. The pseudo-code for Winnow is shown in Figure 6.

Winnow has been successfully applied to many learning problems. It has the very nice property that one can prove that the number of mistakes only grows logarithmically in the number of variables (and linearly in the number of relevant variables). For example, it can be shown that for the learning of monotone disjunctions of at most k literals, if Winnow is run with $\alpha > 1$ and $\theta \geq 1/\alpha$, then the total number of mistakes is at most $\alpha k(\log_\alpha \theta + 1) + n/\theta$. Observe that

Winnow need not have prior knowledge of k although the number of mistakes depends on k. [Littlestone, 1988]) showed how to optimally choose θ and α if an upperbound on k is known a priori. Also, while Winnow is designed to learn a linearly separable class, reductions (discussed in Section 5.1) can be used to apply Winnow to classes for which the positive and negative points are not linearly separable, e.g. k-DNF.

4.1.2 The Halving Algorithm and Weighted Majority Algorithm

We now discuss some of the key techniques for designing good on-line learning algorithms for the special case of concept learning (i.e. learning boolean-valued functions). If one momentarily ignores the issue of computation time, then the halving algorithm performs very well. It works as follows. Initially all concepts in the concept class $\mathcal{C}$ are candidates for the target concept. To make a prediction for instance x, the learner takes a majority vote based on all remaining candidates (breaking a tie arbitrarily). Then when the feedback is received, all concepts that disagree are removed from the set of candidates. It can be shown that at each step the number of candidates is reduced by a factor of at least 2. Thus, the number of prediction mistakes made by the halving algorithm is at most $\lg |\mathcal{C}|$.

Clearly the halving algorithm will perform poorly if the data is noisy. We now briefly discuss the weighted majority algorithm which is one of several multiplicative weight-update schemes for generalizing the halving algorithm to tolerate noise. Also, the weighted majority algorithm provides a simple and effective method for constructing a learning algorithm A that is provided with a pool of "experts", one of which is known to perform well, but A does not know which one. Associated with each expert is a weight that gives A's confidence in the accuracy of that expert. When asked to make a prediction, A predicts by combining the votes from its experts based on their associated weights. When an expert suggests the wrong prediction, A passes that information to the given expert and reduces its associated weight using a multiplicative weight-updating scheme. Namely, the weight associated with each expert that mispredicts is multiplied by some weight $0 \leq \beta < 1$. By selecting $\beta > 0$ this algorithm is robust against noise in the data. Figure 7 shows the weighted majority algorithm in more depth.

We now briefly discuss some learning problems in which weighted majority is applicable.

Weighted Majority Algorithm (WM)

Let w_i be the weight associated with algorithm A_i

For $1 \leq i \leq n$

Let $w_i = 1$

To make a prediction for an instance $x \in X$:

Let $\mathcal{A}_0$ be the set of algorithms that predicted 0

Let $\mathcal{A}_1$ be the set of algorithms that predicted 1

Let $q_0 = \sum_{A_i \in \mathcal{A}_0} w_i$

Let $q_1 = \sum_{A_i \in \mathcal{A}_1} w_i$

Predict 0 iff $q_0 \geq q_1$

If a mistake is made:

Let γ be the prediction

For $1 \leq i \leq n$

If A_i's prediction for x is γ

Let $w_i = w_i \cdot \beta$ (where $0 \leq \beta < 1$)

Inform A_i of the mistake

Figure 7: The weighted majority algorithm.

Suppose one knows that the correct prediction comes from some target concept selected from a known concept class. Then one can apply the weighted majority algorithm where each concept in the class is one of the algorithms in the pool. For such situations, the weighted majority algorithm is a robust generalization of the halving algorithm. (In fact, the halving algorithm corresponds to the special case where $\beta = 0$.) As another example, the weighted majority algorithm can often be applied to help in situations in which the prediction algorithm has a parameter that must be selected and the best choice for the parameter depends on the target. In such cases one can build the pool of algorithms by choosing various values for the parameter.

We now describe some of the results known about the performance of the weighted majority algorithm. If the best algorithm in the pool $\mathcal{A}$ makes at most m mistakes, then the worst case number of mistakes made by the weighted majority algorithm is $O(\log |\mathcal{A}| + m)$ where the constant hidden within the big-oh notation depends on β. Specifically, the number of mistakes made by the weighted majority algorithm is at most: $\frac{\log |\mathcal{A}| + m \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$ if one algorithm in $\mathcal{A}$ makes at most m mistakes, $\frac{\log \frac{|\mathcal{A}|}{k} + m \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$ if there exists a set of k algorithms in $\mathcal{A}$ such that each algorithm makes at most m mistakes, and $\frac{\log \frac{|\mathcal{A}|}{k} + \frac{m}{k} \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$ if the total number of mistakes made by a set of k algorithms in $\mathcal{A}$ is m.

When $|\mathcal{A}|$ is not polynomial, the weighted majority algorithm (when directly implemented), is not computationally feasible. Recently, [Maass and Warmuth, 1995] introduced what they call the virtual weight technique to implicitly maintain the exponentially large set of weights so that the time to compute a prediction and then update the "virtual" weights is polynomial. More specifically, the basic idea is to simulate Winnow by grouping concepts that "behave alike" on seen examples into blocks. For each block only one weight has to be computed and one constructs the blocks so that the number of concepts combined in each block as well as the weight for the block can be efficiently computed. While the number of blocks increases as new counterexamples are received, the total number of blocks is polynomial in the number of mistakes. Thus all predictions and updates can be performed in time polynomial in the number of blocks, which is in turn polynomial in the number of prediction mistakes of Winnow. Many variations of the basic weighted majority algorithm have also been studied. The results of [Cesa-Bianchi, *et al.*, 1993] demonstrate how to tune β as a function of an upper bound on the noise rate.

4.2 Query Learning Model

A very well-studied formal learning model is the membership and equivalence query model developed by Angluin [Angluin, 1988]. In this model (often called the exact learning model) the learner's goal is to learn exactly how an unknown (boolean) target function f, taken from some known concept class $\mathcal{C}$, classifies all instances from the domain. This goal is commonly referred to as exact identification. The learner has available two types of queries to find out about f: one is a **membership query**, in which the learner supplies an instance x from the domain and is told $f(x)$. The other query provided is an **equivalence query** in which the learner presents a candidate function h and either is told that $h \equiv f$ (in which case learning is complete), or else is given a **counterexample** x for which $h(x) \neq f(x)$. There is a very close relationship between this learning model and the on-line learning model (supplemented with membership queries) when applied to a classification problem. Using a standard transformation [Angluin, 1988, Littlestone, 1988], algorithms that use membership and equivalence queries can easily be converted to on-line learning algorithms that use membership queries. Under such a transformation the number of counterexamples provided to the learner in response to the learner's

equivalence queries directly corresponds to the number of mistakes made by the on-line algorithm.

In this model a number of interesting polynomial time algorithms are known for learning deterministic finite automata [Angluin, 1987], Horn sentences [Angluin, Frazier, and Pitt, 1992], read-once formulas [Angluin, Hellerstein, and Karpinski, 1993], read-twice DNF formulas [Aizenstein and Pitt, 1991], decision trees [Bshouty and Mansour, 1995], and many others. It is easily shown that membership queries alone are not sufficient for efficient learning of these classes, and Angluin has developed a technique of "approximate fingerprints" to show that equivalence queries alone are also not enough [Angluin, 1990]. (In both cases the arguments are information theoretic, and hold even when the computation time is unbounded.) The work of [Bshouty, Goldman, Hancock and Matar, 1996] extended Angluin's results to establish tight bounds on how many equivalence queries are required for a number of these classes. Maass and Turán studied upper and lower bounds on the number of equivalence queries required for learning (when computation time is unbounded), both with and without membership queries [Maass and Turán, 1992].

It is known that any class learnable exactly from equivalence queries can be learned in the PAC setting [Angluin, 1988]. At a high level the exact learning algorithm is transformed to a PAC algorithm by having the learner use random examples to "search" for a counterexample to the hypothesis of the exact learner. If a counterexample is found, it is given as a response to the equivalence query. Furthermore, if a sufficiently large sample was drawn and no counterexample was found then the hypothesis has error at most ϵ (with probability at least $1-\delta$). The converse does not hold [Blum, 1990]. That is, there are concept classes that are efficiently PAC learnable but cannot be efficiently learned in the exact model.

We now describe Angluin's algorithm for learning monotone DNF formulas (see Figure 8). A prime implicant t of a formula f is a satisfiable conjunction of literals such that t implies f but no proper subterm of t implies f. For example, $f = (a \wedge c) \vee (b \wedge \bar{c})$ has prime implicants $a \wedge c$, $b \wedge \bar{c}$, and $a \wedge b$. The number of prime implicants of a general DNF formula may be exponentially larger than the number of terms in the formula. However, for monotone DNF the number of prime implicants is no greater than the number of terms in the formula. The key to the analysis is to show that at each iteration, the term t is a new prime implicant of f, the target concept. Since

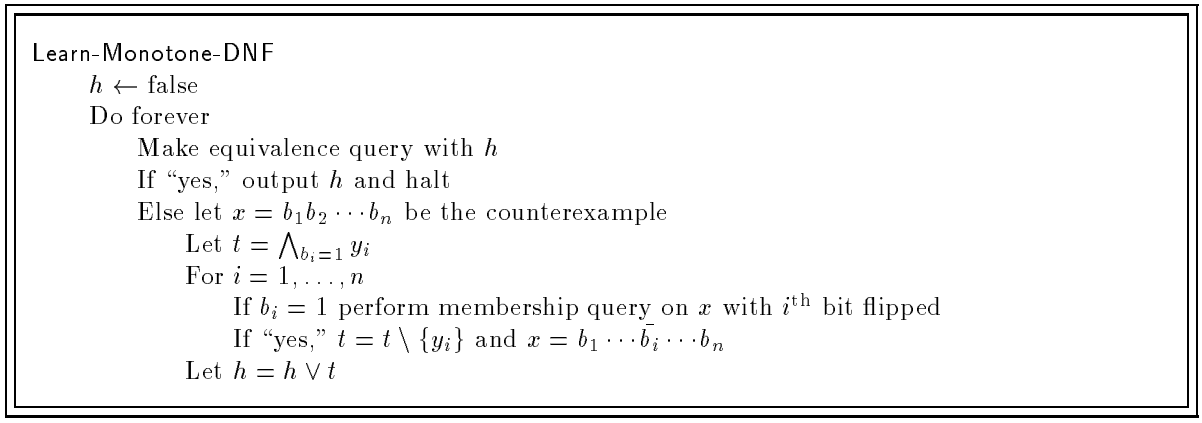

Figure 8: An algorithm that uses membership and equivalence queries to exactly learn an unknown monotone DNF formula over the domain $\{0,1\}^n$. Let $\{y_1, y_2, \ldots, y_n\}$ be the boolean variables and let h be the learner's hypothesis.

the loop iterates exactly once for each prime implicant there are at most m counterexamples where m is the number of terms in the target formula. Since at most n membership queries are performed during each iteration there are at most nm membership queries overall.

5 Hardness Results

In order to understand what concept classes are learnable, it is essential to develop techniques to prove when a learning problem is hard. Within the learning theory community, there are two basic type of hardness results that apply to all of the models discussed here. There are **representation-dependent** hardness results in which one proves that one cannot efficiently learn $\mathcal{C}$ using a hypothesis class of $\mathcal{H}$. These hardness results typically rely on some complexity theory assumption[8] such as RP $\neq$ NP. For example, given that RP $\neq$ NP, [Pitt and Warmuth, 1990] showed that k-term-DNF is not learnable using the hypothesis class of k-term-DNF. While such results provide some information, what one would really like to obtain is a hardness result for learning a concept class using any reasonable (i.e. polynomially evaluatable) hypothesis class. (For example, while we have a representation-dependent hardness result for learning k-

[8]For background on Complexity Classes see Chapter 33 (NP defined) and Chapter 35, Section 3 (RP defined) of this handbook.

term-DNF, there is a simple algorithm to PAC learn the class of k-term-DNF formulas using a hypothesis class of k-CNF formulas.) **Representation-independent** hardness results meet this more stringent goal. However, they depend on cryptographic (versus complexity theoretic) assumptions. [Kearns and Valiant, 1989] gave representation-independent hardness results for learning several concept classes such as Boolean formulas, deterministic finite automata, and constant-depth threshold circuits (a simplified form of "neural networks"). These hardness results are based on assumptions regarding the intractability of various cryptographic schemes such as factoring Blum integers and breaking the RSA function.

5.1 Prediction-Preserving Reductions

Given that we have some representation-independent hardness result (assuming the security of various cryptographic schemes) one would like a "simple" way to prove that other problems are hard in a similar fashion as one proves a desired algorithm is intractable by reducing a known NP-complete problem to it[9]. Such a complexity theory for predictability has been provided by [Pitt and Warmuth, 1990]. They formally define a prediction preserving reduction from concept class $\mathcal{C}$ over domain $\mathcal{X}$ to concept class $\mathcal{C}'$ over domain $\mathcal{X}'$ (denoted by $\mathcal{C} \leq \mathcal{C}'$) that allows an efficient learning algorithm for $\mathcal{C}'$ to be used to obtain any efficient learning algorithm for $\mathcal{C}$. The requirements for such a prediction-preserving reduction are: (1) an efficient instance transformation g from $\mathcal{X}$ to $\mathcal{X}'$, and (2) the existence of an image concept. The instance transformation g must be polynomial time computable. Hence if $g(x) = x'$ then the size of x' must be polynomially related to the size of x. So for $x \in \mathcal{X}_n$, $g(x) \in \mathcal{X}_{p(n)}$ where $p(n)$ is some polynomial function of n. It is also important that g be independent of the target function. We now define what is meant by the existence of an image concept. For every $f \in \mathcal{C}_n$ there must exist some $f' \in \mathcal{C}'_{p(n)}$ such that for all $x \in \mathcal{X}_n$, $f(x) = f'(g(x))$ and the number of bits to represent f' is polynomially related to the number of bits to represent f.

As an example, let $\mathcal{C}$ be the class of DNF formulas over $\mathcal{X} = \{0,1\}^n$, and $\mathcal{C}'$ be the class of monotone DNF formulas over $\mathcal{X}' = \{0,1\}^{2n}$. We now show that $\mathcal{C} \leq \mathcal{C}'$. Let $y_1, \ldots, y_n$ be

[9] See Chapter 34 of this handbook for background on reducibility and completeness.

the variables for each concept from $\mathcal{C}$. Let $y'_1, \ldots, y'_{2n}$ be the variables for $\mathcal{C}'$. The intuition behind the reduction is that variable y_i for the DNF problem is associated with variable y_{2i-1} for the monotone DNF problem. And variable $\overline{y_i}$ for the DNF problem is associated with variable y_{2i} for the monotone DNF problem. So for example $x = b_1 b_2 \cdots b_n$ where each $b_i \in \{0, 1\}$ and $g(x) = b_1(1 - b_1)b_2(1 - b_2) \cdots b_n(1 - b_n)$. Given a target concept $f \in \mathcal{C}$, the image concept f' is obtained by replacing each non-negated variable y_i from f with y'_{2i-1} and each negated variable $\overline{y_i}$ from f with y'_{2i}. It is easily confirmed that all required conditions are met.

If $\mathcal{C} \leq \mathcal{C}'$, what implications are there with respect to learnability? Observe that if we are given a polynomial prediction algorithm A' for $\mathcal{C}'$, one can use A' to obtain a polynomial prediction algorithm A for $\mathcal{C}$ as follows. If A' requests a labeled example then A can obtain a labeled example $x \in \mathcal{X}$ from its oracle and give $g(x)$ to A'. Finally, when A' outputs hypothesis h', A can make a prediction for $x \in \mathcal{X}$ using $h(g(x))$. Thus if $\mathcal{C}$ is known not to be learnable then $\mathcal{C}'$ also is not learnable. Thus the reduction given above implies that the class of monotone DNF formulas is just as hard to learn in the PAC model as the class of arbitrary DNF formulas. Equivalently, if there were an efficient PAC algorithm for the class of monotone DNF formulas, then there would also be an efficient PAC algorithm for arbitrary DNF formulas.

[Pitt and Warmuth, 1990] gave a prediction preserving reduction from the class of boolean formulas to class of DFAs. Thus since [Kearns and Valiant, 1989] proved that boolean formula are not predictable (under cryptographic assumptions), it immediately follows that DFAs are not predictable. In other words, DFAs cannot be efficiently learned from random examples alone. Recall that any algorithm to exactly learn using only equivalence queries can be converted into an efficient PAC algorithm. Thus if DFAs are not efficiently PAC learnable (under cryptographic assumptions), it immediately follows that DFAs are not efficiently learnable from only equivalence queries (under cryptographic assumptions). Contrasting this negative result, recall that DFAs are exactly learnable from membership and equivalence queries [Angluin, 1987], and thus are PAC learnable with membership queries.

Notice that for $\mathcal{C} \leq \mathcal{C}'$, the result that an efficient learning algorithm for $\mathcal{C}'$ also provides an efficient algorithm for $\mathcal{C}$ relies heavily on the fact that membership queries are NOT allowed. The

problem created by membership queries (whether in the PAC or exact model) is that algorithm A' for $\mathcal{C}'$ may make a membership query on an example $x' \in \mathcal{X}'$ for which $g^{-1}(x')$ is not in $\mathcal{X}$. For example, the reduction described earlier demonstrates that if there is an efficient algorithm to PAC learn monotone DNF formulas then there is an efficient algorithm to PAC learn DNF formulas. Notice that we already have an algorithm Learn-Monotone-DNF that exactly learns this class with membership and equivalence queries (and thus can PAC learn the class when provided with membership queries). Yet, the question of whether or not there is an algorithm with access to a membership oracle to PAC learn DNF formulas remains one of the biggest open questions within the field. We now describe the problem with using the algorithm Learn-Monotone-DNF to obtain an algorithm for general DNF formulas. Suppose that we have 4 boolean variables (thus the domain is $\{0,1\}^4$). The algorithm Learn-Monotone-DNF could perform a membership query on the example 00100111. While this example is in the domain $\{0,1\}^8$ there is no $x \in \{0,1\}^4$ for which $g(x) = 00100111$. Thus the provided membership oracle for the DNF problem cannot be used to respond to the membership query posed by Learn-Monotone-DNF.

We now define a more restricted type of reduction $\mathcal{C} \leq_{wmq} \mathcal{C}'$ that yields results even when membership queries are allowed. For these reductions, we just add the following third condition to the two conditions already described: for all $x' \in \mathcal{X}'$, if x' is not in the image of g (i.e. there is no $x \in \mathcal{X}$ such that $g(x) = x'$), then the classification of x' for the image concept must always be positive or always be negative. As an example, we show that $\mathcal{C} \leq_{wmq} \mathcal{C}'$ where $\mathcal{C}$ is the class of DNF formulas and $\mathcal{C}'$ is the class of read-thrice DNF formulas (meaning that each literal can appear at most three times). Thus learning a read-thrice DNF formula (even with membership queries) is as hard as learning general DNF formula (with membership queries). Let s be the number of literals in f. (If s is not known a priori the standard doubling technique can be applied.) The mapping g maps from an $x \in \{0,1\}^n$ to an $x' \in \{0,1\}^{sn}$. More specifically, $g(x)$ simply repeats, s times, each bit of x. To see that there is an image concept, note that we have s variables for each concept in $\mathcal{C}'$ associated with each variable for a concept in $\mathcal{C}$. Thus we can rewrite $f \in \mathcal{C}$ as a formula $f' \in \mathcal{C}'$ in which each variable only appears once. At this point we have shown $\mathcal{C} \leq \mathcal{C}'$ but still need to do more to satisfy the new third condition. We want to

ensure that the s variables (call them $\ell'_1, \ldots, \ell'_s$) for f' associated with one literal ℓ_i of f all take the same value. We do this by defining our final image concept as: $f' \wedge \Gamma_1 \wedge \cdots \wedge \Gamma_n$ where n is the number of variables and Γ_i is of the form $(\ell'_1 \rightarrow \ell'_2) \wedge \cdots \wedge (\ell'_{s-1} \rightarrow \ell'_s) \wedge (\ell'_s \rightarrow \ell'_1)$. This formula evaluates to true exactly when $\ell'_1, \ldots, \ell'_s$ have the same value. Thus an x' for which no $g(x) = x'$ will not satisfy some Γ_i and thus we can respond "no" to a membership query on x'. Finally, f' is a read-thrice DNF formula. This reduction also proves that boolean formulas $\leq_{wmq}$ read-thrice boolean formulas.

6 Weak Learning and Hypothesis Boosting

As originally defined, the PAC model requires the learner to produce, with arbitrarily high probability, a hypothesis that is arbitrarily close to the target concept. While for many problems it is easy to find simple algorithms ("rules-of-thumb") that are often correct, it seems much harder to find a single hypothesis that is highly accurate. Informally, a **weak learning** algorithm is one that outputs a hypothesis that has some advantage over random guessing. (To contrast this sometimes a PAC learning algorithm is called a strong learning algorithm.) This motivates the question: are there concept classes for which there is an efficient weak learner, but there is no efficient PAC learner? Somewhat surprisingly the answer is no [Schapire, 1990]. The technique used to prove this result is to take a weak learner and transform (boost) it into a PAC learner. The general method of converting a rough rule-of-thumb (weak learner) into a highly accurate prediction rule (PAC learner) is referred to as **hypothesis boosting**[10].

We now formally define the weak learning model [Kearns and Valiant, 1989, Schapire, 1990]. As in the PAC model, there is the instance space $\mathcal{X}$ and the concept class $\mathcal{C}$. Also the examples are drawn randomly and independently according to a fixed but unknown distribution $\mathcal{D}$ on $\mathcal{X}$. The learner's hypothesis h must be a polynomial time function that given an $x \in \mathcal{X}$ returns a prediction of $f(x)$ for $f \in \mathcal{C}$, the unknown target concept. The accuracy requirements for the

[10] We note that one can easily boost the confidence δ by first designing an algorithm A that works for say $\delta = 1/2$ and then running A several times taking a majority vote. For an arbitrary $\delta > 0$ the number of runs of A needed are polynomial in $\lg 1/\delta$.

hypothesis of a weak learner are as follows. There is a polynomial function $p(n)$ and algorithm A such that, for all $n \geq 1$, $f \in \mathcal{C}_n$, for all distributions $\mathcal{D}$, and for all $0 < \delta \leq 1$, algorithm A, given n, δ, and access to labeled examples from $\mathcal{D}$, outputs a hypothesis h such that, with probability $\geq 1 - \delta$, $error_D(h)$ is at most $(1/2 - 1/p(n))$. Algorithm A should run in time polynomial in n and $1/\delta$.

If $\mathcal{C}$ is strongly learnable, then it is weakly learnable—just fix $\epsilon = 1/4$ (or any constant less than $1/2$). The converse (weak learnability implying strong learnability) is not at all obvious. In fact, if one restricts the distributions under which the weak learning algorithm runs then weak learnability does not imply strong learnability. Namely, [Kearns and Valiant, 1989] have shown that under a uniform distribution, monotone boolean functions are weakly, but not strongly, learnable. Thus it is important to take advantage of the requirement that the weak learning algorithm must work for all distributions. Using this property, [Schapire, 1990] proved the converse result: if concept class $\mathcal{C}$ is weakly learnable, then it is strongly learnable.

Proving that weak learnability implies strong learnability has also been called the hypothesis boosting problem, because a way must be found to boost the accuracy of slightly-better-than-half hypotheses to be arbitrarily close to 1. There have been several boosting algorithms proposed since the original work of [Schapire, 1990]. Figure 9 describes AdaBoost [Freund and Schapire, 1996] that has shown promise in empirical use. The key to forcing the weak learner to output hypotheses that can be combined to create a highly accurate hypothesis is to create different distributions on which the weak learner is trained.

Freund and Schapire [Freund and Schapire, 1996] have done some experiments showing that by applying AdaBoost to some simple rules of thumb or using C4.5 [Quinlan, 1993] as the weak learner they can perform better than previous algorithms on some standard benchmarks. Also, Freund and Schapire have presented a more general version for AdaBoost for the situation in which the predictions are real-valued (versus binary). Some of the practical advantages of AdaBoost are that is is fast, simple and easy to program, requires no parameters to tune (besides T, the number of rounds), no prior knowledge is needed about the weak learner, it is provably effective, and it is flexible since you can combine it with any classifier that finds weak hypotheses.

AdaBoost

For $i = 1, \ldots, m$

$D_1(i) = 1/m$

For $t = 1, \ldots, T$

Call WeakLearn providing it with the distribution D_t to obtain the hypothesis h_t

Calculate the error of h_t : $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$

If $\epsilon_t > 1/2$, set $T = t - 1$ and abort the loop

Let $\beta_t = \epsilon_t/(1 - \epsilon_t)$

Update distribution D_t:

Let Z_t be a normalization constant so D_{t+1} is a valid probability distribution

If $h_t(x_i) = y_i$

Let $D_{t+1} = \frac{D_t(i)}{Z_t} \cdot \beta_t$

Else

Let $D_{t+1} = \frac{D_t(i)}{Z_t}$

Output the final hypothesis:

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$$

Figure 9: The procedure AdaBoost to boost the accuracy of a mediocre hypothesis (created by WeakLearn to a very accurate hypothesis. The input is a sequence $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ of labeled examples where each label comes from the set $Y = \{1, \ldots, k\}$.

7 Research Issues and Summary

In this chapter, we have described the fundamental models and techniques of computational learning theory. There are many interesting research directions besides those discussed here. One general direction of research is in defining new, more realistic models, and models that capture different learning scenarios. Here are a few examples. Often, as in the problem of weather prediction, the target concept is probabilistic in nature. Thus [Kearns and Schapire, 1990] introduced the p-concepts model. Also there has been a lot of work in extending the VC theory to real-valued domains (e.g. [Haussler, 1992]). In both the PAC and on-line models, many algorithms use membership queries. While most work has assumed that the answers provided to the membership queries are reliable, in reality a learner must be able to handle inconclusive and noisy results from the membership queries. See [Blum, Chalasani, Goldman, and Slonim, 1995] for a summary of several models introduced to address this situation. As one last example, there has been much work recently in exploring models of a "helpful teacher," since teaching is often used to assist human learning (e.g. [Goldman and Mathias, 1996, Angluin and Kriķis, 1997]).

Finally, there has been work to bridge the computational learning research with the research from other fields such as neural networks, natural language processing, DNA analysis, inductive logic programming, information retrieval, expert systems, and many others.

8 Defining Terms

Classification Noise A model of noise in which the the label may be reported incorrectly. In the random classification noise model, with probability η the label is inverted. In the malicious classification noise model, with probability η an adversary can choose the label. In both models with probability $1 - \eta$ the example is not corrupted. The quantity η is referred to as the noise rate.

Concept Class A set of rules from which the target function is selected.

Counterexample A counterexample x to a hypothesis h (where the target concept is f) is either an example for which $f(x) = 1$ and $h(x) = 0$ (a positive counterexample) or for

which $f(x) = 0$ and $h(x) = 1$ (a negative counterexample).

Equivalence Query A query to an oracle in which the learner provides a hypothesis h and is either told that h is logically equivalent to the target or otherwise given a counterexample.

Hypothesis Boosting The process of taking a weak learner that predicts correctly just over half of the time and transforming (boosting) it into a PAC (strong) learner whose predictions are as accurate as desired.

Hypothesis Class The class of functions from which the learner's hypothesis is selected.

Membership Query The learner supplies the membership oracle with an instance x from the domain and is told the value of $f(x)$.

Occam Algorithm An algorithm that draws a sample S and then outputs a hypothesis consistent with the examples in S such that the size of the hypothesis is sublinear in S (i.e. it performs some data compression).

On-Line Learning Model The learning session consists of a set of trials where in each trial, the learner is given an unlabeled instance $x \in \mathcal{X}$. The learner uses its current hypothesis to predict a value $p(x)$ for the unknown (real-valued) target and is then told the correct value for $f(x)$. The performance of the learner is measured in terms of the sum of the loss over all predictions.

PAC Learning This is a batch model in which first there is a training phase in which the learner sees examples drawn randomly from an unknown distribution $\mathcal{D}$, and labeled according to the unknown target concept f drawn from a known concept class $\mathcal{C}$. The learner's goal is to output a hypothesis that has error at most ϵ with probability at least $1-\delta$. (The learner receives ϵ and δ as inputs.) In the proper PAC learning model the learner must output a hypothesis from $\mathcal{C}$. In the non-proper PAC learning model the learner can output any hypothesis h for which $h(x)$ can be computed in polynomial time.

Representation-Dependent Hardness Result A hardness result in which one proves that one cannot efficiently learn $\mathcal{C}$ using a hypothesis class of $\mathcal{H}$. These hardness results typically

rely on some complexity theory assumption such as RP $\neq$ NP.

Representation-Independent Hardness Result A hardness result in which no assumption is made about the hypothesis class used by the learner (except the hypothesis can be evaluated in polynomial time). These hardness results typically rely on cryptographic assumptions such as the difficulty of breaking RSA.

Statistical Query Model A learning model in which the learner does not have access to labeled examples, but rather only can ask queries about statistics about the labeled examples and receive unlabeled examples. Any statistical query algorithm can be converted to a PAC algorithm that can handle random classification noise (as well as some other types of noise).

Vapnik-Chervonenkis (VC) Dimension A finite set $S \subseteq \mathcal{X}$ is shattered by $\mathcal{C}$ if for each of the $2^{|S|}$ subsets $S' \subseteq S$, there is a concept $f \in \mathcal{C}$ that contains all of S' and none of $S - S'$. In other words, for any of the $2^{|S|}$ possible labelings of S (where each example $s \in S$ is either positive or negative), there is some $f \in \mathcal{C}$ that realizes the desired labeling (see Figure 2). The Vapnik-Chervonenkis dimension of $\mathcal{C}$, denoted as $\text{VCD}(\mathcal{C})$, is the smallest d for which no set of $d+1$ examples is shattered by $\mathcal{C}$. Equivalently, $\text{VCD}(\mathcal{C})$ is the cardinality of the largest finite set of points $S \subseteq X$ that is shattered by $\mathcal{C}$.

Weak Learning Model A variant of the PAC model in which the learner is only required to output a hypothesis that with probability $\geq 1 - \delta$, has error at most $(1/2 - 1/p(n))$. I.e., it does noticeably better than random guessing. Recall in the standard PAC (strong learning) model the learner must output a hypothesis with arbitrarily low error.

References

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

[Angluin, 1987] Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Inform. Comput. 75*, 2 (Nov.), 87–106.

[Angluin, 1988] Angluin, D. 1988. Queries and concept learning. *Machine Learning 2*, 4 (April), 319–342.

[Angluin, 1990] Angluin, D. 1990. Negative results for equivalence queries. *Machine Learning 5*, 121–150.

[Angluin, Frazier, and Pitt, 1992] Angluin, D., Frazier, M., and Pitt, L. 1992. Learning conjunctions of Horn clauses. *Machine Learning 9*, 147–164.

[Angluin, Hellerstein, and Karpinski, 1993] Angluin, D., Hellerstein, L., and Karpinski, M. 1993. Learning read-once formulas with queries. *J. ACM 40*, 185–210.

[Angluin and Krikis, 1997] Angluin, D., Krikis, M. 1997. Teachers, learners and black boxes. In *Proc. 10th Annu. Conf. on Comput. Learning Theory* (1997). ACM Press, New York, NY, pp. 285–297.

[Angluin and Laird, 1988] Angluin, D. and Laird, P. 1988. Learning from noisy examples. *Machine Learning 2*, 4, 343–370.

[Angluin and Smith, 1987] Angluin, D. and Smith, C. 1987. Inductive inference. In *Encyclopedia of Artificial Intelligence*, pp. 409–418. J. Wiley and Sons, New York.

[Anthony and Biggs, 1992] Anthony, M. and Biggs, N. 1992. *Computational Learning Theory, Cambridge Tracts in Theoretical Computer Science (30).* Cambridge Tracts in Theoretical Computer Science (30). Cambridge University Press.

[Aslam and Decatur, 1997] Aslam, J. A. and Decatur, S. E. 1997. Specification and simulation of statistical query algorithms for efficiency and noise tolerance. *Journal of Computer and System Sciences*. To appear.

[Blum, 1990] Blum, A. 1990. Separating distribution-free and mistake-bound learning models over the Boolean domain. In *Proc. of the 31st Symposium on the Foundations of Comp. Sci.* (1990). IEEE Computer Society Press, Los Alamitos, CA, pp. 211–218.

[Blum, Chalasani, Goldman, and Slonim, 1995] Blum, A., Chalasani, P., Goldman, S. A., and Slonim, D. K. 1995. Learning with unreliable boundary queries. In *Proc. 8th Annu. Conf. on Comput. Learning Theory* (1995). ACM Press, New York, NY, pp. 98–107.

[Blumer, Ehrenfeucht, Haussler, and Warmuth, 1987] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. 1987. Occam's razor. *Inform. Proc. Lett. 24*, 377–380.

[Blumer, Ehrenfeucht, Haussler, and Warmuth, 1989] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. 1989. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM 36*, 4, 929–965.

[Bshouty, Goldman, Hancock, and Matar, 1996] Bshouty, N., Goldman, S., Hancock, T., and Matar, S. 1996. Asking questions to minimize errors. *Journal of Computer and System Sciences 52*, 2 (April), 268–286.

[Bshouty and Mansour, 1995] Bshouty, N. H. and Mansour, Y. 1995. Simple learning algorithms for decision trees and multivariate polynomials. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science* (1995). IEEE Computer Society Press, Los Alamitos, CA, pp. 304–311.

[Cesa-Bianchi, Freund, Helmbold, Haussler, Schapire, and Warmuth, 1993] Cesa-Bianchi, N., Freund, Y., Helmbold, D. P., Haussler, D., Schapire, R. E., and Warmuth, M. K. 1993. How to use expert advice. In *Proc. 25th Annu. ACM Sympos. Theory Comput.* (1993). ACM Press, New York, NY, pp. 382–391. Expanded version in Univ. of Calif. Computer Research Lab TR UCSC-CRL-94-33, From Santa Cruz, CA.

[Devroye, Györfi, and Lugosi, 1996] Devroye, L., Györfi, L., and Lugosi, G. 1996. *A Probabalistic Theory of Pattern Recognition, Applications of Mathematics.* Applications of Mathematics. Springer-Verlag, New York.

[Duda and Hart, 1973] DUDA, R. O. AND HART, P. E. 1973. *Pattern Classification and Scene Analysis*. Wiley.

[Ehrenfeucht and Haussler, 1989] EHRENFEUCHT, A. AND HAUSSLER, D. 1989. A general lower bound on the number of examples needed for learning. *Inform. Comput. 82*, 3 (Sept.), 247–251.

[Freund and Schapire, 1996] FREUND, Y. AND SCHAPIRE, R. 1996. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning* (1996). Morgan Kaufmann, pp. 148–156.

[Gold, 1967] GOLD, E. M. 1967. Language identification in the limit. *Inform. Control 10*, 447–474.

[Goldman and Mathias, 1996] GOLDMAN, S. AND MATHIAS, D. 1996. Teaching a smarter learner. *J. of Comput. Syst. Sci. 52*, 2 (April), 255–267.

[Goldman and Sloan, 1995] GOLDMAN, S. AND SLOAN, R. 1995. Can PAC learning algorithms tolerate random attribute noise? *Algorithmica 14*, 1 (July), 70–84.

[Goldman and Scott, 1996] GOLDMAN, S. A. AND SCOTT, S. D. 1996. A theoretical and empirical study of a noise-tolerant algorithm to learn geometric patterns. In *Proceedings of the Thirteenth International Conference on Machine Learning* (1996). Morgan Kaufmann, pp. 191–199.

[Haussler, 1992] HAUSSLER, D. 1992. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Inform. Comput. 100*, 1 (Sept.), 78–150.

[Haussler, Kearns, Littlestone, and Warmuth, 1991] HAUSSLER, D., KEARNS, M., LITTLESTONE, N., AND WARMUTH, M. K. 1991. Equivalence of models for polynomial learnability. *Inform. Comput. 95*, 2 (Dec.), 129–161.

[Haussler, Littlestone, and Warmuth, 1994] HAUSSLER, D., LITTLESTONE, N., AND WARMUTH, M. K. 1994. Predicting $\{0,1\}$ functions on randomly drawn points. *Inform. Comput. 115*, 2, 284–293.

[Kearns, 1993] Kearns, M. 1993. Efficient noise-tolerant learning from statistical queries. In *Proc. 25th Annu. ACM Sympos. Theory Comput.* (1993). ACM Press, New York, NY, pp. 392–401.

[Kearns and Valiant, 1989] Kearns, M. and Valiant, L. G. 1989. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proc. of the 21st Symposium on Theory of Computing* (1989). ACM Press, New York, NY, pp. 433–444.

[Kearns and Vazirani, 1994] Kearns, M. and Vazirani, U. 1994. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, Massachusetts.

[Kearns and Schapire, 1990] Kearns, M. J. and Schapire, R. E. 1990. Efficient distribution-free learning of probabilistic concepts. In *Proc. of the 31st Symposium on the Foundations of Comp. Sci.* (1990). IEEE Computer Society Press, Los Alamitos, CA, pp. 382–391.

[Kearns, Schapire, and Sellie, 1994] Kearns, M. J., Schapire, R. E., and Sellie, L. M. 1994. Toward efficient agnostic learning. *Machine Learning 17*, 2/3, 115–142.

[Kodratoff and Michalski, 1990] Kodratoff, Y. and Michalski, R. S., Eds. 1990. *Machine Learning: An Artificial Intelligence Approach*, vol. III. Morgan Kaufmann, Los Altos, California.

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

[Maass and Turán, 1992] Maass, W. and Turán, G. 1992. Lower bound methods and separation results for on-line learning models. *Machine Learning 9*, 107–145.

[Maass and Warmuth, 1995] Maass, W. and Warmuth, M. K. 1995. Efficient learning with virtual threshold gates. In *Proc. 12th International Conference on Machine Learning* (1995). Morgan Kaufmann, pp. 378–386.

[Natarajan, 1991] Natarajan, B. K. 1991. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann, San Mateo, CA.

[Pitt and Valiant, 1988] PITT, L. AND VALIANT, L. 1988. Computational limitations on learning from examples. *J. ACM 35*, 965–984.

[Pitt and Warmuth, 1990] PITT, L. AND WARMUTH, M. K. 1990. Prediction preserving reducibility. *J. of Comput. Syst. Sci. 41*, 3 (Dec.), 430–467. Special issue of the for the *Third Annual Conference of Structure in Complexity Theory* (Washington, DC., June 88).

[Quinlan, 1993] QUINLAN, J. R. 1993. *C4.5: Programs for machine learning*. Morgan Kaufmann.

[Rosenblatt, 1958] ROSENBLATT, F. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev. 65*, 386–407. (Reprinted in *Neurocomputing* (MIT Press, 1988).).

[Schapire, 1990] SCHAPIRE, R. E. 1990. The strength of weak learnability. *Machine Learning 5*, 2, 197–227.

[Sloan, 1988] SLOAN, R. 1988. Types of noise in data for concept learning. In *Proc. 1st Annu. Workshop on Comput. Learning Theory* (San Mateo, CA, 1988). Morgan Kaufmann, pp. 91–96.

[Valiant, 1984] VALIANT, L. G. 1984. A theory of the learnable. *Commun. ACM 27*, 11 (Nov.), 1134–1142.

[Valiant, 1985] VALIANT, L. G. 1985. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence, vol. 1* (Los Angeles, California, 1985). International Joint Committee for Artificial Intelligence, pp. 560–566.

[Vapnik and Chervonenkis, 1971] VAPNIK, V. N. AND CHERVONENKIS, A. Y. 1971. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications 16*, 2, 264–280.

9 Further Information

Good introductions to computational learning theory (along with pointers to relevant literature) can be found in such textbooks as [Kearns and Vazirani, 1994],[Natarajan, 1991], and [Anthony and Biggs, 1992]. Many recent results can be found in the proceedings from the following conferences: ACM Conference on Computational Learning Theory (COLT), European Conference on Computational Learning Theory (EuroCOLT), International Workshop on Algorithmic Learning Theory (ALT), International Conference on Machine Learning (ICML), IEEE Symposium on Foundations of Computer Science (FOCS), ACM Symposium on Theoretical Computing (STOC), and Neural Information Processing Conference (NIPS).

Some major journals in which many learning theory papers can be found are: *Machine Learning*, *Journal of the ACM*, *SIAM Journal of Computing*, *Information and Computation*, and *Journal of Computer and System Sciences*. See [Kodratoff and Michalski, 1990] for background information on machine learning, and see [Angluin and Smith, 1987] for a discussion of inductive inference models and research.