Computational Learning Theory

Lecture Notes for CS 582

Spring Semester, 1991

Sally A. Goldman
Department of Computer Science
Washington University
St. Louis, Missouri 63130

WUCS-91-36

Preface

This manuscript is a compliation of lecture notes from the graduate level course CS 582, "Computational Learning Theory," I taught at Washington University in the spring of 1991. Students taking the course were assumed to have background in the design and analysis of algorithms as well as good mathematical background. Given that there is no text available on this subject, the course material was drawn from recent research papers. I selected the first twelve topics and the remainder were selected by the students from a list of provided topics. This list of topics is given at the end of these notes.

These notes were mostly written by the students in the class and then reviewed by me. However, there are likely to be errors and omissions, particularly with regard to the references. Readers finding errors in the notes are encouraged to notify me by electronic mail (`sg@cs.wustl.edu`) so that later versions may be corrected.

Acknowledgements

This compilation of notes would not have been possible without the hard work of the following students: Andy Fingerhut, Nilesh Jain, Gadi Pinkas, Kevin Ruland, Marc Wallace, Ellen Witte, and Weilan Wu.

I thank Mike Kearns and Umesh Vazirani for providing me with a draft of the scribe notes from their Computational Learning Theory course taught at University of California at Berkeley in the Fall of 1990. Some of my lectures were prepared using their notes. Also most of the homework problems which I gave came from the problems used by Ron Rivest for his Machine Learning course at MIT taught during the Falls of 1989 and 1990. I thank Ron for allowing me to include these problems here.

Contents

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 1: Introduction

Lecturer: Sally Goldman *Scribe: Ellen Witte*

1.1 Course Overview

Building machines that learn from experience is an important research goal of artificial intelligence, and has therefore been an active area of research. Most of the work in machine learning is empirical research. In such research, learning algorithms typically are judged by their performance on sample data sets. Although these *ad hoc* comparisons may provide some insight, it is difficult to compare two learning algorithms carefully and rigorously, or to understand in what situations a given algorithm might perform well, without a formally specified learning model with which the algorithms may be evaluated.

Recently, considerable research attention has been devoted to the theoretical study of machine learning. In *computational learning theory* one defines formal mathematical models of learning that enable rigorous analysis of both the predictive power and the computational efficiency of learning algorithms. The analysis made possible by these models provides a framework in which to design algorithms that are provably more efficient in both their use of time and data.

During the first half of this course we will cover the basic results in computational learning theory. This portion will include a discussion of the distribution-free (or PAC) learning model, the model of learning with queries, and the mistake-bound (or on-line) learning model. The primary goal is to understand how these models relate to one another and what classes of concepts are efficiently learnable in the various models. Thus we will present efficient algorithms for learning various concept classes under each model. (And in some cases we will consider what can be done if the computation time is not restricted to be polynomial.) In contrast to these positive results we present hardness results for some concept classes indicating that no efficient learning algorithm exists. In addition to studying the basic noise-free versions of these learning models, we will also discuss various models of noise and techniques for designing algorithms that are robust against noise. Finally, during the second half of this course we will study a selection of topics that follow up on the material presented during the first half of the course. These topics were selected by the students, and are just a sample of the types of other results that have been obtained. We warn the reader that this course only covers a small portion of the models, learning techniques, and methods for proving hardness results that are currently available in the literature.

1.2 Introduction

In this section we give a very basic overview of the area of computational learning theory. Portions of this introduction are taken from Chapter 2 of Goldman's thesis [18]. Also see Chapter 2 of Kearns' thesis [27] for additional definitions and background material.

1.2.1 A Research Methodology

Before describing formal models of learning, it is useful to outline a research methodology for applying the formalism of computational learning theory to "real-life" learning problems. There are four steps to the methodology.

1. Precisely define the problem, preserving key features while simplifying as much as possible.

2. Select an appropriate formal learning model.

3. Design a learning algorithm.

4. Analyze the performance of the algorithm using the formal model.

In Step 2, selecting an appropriate formal learning model, there are a number of questions to consider. These include:

- What is being learned?
- How does the learner interact with the environment? (e.g., Is there a helpful teacher? An adversary?)
- What is the prior knowledge of the learner?
- How is the learner's hypothesis represented?
- What are the criteria for successful learning?
- How efficient is the learner in time, data and space?

It is critical that the model chosen accurately reflect the real-life learning problem.

1.2.2 Definitions

In this course, we consider a restricted type of learning problem called *concept learning*. In a concept learning problem there are a set of *instances* and a single *target concept* that classifies each instance as a positive or a negative instance. The *instance space* denotes the set of all instances that the learner may see. The *concept space* denotes the set of all concepts from which the target concept can be chosen. The learner's goal is to devise a hypothesis of the target concept that accurately classifies each instance as positive or negative.

For example, one might wish to teach a child how to distinguish chairs from other furniture in a room. Each item of furniture is an instance; the chairs are positive instances and all other items are negative instances. The goal of the learner (in this case the child) is to develop a rule for the concept of a chair. In our models of learning, one possibility is that the rules are Boolean functions of features of the items presented, such as has-four-legs or is-wooden.

Since we want the complexity of the learning problem to depend on the "size" of the target concept, often we assign to it some natural size measure n. If we let X_n denote the set of instances to be classified for each problem of size n, we say that $X = \bigcup_{n \geq 1} X_n$ is the *instance space*, and each $x \in X$ is an *instance*. For each $n \geq 1$, we define each $C_n \subseteq 2^{X_n}$ to be a *family of concepts* over X_n, and $C = \bigcup_{n \geq 1} C_n$ to be a *concept class* over X. For $c \in C_n$ and $x \in X_n$, $c(x)$ denotes the classification of c on instance x. That is, $c(x) = 1$ if and only if $x \in c$. We say that x is a positive instance of c if $c(x) = 1$ and x is a negative instance of c if $c(x) = 0$. Finally, a *hypothesis* h for C_n is a rule that given any $x \in X_n$ outputs in polynomial time a prediction for $c(x)$. The *hypothesis space* is the set of all hypotheses h that the learning algorithm may output. The hypothesis must make a prediction for each $x \in X_n$. For the learning models which we will study here, it is not acceptable for the hypothesis to answer "I don't know" for some instances.

To illustrate these definitions, consider the concept class of monomials. (A monomial is a conjunction of literals, where each literal is either some boolean variable or its negation.) For this concept class, n is the number of variables. Thus $|X_n| = 2^n$ where each $x \in X_n$ represents an assignment of 0 or 1 to each variable. Observe that each variable can be placed in the target concept unnegated, placed in the target concept negated, or not put in the target concept at all. Thus, $|C_n| = 3^n$. One possible target concept for the class of monomials over five variables is $x_1\overline{x_4}x_5$. For this target concept, the instance "10001" is a positive instance and "00001" is a negative instance.

We will model the learning process as consisting of two phases, a training phase and a performance phase. In the training phase the learner is presented with labeled examples (i.e., an example is chosen from the instance space and labeled according to the target concept). Based on these examples the learner must devise a hypothesis of the target concept. In the performance phase the hypothesis is used to classify examples from the instance space, and the accuracy of the hypothesis is evaluated. The various models of learning will differ primarily in the way they allow the learner to interact with the environment during the training phase and how the hypothesis is evaluated.

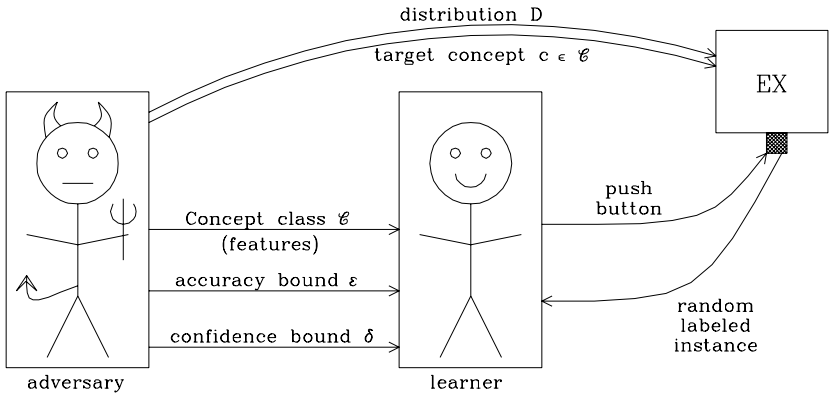

Figure 1.1: The PAC learning model.

1.3 The Distribution-Free (PAC) Model

The first formal model of machine learning we shall consider is the distribution-free or PAC model introduced by Valiant [43] in 1984. (This work initiated the field of computational learning theory.) In this model, an adversary chooses a concept class C, a target concept $c \in C$ and an arbitrary distribution D on the instance space. (We note that absolutely *no* restrictions are placed on D.) The learner is presented with the concept class C, an accuracy bound ϵ and a confidence bound δ. The learner is required to formulate a hypothesis h of the target concept based on labeled examples drawn randomly from the distribution D (which is unknown to the learner). See Figure 1.1.[1]

The PAC model requires the learner to produce a hypothesis which meets a certain error criteria. We can define the *error of the hypothesis* h on target concept c under distribution D to be:

$$error_D(h) = \Pr[c \oplus h] = \sum_{c(x) \neq h(x)} \Pr[x]$$

where $c \oplus h$ is the symmetric difference between c and h (i.e., the instances for which c and h differ). The error is the sum of the weight under distribution D placed on the examples for which c and h differ.

The goals of the learner are as follows:

1. With high probability ($\geq 1 - \delta$) the hypothesis must give a good approximation ($error_D(h) \leq \epsilon$) of the target concept.

2. The time and sample complexity of the learning algorithm must be polynomial in the size of the target concept, $1/\epsilon$ and $1/\delta$. (The sample complexity is the number of labeled examples needed by the algorithm.) Observe that as ϵ and δ go down, the algorithm is allowed more time and samples to produce a hypothesis.

[1] Compliments of Andy Fingerhut.

This model is called distribution-free because the distribution on the examples is unknown to the learner. Because the hypothesis must have high **p**robability of being **a**pproximately **c**orrect, it is also called the PAC model.

1.4 Learning Monomials in the PAC Model

We would like to investigate various concept classes that can be learned efficiently in the PAC model. The concept class of monomials is one of the simplest to learn and analyze in this model. Furthermore, we will see that the algorithm for learning monomials can be used to learn more complicated concept classes such as k-CNF.

A monomial is a conjunction of literals, where each literal is a variable or its negation. In describing the algorithm for learning monomials we will assume that n, the number of variables, is known. (If not, then it can be determined simply by counting the number of bits in the first example.)

We now describe the algorithm given by Valiant [43] for learning monomials. The algorithm is based on the idea that a positive example gives significant information about the monomial being learned. For example, if $n = 5$, and we see a positive example "10001", then we know that the monomial does *not* contain $\overline{x_1}$, x_2, x_3, x_4 or $\overline{x_5}$. A negative example does not give us near as much information since we do not know which of the bits caused the example to violate the target monomial.

Algorithm Learn-Monomials(n,ϵ,δ)

1. Initialize the hypothesis to the conjunction of all $2n$ literals.

$$h = x_1\overline{x_1}x_2\overline{x_2}\cdots x_n\overline{x_n}$$

2. Make $m = 1/\epsilon(n \ln 3 + \ln 1/\delta)$ calls to EX.

   - For each positive instance, remove x_i from h if $x_i = 0$ and remove $\overline{x_i}$ from h if $x_i = 1$.
   - For each negative instance, do nothing.

3. Output the remaining hypothesis.

In analyzing the algorithm, there are three measures of concern. First, is the number of examples used by the algorithm polynomial in n, $1/\epsilon$ and $1/\delta$? The algorithm uses $m = 1/\epsilon(n \ln 3 + \ln 1/\delta)$ examples; clearly this is polynomial in n, $1/\epsilon$ and $1/\delta$. Second, does the algorithm take time polynomial in these three parameters? The time taken per example is constant, so the answer is yes. Third is the hypothesis sufficiently accurate by the criteria of the PAC model? In other words, is $\Pr[error_D(h) \leq \epsilon] \geq (1 - \delta)$?

To answer the third question we first show that the final hypothesis output by the algorithm is consistent with all m examples seen in training. That is, the hypothesis correctly

classifies all of these examples. We will also show that the hypothesis logically implies the target concept. This means that the hypothesis does not classify any negative example as positive. We can prove this by induction on the number of examples seen so far. We initialize the hypothesis to the conjunction of all $2n$ literals, which is logically equivalent to classifying every example as false. This is trivially consistent with all examples seen, since initially no examples have been seen. Also, it trivially implies the target concept since false implies anything. Let h_i be the hypothesis after i examples. Assuming the hypothesis h_k is consistent with the first k examples and implies the target concept, we show the hypothesis h_{k+1} is consistent with the first $k+1$ examples and still implies the target concept. If example $k+1$ is negative, $h_{k+1} = h_k$. Since h_k implies the target concept, it does not classify any negative example as positive. Therefore h_{k+1} correctly classifies the first $k+1$ examples and implies the target concept. If example $k+1$ is positive, we alter h_k to include this example within those classified as positive. Clearly h_{k+1} correctly classifies the first $k+1$ examples. In addition, it is not possible for some negative example to satisfy h_{k+1}, so this new hypothesis logically implies the target concept.

To analyze the error of the hypothesis, we define an ϵ-*bad* hypothesis h' as one with $error(h') > \epsilon$. We satisfy the PAC error criteria if the final hypothesis (which we know is consistent with all m examples seen in training) is not ϵ-bad. By the definition of ϵ-bad,

$$\Pr[\text{an } \epsilon\text{-bad hyp is consistent with 1 ex}] \leq 1 - \epsilon$$

and since each example is taken independently,

$$\Pr[\text{an } \epsilon\text{-bad hyp is consistent with m exs}] \leq (1-\epsilon)^m.$$

Since the hypothesis comes from C, the maximum number of hypotheses is $|C|$. Thus,

$$\Pr[\exists \text{ an } \epsilon\text{-bad hyp consistent with m exs}] \leq |C|(1-\epsilon)^m.$$

Now, we require that $\Pr[h \text{ is } \epsilon\text{-bad}] \leq \delta$, so we must choose m to satisfy

$$|C|(1-\epsilon)^m \leq \delta.$$

Solving for m,

$$m \geq \frac{\ln |C| + \ln 1/\delta}{-\ln(1-\epsilon)}.$$

Using the Taylor series expansion for e^x

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots > 1 + x,$$

letting $x = -\epsilon$ and taking ln of both sides we can infer that $\epsilon < -\ln(1-\epsilon)$. Also, $|C| = 3^n$ since each of the variables may appear in the monomial negated, unnegate, or not appear at all. So if $m \geq 1/\epsilon(n \ln 3 + \ln 1/\delta)$ then $\Pr[error(h) > \epsilon] \leq \delta$.

It is interesting to note that only $O(n \ln 3)$ examples are required by the algorithm (ignoring dependence on ϵ and δ) even though there are 2^n examples to classify. Also, the analysis of the number of examples required can be applied to any algorithm which finds a hypothesis consistent with all the examples seen during training. Observe, this is only an upper bound. In some cases a tighter bound on the number of examples may be achievable.

1.5 Learning k-CNF and k-DNF

We now describe how to extend this result for monomials to the more expressive classes of k-CNF and k-DNF. The class *k-Conjunctive Normal Form*, denoted $k\text{-CNF}_n$ consists of all Boolean formulas of the form $C_1 \wedge C_2 \wedge \ldots \wedge C_l$ where each clause C_i is the disjunction of at most k literals over $x_1, \ldots, x_n$. We assume k is some constant. We will often omit the subscript n. If $k = n$ then the class k-CNF consists of all CNF formulas. The class of monomials is equivalent to 1-CNF.

The class *k-Disjunctive Normal Form*, denoted $k\text{-DNF}_n$ consists of all Boolean formulas of the form $T_1 \vee T_2 \vee \ldots \vee T_l$ where each term T_i is the conjunction of at most k literals over $x_1, \ldots, x_n$.

There is a close relationship between k-CNF and k-DNF. By DeMorgan's law

$$f = C_1 \wedge C_2 \wedge \ldots \wedge C_l \Rightarrow \overline{f} = T_1 \vee T_2 \ldots \vee T_l$$

where T_i is the conjunction of the negation of the literals in C_i. For example if

$$f = (a \vee \overline{b} \vee c) \wedge (\overline{d} \vee e)$$

then by DeMorgan's law

$$\overline{f} = \overline{(a \vee \overline{b} \vee c)} \vee \overline{(\overline{d} \vee e)}.$$

Finally, applying DeMorgan's law once more we get that

$$\overline{f} = (\overline{a} \wedge b \wedge \overline{c}) \vee (d \wedge \overline{e}).$$

Thus the classes k-CNF and k-DNF are duals of each other in the sense that exchanging $\wedge$ for $\vee$ and complementing each variable gives a transformation between the two representations. So an algorithm for k-CNF can be used for k-DNF (and vice versa) by swapping the use of positive and negative examples, and negating all the attributes.

We now describe a general technique for modifying the set of variables in the formula and apply this technique to generalize the monomial algorithm given in the previous section to learn k-CNF. The idea is that we define a new set of variables, one for each possible clause. Next we apply our monomial algorithm using this new variable set. To do this we must compute the value for each new variable given the value for the original variables. However, this is easily done by just evaluating the corresponding clause on the given input. Finally, it is straightforward to translate the hypothesis found by the monomial algorithm into a k-CNF formula using the correspondence between the variables and the clauses.

We now analyze the complexity of the above algorithm for learning k-CNF. Observe that the number of possible clauses is upper bounded by:

$$\sum_{i=1}^{k} \binom{2n}{k} = O(n^k).$$

This overcounts the number of clauses since it allows clauses containing both x_i and $\overline{x}_i$. The high order terms of the summation dominate. Since $\binom{2n}{k} < (2n)^k$, the number of clauses is $O(n^k)$ which is polynomial in n for any constant k. Thus the time and sample complexity of our k-CNF algorithm are polynomial.

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 2: Two-Button PAC Model

Lecturer: Sally Goldman *Scribe: Ellen Witte*

2.1 Model Definition

Here we consider a variant of the single button PAC model described in the previous notes. The material presented in this lecture is just one portion of the paper, "Equivalence of Models for Polynomial Learnability," by David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth [21]. Rather than pushing a single button to receive an example drawn randomly from the distribution D, a variation of this model has two buttons, one for positive examples and one for negative examples. (In fact, the this is the model originally introduced by Valiant.) There are two distributions, one on the positive examples and one on the negative examples. When the positive button is pushed a positive example is drawn randomly from the distribution D^+. When the negative button is pushed a negative example is drawn randomly from the distribution D^-. A learning algorithm in the two-button model is required to be accurate within the confidence bound for both the positive and negative distributions. Formally, we require

$$\Pr[error^+(h) \equiv \Pr_{D^+}(c \oplus h) \leq \epsilon] \geq 1 - \delta$$

$$\Pr[error^-(h) \equiv \Pr_{D^-}(c \oplus h) \leq \epsilon] \geq 1 - \delta.$$

Offering two buttons allows the learning algorithm to choose whether a positive or negative example will be seen next. It should be obvious that this gives the learner at least as much power as in the one button model. In fact, we will show that the one-button and two-button models are equivalent in the concept classes they can learn efficiently.

2.2 Equivalence of One-Button and Two-Button Models

In this section we show that one-button PAC-learnability is equivalent to two-button PAC-learnability.

Theorem 2.1 *One-button PAC-learnability (1BL) is equivalent to two-button PAC-learnability (2BL).*

Proof:
Case 1: 1BL $\Rightarrow$ 2BL.

Let $c \in C$ be a target concept over an instance space X that we can learn in the one-button model using algorithm A_1. We show that there is an algorithm A_2 for learning c in the two-button model. Let D^+ and D^- be the distributions over the positive and negative examples respectively. The algorithm A_2 is as follows:

1. Run A_1 with inputs n, $\epsilon/2$, δ.
2. When A_1 requires an example, flip a fair coin. If the outcome is heads, push the positive example button and give the resulting example to A_1. If the outcome is tails, push the negative example button and give the resulting example to A_1.
3. When A_1 terminates, return the hypothesis h found by A_1.

The use of the fair coin to determine whether to give A_1 a positive or negative example results in a distribution D seen by A_1 defined by,

$$D(x) = 1/2D^+(x) + 1/2D^-(x).$$

Let e be the error of h on D, e^+ be the error of h on D^+ and e^- be the error of h on D^-. Then

$$e = e^+/2 + e^-/2.$$

Since A_1 is an algorithm for PAC learning, it satisfies the error criteria. That is,

$$\Pr[e \leq \epsilon/2] \geq 1 - \delta.$$

Since $e \geq e^+/2$ and $e \geq e^-/2$ we can conclude that

$$\Pr[e^+/2 \leq \epsilon/2] \geq 1 - \delta$$

$$\Pr[e^-/2 \leq \epsilon/2] \geq 1 - \delta.$$

Therefore, $e^+ \leq \epsilon$ and $e^- \leq \epsilon$ each with probability $\geq 1 - \delta$, and so algorithm A_2 satisfies the accuracy criteria for the two-button model. Notice that we had to run A_1 with an error bound of $\epsilon/2$ in order to achieve an error bound of ϵ in A_2.

Case 2: 2BL $\Rightarrow$ 1BL.

Let $c \in C$ be a target concept over instance space X that we can learn in the two-button model using algorithm A_2. We show that there is an algorithm A_1 for learning c in the one-button model.

The idea behind the algorithm is that A_1 will draw some number of examples from D initially and store them in two bins, one bin for positive examples and one bin for negative examples. Then A_1 will run A_2. When A_2 requests an example, A_1 will supply one from the appropriate bin. Care will need to be exercised because there may not be enough of one type of example to run A_2 to completion. Let m be the total number of examples (of both types) needed to run A_2 to completion with parameters n, ϵ, $\delta/3$. Algorithm A_1 is as follows:

1. Make q calls to EX and store the positive examples in one bin and the negative examples in another bin, where
$$q = \max\left\{\frac{2}{\epsilon}m, \frac{8}{\epsilon}\ln\frac{3}{\delta}\right\}.$$

2. If the number of positive examples is $< m$ then output the hypothesis false. (This hypothesis classifies all instances as negative.)

3. Else if the number of negative examples is $< m$ then output the hypothesis true. (This hypothesis classifies all instances as positive.)

4. Else there are enough positive and negative examples, so run A_2 to completion with parameters (n,min(ϵ, 1/2),δ/3). Output the hypothesis returned by A_2.

For now this choice of q seems to have appeared out of thin air. The rationale for the choice will become clear in the analysis of the algorithm. Before continuing the analysis, we need an aside to discuss Chernoff Bounds, which will be needed in the proof.

Aside: Chernoff Bounds

Chernoff Bounds are formulas which bound the area under the tails of the binomial distribution. We now describe some of the bounds cited in the literature. Much of this discussion is taken directly from Sloan [40]. Normally we are trying to say that if we run m Bernoulli trials each with probability of success p, then the chance of getting a number of successes very much different from pm is exponentially vanishing.

Formally, let $X_1, X_2, \ldots X_m$ be independent Boolean random variables each with probability of p ($0 \leq p \leq 1$) of being 1. We now define a random variable $S = \sum_{i=1}^{m} X_i$. Clearly the expectation of S is pm.

Define $LE(p, m, r) = \Pr[S \leq r]$ (i.e. the probability of at most r successes in m independent trials of a Bernoulli random variable with probability of success p). Let $GE(p, m, r) = \Pr[S \geq r]$ (i.e. the probability of at least r successes in m independent trials of a Bernoulli random variable with probability of success p). So $LE(p, m, r)$ bounds the area in the tail at the low end of the binomial distribution, while $GE(p, m, r)$ bounds the area in the tail at the high end of the binomial distribution.

Hoeffding's Inequality [22] states that:

$$\Pr[S \geq pm + t] \leq e^{-2mt^2} \tag{2.1}$$
$$\Pr[S \geq \alpha m], \Pr[S \leq \alpha m] \leq e^{-2m(\alpha - p)^2} \tag{2.2}$$

where it is understood that in Equation (2.2) the first α must be at least p and the second α must be at most p.

The above bound is as good or better than any of the others in the literature except for the case when $p < 1/4$. In this case the following bounds given by Angluin and Valiant [8] are better:

$$LE(p, m, (1-\alpha)pm) \leq e^{-\alpha^2 mp/2} \tag{2.3}$$

$$GE(p, m, (1+\alpha)pm) \leq e^{-\alpha^2 mp/3}. \tag{2.4}$$

where $0 \leq \alpha \leq 1$. Note that in Equation (2.3), $r = (1-\alpha)pm \leq pm$. And in Equation (2.4), $r = (1+\alpha)pm \geq pm$.

We now return to the analysis of algorithm A_1. Let p^+ denote the probability that we draw a positive example from D, and p^- denote the probability that we draw a negative example from D. To analyze the accuracy of algorithm A_1 we consider four cases based on the probabilities p^+ and p^-. For each case we will show that $\Pr[error(h) \leq \epsilon] \geq 1 - \delta$.

Subcase A: $p^+ \geq \epsilon$, $p^- \geq \epsilon$.

There are three things which may happen in algorithm A_1. First, we may not have enough of one type of example and thus will output a hypothesis of true or false. Notice that if this occurs then $\Pr[error_D(h) \leq \epsilon] = 0$ since $p^+, p^- \geq \epsilon$. Second, we may have enough examples to run A_2 to completion and receive an ϵ-bad hypothesis from A_2. Third, we may have enough examples to run A_2 to completion receive an ϵ-good hypothesis from A_2. The first two outcomes are undesirable. We must make sure they occur with probability at most δ.

Let us determine the probability that we do not have enough of one type of example. Consider the positive examples. The probability of drawing a positive example is $p^+ \geq \epsilon$. We are interested in the probability that we draw fewer than m positive examples in q calls to EX. This probability can be bounded by the Chernoff bound of Equation (2.3).

$$\Pr[< m \text{ positive exs in } q \text{ calls to EX}] \leq LE(\epsilon, q, m)$$

From Equation (2.3) we know that

$$LE(p, m', (1-\alpha)m'p) \leq e^{-\alpha^2 m' p/2}.$$

So, with $p = \epsilon$,$m' = q = \frac{2}{\epsilon}m$ and $\alpha = 1/2$ we obtain

$$LE(\epsilon, q, m) \leq e^{-m/4}.$$

Finally, we require that this bad even occurs with probability at most $\delta/3$. That is, we must have

$$e^{-m/4} \leq \delta/3.$$

Solving for m yields

$$m \geq 4 \ln \frac{3}{\delta} \Rightarrow q \geq \frac{8}{\epsilon} \ln \frac{3}{\delta}$$

which is satisfied by the choice of q in the algorithm.

The same analysis holds for bounding the probability that there are fewer than m negative examples in q calls to EX. That is, we know

$$\Pr[< m \text{ negative exs in } q \text{ calls to EX}] \leq \delta/3$$

$$\Pr[< m \text{ positive exs in } q \text{ calls to EX}] \leq \delta/3.$$

This implies that

$$\Pr[\geq m \text{ positive exs and } \geq m \text{ negative exs in } q \text{ calls to EX}] \geq 1 - 2\delta/3.$$

If we have enough positive and negative examples then we will run algorithm A_2 to completion. With probability $\geq 1 - \delta/3$ algorithm A_2 will return a hypothesis with $e^+ \leq \epsilon$ and $e^- \leq \epsilon$. This implies that the hypothesis h returned by A_2 satisfies:

$$\begin{aligned} error_D(h) &= p^+ e^+ + p^- e^- \\ &= p^+ e^+ + (1 - p^+) e^- \\ &\leq p^+ \epsilon + (1 - p^+) \epsilon \\ &= \epsilon \end{aligned}$$

We have determined the following probabilities of each of the two bad outcomes that can occur when running A_1.

- With probability $\leq 2\delta/3$ we do not have enough of one type of example to run A_2.
- The probability that we run A_2 and it returns a bad hypothesis is given by the product of the probability that we have enough examples and the probability A_2 returns a bad hypothesis. This probability is at most $\left(1 - \frac{2\delta}{3}\right) \frac{\delta}{3} \leq \delta/3$.

In all other cases, the hypothesis output will be ϵ-good. Combining these probabilities yields the following expression for the probability that the good outcome occurs.

$$\Pr[error_D(h) \leq \epsilon] \geq 1 - \left(\frac{2\delta}{3} + \frac{\delta}{3}\right) = 1 - \delta$$

Subcase B: $p^+ < \epsilon$, $p^- \geq \epsilon$.

In this case, we have a good chance of getting more than m negative examples and less than m positive examples. If this occurs algorithm A_1 will output a hypothesis of false. Since $p^+ < \epsilon$, a hypothesis h of false satisfies $\Pr[error_D(h) \leq \epsilon] = 1$. This is good. It is also possible (although less likely) that we will not get enough negative examples, or that we will get enough of each kind of example to run A_2. In running A_2 we may get a hypothesis which has error $> \epsilon$. We must ensure that these bad things occur with probability at most δ.

More formally, we consider two possible bad situations. First, we may draw less than m negative examples. In Subcase A we showed that this will occur with probability $\leq \delta/3$. In this situation we will draw enough positive examples and thus output a hypothesis h of true. For this hypothesis, $\Pr[error_D(h) > \epsilon] = 1$. In the second situation, we draw at least m negative examples. This occurs with probability $\geq 1 - \delta/3$. In the worst case we also draw at least m positive examples and thus run A_2. We know that the hypothesis h returned by A_2 will satisfy $\Pr[error_D(h) \leq \epsilon] \geq 1 - \delta/3$. This implies that $\Pr[error_D(h) > \epsilon] \leq \delta/3$. Combining these two results we have

$$\Pr[error_D(h) > \epsilon] \leq \delta/3 + \delta/3$$

which implies that

$$\Pr[error_D(h) \leq \epsilon] \geq 1 - 2\delta/3.$$

Subcase C: $p^- < \epsilon$, $p^+ \geq \epsilon$.

This case is the same as Subcase B but with the roles of positive and negative examples interchanged. By a similar analysis,

$$\Pr[error_D(h) \leq \epsilon] \geq 1 - 2\delta/3.$$

Subcase D: $p^+ < \epsilon$, $p^- < \epsilon$.

Since $\epsilon \leq 1/2$, this case cannot occur.

These four subcases cover the behavior of algorithm A_1 for all possible values of p^+ and p^-. Thus, 2BL $\Rightarrow$ 1BL. Taken with Case 1 proving 1BL $\Rightarrow$ 2BL we have shown that one-button PAC learnability is equivalent to two-button PAC learnability.

■

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 3: Learning k-term-DNF

Lecturer: Sally Goldman *Scribe: Weilan Wu*

3.1 Introduction

In this lecture, we shall discuss the learnability of k-term-DNF formulas. Most of the material presented in this lecture comes from the paper "Computational Limitations on Learning from Examples," by Leonard Pitt and Leslie Valiant [33].

When introducing the PAC model, we showed that both k-CNF and k-DNF are PAC learnable using a hypothesis space of k-DNF and k-CNF respectively. Does a similar result hold for k-term-DNF and k-clause-CNF? We first show that k-term-DNF is not PAC learnable by k-term-DNF in polynomial time, unless RP=NP. In fact, this hardness result holds even for learning monotone k-term-DNF by k-term-DNF. Likewise, k-clause-CNF is not PAC learnable by k-clause-CNF in both the monotone and unrestricted case. Contrasting these negative results, we then describe an efficient algorithm to learn k-term-DNF by k-CNF. As a dual result, we can learn k-clause-CNF using the hypothesis space of k-DNF. Finally, we explore the representational power of the concept classes that we have considered so far and the class of k-decision-lists [35].

Before describing the representation-dependent hardness result, we first give some basic definitions. The concept class of k-term-DNF is defined as follows:

Definition 3.1 *For any constant k, the class of k-term-DNF formulas contains all disjunctions of the form $T_1 \vee T_2 \vee \ldots \vee T_k$, where each T_i is monomial.*

Up to now we have assumed that in the PAC model the hypothesis space available to the learner is equivalent to the concept class. That is, each element of the hypothesis space corresponds to the representation of an element of the concept space. However, in general one can talk about a concept class C being PAC learnable by H (possibly different from C). Formally, we say that C is PAC learnable by H, if there exists a polynomial-time learning algorithm A such that for any $c \in C$, any distribution D and any ϵ, δ, A can output with probability at least $1 - \delta$ a hypothesis $h \in H$ such that h has probability at most ϵ of disagreeing with c on a randomly drawn instance from D.

3.2 Representation-dependent Hardness Results

In this section, we will show that for $k \geq 2$, k-term-DNF is not PAC learnable using a hypothesis from k-term-DNF. This type of hardness result is *representation-dependent* since

it only holds if the learner's hypothesis class (or representation class) is restricted to be a certain class. Note that when $k = 1$, the class of k-term-DNF formulas is just the class of monomials, which we know is learnable using a hypothesis of a monomial.

We prove this hardness result by by reducing the learning problem to k-NM-Colorability, a generalization of the Graph k-Colorability problem. Before defining this problem we first describe two known NP-complete problems: Graph k-colorability (NP-complete for $k \geq 3$) and Set Splitting. These descriptions come from Garey and Johnson [14].

Problem: Graph k-colorability
Instance: For a graph $G = (V, E)$, with positive integer $k \leq |V|$.
Question: Is G k-colorable? That is, does there exist a function $f : V \rightarrow \{1, \ldots, k\}$, such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

Problem: Set Splitting
Instance: Collection C of subsets of a finite set S.
Question: Is there a partition of S into two subsets S_1, S_2, such that no subset in C is entirely contained in either S_1 or S_2?

We now generalize both of these problems to obtain the k-NM-Colorability problem which we use for our reduction.

Problem: k-NM-Colorability
Instance: A finite set S and a collection $C = \{c_1, \ldots, c_m\}$ of constraints $c_i \subseteq S$.
Question: Is there a k-coloring χ of the elements of S, such that for each constraints $c_i \in C$, the elements of c_i are not MONOCHROMATICALLY colored (i.e. $\forall c_i \in C$, $\exists x, y \in c_i$, such that $\chi(x) \neq \chi(y)$)?

We now argue that k-NM-Colorability is NP-complete. Clearly k-NM-Colorability is in NP. Note that if every $c_i \in C$ has size 2, then the k-NM-Colorability problem is simply the Graph-k-Colorability problem. Since the Graph-k-Colorability is NP-complete for $k \geq 3$, we only need to show that 2-NM-Colorability is NP-hard. However, note that 2-NM-Colorability is exactly the Set Splitting problem which is NP-complete. Thus it follows that k-NM-Colorability is NP-complete.

We now prove the main results of this section that k-term-DNF is not PAC learnable by k-term-DNF.

Theorem 3.1 *For all integers $k \geq 2$, k-term DNF is not PAC learnable (in polynomial time) using a hypothesis from k-term-DNF unless RP=NP.*

Proof: We reduce k-NM-coloring to the k-term-DNF learning problem. Let (S, C) be an instance of k-NM-coloring, we construct a k-term-DNF learning problem as follows: Each instance (S, C) will correspond to a particular k-term-DNF formula to be learned. If $S = \{s_1, \ldots, s_n\}$, then we will create n variables $\{x_1, \ldots, x_n\}$ for the learning problem. We now describe the positive and negative examples, as well as the distributions D^+ and D^-.

- The positive examples are $\{\vec{p}_i\}_{i=1}^n$, where $\vec{p}_i$ is the vector with x_i=0, and for $j \neq i$,

x_j=1. Thus there are n positive examples. Finally, let D^+ be uniform over these positive examples (i.e. each has weight $1/n$).

- The negative examples are $\{\vec{n}_i\}_{i=1}^{|C|}$, where for each constraint $c_i \in C$, if $c_i = \{s_{i_1}, \ldots, s_{i_m}\}$, then $\vec{n}_i = \vec{0}_{i_1,i_2,\ldots,i_m}$ (all elements of S in c_i are 0, the others are 1). For example, if the constraint c_i is $\{s_1, s_3, s_8\}$, then the vector corresponding to is: $\vec{n}_i =< 0101111011\ldots >$. Finally, let D^- be uniform over these negative examples (so each has weight $1/|C|$).

We now show that a k-term-DNF formula is consistent with all the positive and negative examples defined above if and only if (S, C) is k-NM-colorable. Then we use this claim to show that the learning problem is solvable in polynomial time if and only if RP= NP.

Claim 3.1 *There is a k-term-DNF formula consistent with all positive and negative examples defined above if and only if (S, C) is k-NM-colorable.*

Proof of Claim:
($\Longleftarrow$) Without loss of generality, assume that (S, C) is k-NM-colorable by a coloring $\chi : S \to \{1, 2, \ldots, k\}$, which uses every color at least once. Let f be the k-term-DNF formula $T_1 \vee T_2 \vee \ldots \vee T_k$, where

$$T_i = \bigwedge_{\chi(s_j) \neq i} x_j.$$

In other words, T_i is the conjunction of all x_j corresponding to s_j that are not colored with i.

We now show that f is consistent with positive examples. The positive example $\vec{p}_j$ ($x_j = 0$, $x_i = 1$ for all $i \neq j$) clearly satisfies the term T_i, where $\chi(s_j) = i$. Thus, f is true for all positive examples.

Finally, we show that f is consistent with the negative examples. Suppose some negative example, say $\vec{n}_i = \vec{0}_{i_1,\ldots,i_m}$ satisfies f, then $\vec{n}_i$ satisfies some term, say T_j. Then every element of constraint $c_i = \{s_{i_1}, \ldots, s_{i_m}\}$ must be colored j, (They are 0 in $\vec{n}_i$ and thus must not be in T_j, hence they are colored with j). But then c_i is monochromatic, giving a contradiction.

($\Longrightarrow$) Suppose $T_1 \vee \ldots \vee T_k$ is a k-term-DNF formula consistent with all positive examples and no negative examples. We now show that, without loss of generality, we can assume for all i, T_i is a conjunction of positive literals.

Case 1: T_i contains at least two negated variables. However, all positive examples have a single 0, so none could satisfy T_i. Thus just remove T_i.

Case 2: T_i contains 1 negated variable $\overline{x_j}$. Then T_i can only be satisfied by the single positive example $\vec{p}_j$. In this case, replace T_i by $T_i' = \bigwedge_{j \neq i} x_j$, which is satisfied only by the vectors $\vec{p}_j$ and $\vec{1}$, neither of which are negative examples.

Thus we now assume that all terms are a conjunction of positive literals. Now color the elements of S by the function: $\chi : S \rightarrow \{1, \ldots, k\}$, defined by $\chi(s_i) = \min\{j : x_i \text{ does not occur in } T_j\}$.

Now we show χ is well defined. Since each positive example p_i satisfies $T_1 \vee \ldots \vee T_k$, it must satisfy some term T_j. But each term is a conjunct of unnegated literals. Thus for some j, x_i must not occur in T_j. Thus each element of S receives a color (which is clearly unique).

Finally we show that χ obeys the constraints. Suppose χ violates constraint c_i, then all of the elements in c_i are colored by the same color, say j. By the definition of χ, none of the literals corresponding to elements in c_i occur in term T_j, so the negative example $\vec{n}_i$ associated with c_i satisfies T_j. This contradicts the assumption that none of the negative examples satisfy the formula $T_1 \vee \ldots \vee T_k$. This completes the proof of the claim.

We now complete the proof of the theorem. Namely, we show how a learning algorithm for k-term-DNF can be used to decide k-NM-colorability in random polynomial time. First we give the definition of complexity class RP.

Definition 3.2 *A set S is accepted in the random polynomial time (i.e. S is in RP) if there exists a randomized algorithm A such that on all inputs, A is guaranteed to halt in polynomial time and, if $x \notin S, A(x) =$ "no", if $x \in S, \Pr[A(x) =$ "yes"$] \geq 1/2$.*

Now we show that if there is a PAC learning algorithm for k-term-DNF, it can be used to decide k-NM-colorability in randomized polynomial time. Given instance (S, C), let D^+ and D^- be defined as above. Choose $\delta < \frac{1}{2}$, $\epsilon < min\{\frac{1}{|S|}, \frac{1}{|C|}\}$.

If (S, C) is k-NM-Colorable, then by the above claim there exists a k-term-DNF formula consistent with the positive and negative examples, so with probability at least $1 - \delta$, our learning algorithm will be able to find it.

Conversely, if (S, C) is not k-NM-Colorable, by the Claim, there does not exist a consistent k-term-DNF formula, and the learning algorithm must either fail to produce a hypothesis in the allotted time, or produce one that is not consistent with at least one example. In either case, this can be observed, and we can determine that no legal k-NM-coloring is possible. ■

Thus we have shown that for $k \geq 2$, k-term-DNF is not PAC learnable by k-term-DNF. Furthermore, note that the target function f created in this proof is monotone and thus this result holds even if the concept class is monotone k-term-DNF and the hypothesis class is k-term-DNF. Finally, a dual hardness result applies for learning k-clause-CNF by k-clause-CNF.

3.3 Learning Algorithm for k-term-DNF

Although k-term-DNF is not PAC learnable by k-term-DNF, we now show that it is learnable using a hypothesis from k-CNF.

Let $f = T_1 \vee T_2 \vee \ldots \vee T_k$ be the target formula, where

$$T_1 = y_1^{(1)} \wedge y_2^{(1)} \wedge \ldots \wedge y_{m_1}^{(1)}$$

$$\begin{aligned} T_2 &= y_1^{(2)} \wedge y_2^{(2)} \wedge \ldots \wedge y_{m_2}^{(2)} \\ \vdots &\quad \vdots \\ T_k &= y_1^{(k)} \wedge y_2^{(k)} \wedge \ldots \wedge y_{m_k}^{(k)} \end{aligned}$$

and $y_i^{(j)}$ represents one of the n variables.

By distributing, we can rewrite f as:

$$f = \bigwedge_{i_1, i_2, \ldots, i_k} (y_{i_1}^{(1)} \vee y_{i_2}^{(2)} \vee \ldots \vee y_{i_k}^{(k)}).$$

Now to learn f using a hypothesis form k-CNF, we introduce $O(2n^k)$ variables $\alpha_1, \alpha_2, \ldots, \alpha_m$, representing all disjunctions of the form: $y_{i_1}^{(1)} \vee y_{i_2}^{(2)} \vee \ldots \vee y_{i_k}^{(k)}$. Learning the conjunction over the α's is equivalent to learning the original disjunction. Note, however, we may not (in general) transform the conjunction we obtain to a k-term-DNF formula, thus we must output it as a k-CNF formula.

3.4 Relations Between Concept Classes

In this section, we briefly study the containment relations between various concept classes.

We have already seen that k-term-DNF is a subclass of k-CNF. We now show that k-term-DNF is properly contained in k-CNF by exhibiting a k-CNF formula which can not be expressed as a k-term-DNF formula. The k-CNF formula, $(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \ldots (x_{2k-1} \vee x_{2k})$, has 2^k terms when we unfold it into the form of DNF formula and thus cannot be represented by a k-term-DNF formula.

As a dual result, it is easily shown that k-clause-CNF is properly contained in k-DNF. Thus it follows that, k-CNF $\cup$ k-DNF $\cup$ k-term-DNF $\cup$ k-clause-CNF is PAC learnable.

We now consider the concept class k-DL as defined by Rivest [35]. Consider a Boolean function f that is defined on $\{0,1\}^n$ by a nested **if–then–else** statement of the form:

$$f(x_1, x_2, \ldots, x_n) = \textbf{if } l_1 \textbf{ then } c_1 \textbf{ elseif } l_2 \textbf{ then } c_2 \cdots \textbf{elseif } l_k \textbf{ then } c_k \textbf{ else } c_{k+1}$$

where the l_j's are literals (either one of the variables or their negations), and the c_j's are either **T** (true) or **F** (false). Such a function is said to be computed by a *simple decision list*. The concept class k-decision lists (k-DL) is just the extension of a simple decision list where the condition in each **if** statement may be the conjunction of up to k literals, for some fixed constant k. We leave it as a simple exercise to show that k-CNF $\cup$ k-DNF is properly contained in k-DL. Thus the first problem of Homework 1 provides an even stronger positive result by showing that k-DL is PAC learnable using a hypothesis from k-DL.

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 4: Handling an Unknown Size Parameter

Lecturer: Sally Goldman

4.1 Introduction

We have seen in earlier notes how to PAC learn a k-CNF formula. Recall that the algorithm used for learning k-CNF created a new variable corresponding to each possible term of at most k literals and then just applied the algorithm for learning monomials. Observe that this algorithm assumes that the learner knows k a priori. Can we modify this algorithm to work when the learner does not have prior knowledge of k? Here we consider a variant of the PAC model introduced in Topic 1 in which there is an unknown size parameter. The material presented in this lecture is just one portion of the paper, "Equivalence of Models for Polynomial Learnability," by David Haussler, Michael Kearns, Nick Littlestone, and Manfred Warmuth [21].

4.2 The Learning Algorithm

In this section we outline a general procedure to convert a PAC-learning algorithm A that assumes a known size parameter s (e.g. the k in k-CNF) to a PAC-learning algorithm B that works without prior knowledge of s. As an example application, this procedure will enable us to convert our algorithms for learning k-CNF, k-CNF, k-term-DNF, k-clause-CNF, or k-DL into corresponding algorithms that do not have prior knowledge of k. Observe, that while the learner does not know s, as one would expect the running time and sample complexity of B will still depend on s. The most optimistic goal would be to have the time and sample complexity of B match that of A.

The basic idea of this conversion is as follows. Algorithm B will run algorithm A with an estimate $\hat{s}$ for s such that this estimate is gradually increased. The key question is: How does algorithm B know when its estimate for s is sufficient? The technique of *hypothesis testing* used to solve this problem is a general technique which is also useful in other situations.

Aside: Hypothesis Testing

We now describe a technique to test if a hypothesis is good. More specifically, given a hypothesis h, an error parameter ϵ, and access to an example oracle EX we would like to determine with high probability if h is an ϵ-good hypothesis. Clearly it is not possible to distinguish a hypothesis with error ϵ from one with error just greater than ϵ, however, we

can distinguish an $\epsilon/2$-good hypothesis from an ϵ-bad one. As we shall see this is sufficient to know when our estimate for the size parameter is large enough.

We now formally describe the hypothesis testing algorithm.

Algorithm Test(h,n,ϵ,δ)

1. Make $m = \left\lfloor \frac{32}{\epsilon}(n \ln 2 + \ln 2/\delta) \right\rfloor$ call to EX.
2. Accept h if it misclassifies at most $\frac{3\epsilon}{4}$ of the examples. Otherwise, reject h.

We now prove that this hypothesis testing procedure achieves the goal stated above.

Lemma 4.1 *The procedure Test when called with parameters h, n, ϵ, and δ has the property that:*

1. *If $error(h) \geq \epsilon$, then* Prob[h is accepted] $\leq \frac{\delta}{2^{n+1}}$
2. *If $error(h) \leq \epsilon/2$, then* Prob[h is rejected] $\leq \frac{\delta}{2^{n+1}}$

Proof Sketch:

We first sketch the proof showing the first property holds. Let p be the error of hypothesis h. Then,

$$\text{Prob}[h \text{ is accepted}] \leq LE\left(p, m, \frac{3}{4}\epsilon m\right).$$

Finally, since $p \geq \epsilon$ it follows that

$$LE\left(p, m, \frac{3}{4}\epsilon m\right) \leq LE\left(p, m, (1 - \frac{1}{4})mp\right) \leq e^{-\frac{m\epsilon}{32}}.$$

Plugging in the value of m used in Test, we get the stated result.

For the second property we know that $p \leq \epsilon/2$ and thus the probability of rejecting h is bounded above by:

$$GE\left(p, m, \frac{3}{4}\epsilon m\right) \leq GE\ paren\frac{\epsilon}{2}, m, (1 + \frac{1}{2})m\frac{\epsilon}{2} \leq e^{-\frac{m\epsilon}{24}}.$$

Again this gives the desired bound. ■

Finally, we note that a two-oracle version of this hypothesis testing procedure can be constructed by running the one oracle version Test twice (replacing δ by $\delta/2$), once using the positive oracle and once using the negative oracle. The two-oracle testing procedure accepts h if an only if both of the above calls to Test accept h. The above lemma also holds for this two-oracle testing procedure.

We now return to the problem of handling an unknown size parameter by describing how algorithm B (unknown s) can be implemented using Algorithm A (known s) as a subroutine.

Let $p(n, s, 1/\epsilon) = \max\{S_A(n, s, \epsilon, 1/2), T_A(n, s, \epsilon, 1/2)\}$ where S_A is the sample complexity of algorithm A and T_A is the time complexity of algorithm A. We know describe algorithm B.

Algorithm $B(n, \epsilon, \delta)$

1 $i \leftarrow 0$
2 UNTIL h is accepted by Test(h, n, ϵ, δ) DO
3 $\quad i \leftarrow i + 1$
4 $\quad \hat{s} \leftarrow \left\lfloor 2^{(i-1)/\ln(\frac{2}{\delta})} \right\rfloor$
5 $\quad h_i \leftarrow$ hypothesis output by $A(n, \hat{s}, \epsilon/2, 1/2)$
6 Output $h = h_i$

Theorem 4.1 *Let h be the hypothesis output by algorithm B as described above. Then* $\text{Prob}[error(h) \leq \epsilon] \geq 1 - \delta$.

Proof Sketch:

Observe that algorithm B's estimate $\hat{s} \geq s$ at the ith repetition for all $i \geq \left\lceil 1 + \ln \frac{2}{\delta} \log_2 s \right\rceil$. Since the size parameter is only an upperbound on the allowable size and algorithm A is a PAC-learning algorithm we know that for any iteration in which $\hat{s} \geq s$, the $\text{Prob}[h_i \leq \epsilon/2] \geq 1/2$. In such a case, the $\text{Prob}[h_i \text{ is accepted by Test}] \geq 3/4$. So if $\hat{s} \geq s$ then the $\text{Prob}[B \text{ halts with hyp. of error} \leq \epsilon/2] \geq 3/8$.

Let $j = \lfloor (\ln 2/\delta)/(\ln 8/5) \rfloor$. Then

$$\text{Prob}[B \text{ fails to halt after } j \text{ iterations with } \hat{s} \geq s] \leq \left(\frac{5}{8}\right)^j \leq \delta/2.$$

Thus with probability at least $1 - \delta/2$, B will halt after at most

$$j' = \left\lceil \ln \frac{2}{\delta} \log_2 s \right\rceil + \left\lceil \frac{\ln 2/\delta}{\ln 8/5} \right\rceil$$

iterations.

Also the probability is at most $\delta/2$ that any call to Test will accept a hypothesis with error greater than ϵ. Thus with probability $\geq 1 - \delta$, algorithm B will halt after at most j' repetitions with an ϵ-good hypothesis.

Finally, one can verify that the time an sample complexity after j' iterations is still polynomial. We refer the reader to the paper by Haussler et al. [21] for the details. ■

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 5: Learning with Noise

Lecturer: Sally Goldman *Scribe: Gadi Pinkas*

5.1 Introduction

Although up to now we have assumed that the data provided by the example oracle is noise-free, in real-life learning problems this assumption is almost never valid. Thus we would like to be able to modify our algorithms so that they are robust against noise. Before considering learning with noise in the PAC model, we must first formally model the noise. Although we will only focus on one type of noise here, we first describe the various formal models of noise that have been considered.

In all cases we assume that the usual noise free examples pass through a noise oracle before being seen by the learner. Each noise oracle represents some noise process being applied to the examples from EX. The output from the noise process is all the learner can observe. The "desired," noiseless output of each oracle would thus be a correctly labeled example (x, s), where x is drawn according to the unknown distribution D. We now describe the actual outputs from the following noise oracles:

Random Misclassification Noise [7]: This noise oracle models a benign form of misclassification noise. When it is called, it calls EX to obtain some (noiseless) (x, s), and with probability $1 - \eta$, it returns (x, s). However, with probability η, it returns $(x, \overline{s})$.

Malicious Noise [44]: This oracle models the situation where 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 or unpredictable. When this oracle is called, with probability $1 - \eta$, it does indeed return a correctly labeled (x, s) where x is drawn according to D. With probability η it returns an example (x, s) about which no assumptions whatsoever may be made. In particular, this example may be maliciously selected by an adversary who has infinite computing power, and has knowledge of the target concept, D, η, and the internal state of the learning algorithm.

Malicious Misclassification Noise [41]: This noise oracle models a situation in which the only source of noise is misclassification, but the nature of the misclassification is unknown or unpredictable. When it is called, it also calls EX to obtain some (noiseless) (x, s), and with probability $1 - \eta$, it returns (x, s). With probability η, it returns (x, l) where l is a label about which no assumption whatsoever may be made. As with malicious noise we assume an omnipotent, omniscient adversary; but in the case the adversary only gets to choose the label of the example.

Uniform Random Attribute Noise [41]: This noise oracle models a situation where the attributes of the examples are subject to noise, but that noise is as benign as possible. For example, the attributes might be sent over a noisy channel. We consider this oracle only when the instance space is $\{0,1\}^n$ (i.e., we are learning Boolean functions). This oracle calls EX and obtains some $(x_1 \cdots x_n, s)$. It then adds noise to this example by independently flipping each bit x_i to $\bar{x}_i$ with probability η for $1 \leq i \leq n$. Note that the label of the "true" example is never altered.

Nonuniform Random Attribute Noise [17]: This noise oracle provides a more realistic model of random attribute noise than uniform random attribute noise.[2] This oracle also only applies when we are learning Boolean functions. This oracle calls EX and obtains some $(x_1 \cdots x_n, s)$. The oracle then adds noise by independently flipping each bit x_i to $\bar{x}_i$ with some fixed probability $\eta_i \leq \eta$ for each $1 \leq i \leq n$.

In this paper we focus on the situation in which there is random misclassification noise. The material presented here comes from the paper "Learning from Noisy Examples," by Dana Angluin and Phil Laird [7]. We show that the hypothesis that minimizes disagreements (i.e. the hypothesis that misclassifies the fewest training examples) meets the PAC correctness criterion when the examples are corrupted by random misclassification noise. Unfortunately, this technique is most often computationally intractable. However, for k-CNF formulas we describe an efficient PAC learning algorithm that works against random misclassification noise. Both positive results need only assume that the noise rate η is less than one half.

Before describing these results, we briefly review what is known about handling the other forms of noise. Sloan [41] has extended the above results to the case of malicious labeling noise. On the other hand, Kearns and Li [25] have shown that the method of minimizing disagreements can only tolerate a small amount of malicious noise. We will study this result in Topic 15.

Unlike the results for labeling noise, in the case of uniform random attribute noise, if one uses the minimal disagreement method, then the minimum error rate obtainable (i.e. the minimum "epsilon") is bounded below by the noise rate [41]. Although the method of minimizing disagreements is not effective against random attribute noise, there are techniques for coping with uniform random attribute noise. In particular, Shackelford and Volper [38] have an algorithm that tolerates large amounts of random attribute noise for learning k-DNF formulas. That algorithm, however, has one very unpleasant requirement: it must be given the *exact* noise rate as an input. Goldman and Sloan [17] describe an algorithm for learning monomials that tolerates large amounts of uniform random attribute noise (any noise rate less than 1/2), and only requires some upper bound on the noise rate as an input. Finally, for nonuniform random attribute noise, Goldman and Sloan [17] have shown that the minimum error rate obtainable is bounded below by one-half of the noise rate, regardless of the technique (or computation time) of the learning algorithm.

[2] Technically, this oracle specifies a family of oracles, each member of which is specified by n variables, $\eta_1, \ldots, \eta_n$, where $0 \leq \eta_i \leq \eta$.

5.2 Learning Despite Classification Noise

Let EX_η be the random misclassification noise oracle with a noise noise rate of η. Thus the standard noise free oracle is EX_0. Also we will use the notation that $C = \{L_1, L_2, \ldots, L_N\}$ where L_* is the target concept. So we only consider the simple case of a finite set of hypothesis.

In this section we will study the random misclassification model and give an algorithm to PAC learn any finite concept space with polynomial sample size (not necessarily with polynomial time). In the next section we show that for $\eta < 1/2$, the class of k-CNF formulas is PAC learnable from EX_η. Observe that the learning problem is not feasible for $\eta \geq 1/2$. If $\eta = 1/2$ the noise distorts all the information and clearly no learning is possible, and when $\eta > 1/2$, we actually learn the complement concept with $\eta < 1/2$.

For now we assume that the learner has an upper bound η_b on the noise rate. That is, $\eta \leq \eta_b < 1/2$. Later we show how to remove this assumption. As one would expect if η_b is very close to $1/2$, we must allow the learner more time and data. In fact, we will require that the time and sample complexity of the learner are polynomial in $1/(1-2\eta_b)$. Observe that this quantity is inversely proportional to how close η_b is to $1/2$.

5.2.1 The Method of Minimizing Disagreements

Our goal in this section is to study the sample complexity for PAC learning under random misclassification noise. For the noise-free case we have seen that if L_i agrees with at least $m \geq (1/\epsilon)(\ln|C| + \ln(1/\delta)) = (1/\epsilon)\ln(N/\delta)$ samples drawn from EX_0 then $\Pr[error(L_i) \geq \epsilon] \leq \delta$. How much more data is needed when there is random misclassification noise?

In the presence of noise the above approach will fail because there is no guarantee that any hypothesis will be consistent with all the examples. However, if we replace the goal of consistency with that of minimizing the number of disagreements with the examples and permit the sample size to depend on η_b, we get an analogous result for the noisy case.

We shall use the following notation in formally describing the method of minimizing disagreements.

- Let σ be the sequence of examples drawn from EX_η.
- Let $F(L_i, \sigma)$ be the number of times L_i disagrees with σ on an example in σ, where L_i disagrees with σ on an example $(\vec{x}, l) \in \sigma$ if and only if L_i classifies $\vec{x}$ differently from l.

Theorem 5.1 *If we draw a sequence σ of*

$$m \geq \frac{2}{\epsilon^2(1-2\eta_b)^2}\ln\left(\frac{2N}{\delta}\right)$$

samples from EX_η and find any hypothesis L_i that minimizes $F(L_i, \sigma)$ then

$$\Pr[error(L_i) \geq \epsilon] \leq \delta$$

.

Proof: We shall use the following notation in the proof. Let d_i be the $error(L_i)$. That is, d_i is the probability that L_i misclassifies a randomly drawn example. Let p_i be the probability that an example from EX_η disagrees with L_i. Observe that p_i is the probability that L_i misclassifies a correctly labeled example ($d_i(1-\eta)$), plus the probability that L_i correctly classifies the example but the example has been improperly labeled by the noise oracle ($(1-d_i)\eta$). Thus

$$p_i = d_i(1-\eta) + (1-d_i)\eta = \eta + d_i(1-2\eta)$$

Note that for the right hypothesis ($L_i = L_*$), $d_i = 0$ and therefore $p_i = \eta$ (i.e., disagreements are only caused by noise). Since $\eta < 1/2$, it follows that for any hypothesis $p_i \geq \eta$. So all hypothesis have an expected rate of disagreement of at least η.

Let an ϵ-bad hypothesis be one for which $d_i \geq \epsilon$. Then for any ϵ-bad hypothesis L_i, we have

$$p_i \geq \eta + \epsilon(1-2\eta).$$

Thus we have a separation of at least $\epsilon(1-2\eta)$ between the disagreement rates of the correct and an ϵ-bad hypothesis. Although η_b is not known, we know that $\eta \leq \eta_b < 1/2$ thus the minimum separation (or gap) is at least $\epsilon(1-2\eta_b)$. We take advantage of this gap in the following manner. We will draw enough examples from EX_η to guarantee with high probability that no ϵ-bad hypothesis has a observed disagreement rate greater than $\eta + \epsilon(1-2\eta_b)/2$. Similarly, we will draw enough examples from EX_η to guarantee with high probability that the correct hypothesis has on observed disagreement rate less than $\eta + \epsilon(1-2\eta_b)/2$. Thus it follows that with high probability L_* will have a lower observed rate of disagreement than any ϵ-bad hypothesis. Thus by selecting the hypothesis with the lowest observed rate of disagreement, the learner knows (with high probability) that this hypothesis has error at most ϵ.

We now formalize this intuition. We will draw m examples from EX_η and compute an empirical estimate for all p_i. That is, we compute $F(L_i, \sigma)$ for every L_i in the hypotheses space. The hypothesis output will be the hypothesis L_i that has the minimum estimate for p_i. What is the probability that L_i is ϵ-bad? Let $s = \epsilon(1-2\eta_b)$. In order for some ϵ-bad hypothesis L_i to minimize $F(L_i, \sigma)$ either the correct hypothesis must have a high disagreement rate

$$F(L_*, \sigma)/m \geq \eta + s/2$$

or an ϵ-bad hypothesis must have a low disagreement rate ($\leq \eta + s/2$). Finally, assuming that neither of these bad events occur, since we select the hypothesis L_i that minimizes the disagreement rate we know that:

$$F(L_i, \sigma)/m < \eta + s/2$$

and thus L_i has error of at most ϵ.

Applying Chernoff bounds for the probability that a good hypothesis has high disagreement:

$$\Pr[F(L_*, \sigma)/m \geq \eta + s/2] = GE(\eta, m, m(\eta + s/2) < \delta/(2N) < \delta/2.$$

And if L_i is ϵ-bad then its probability to have low disagreement is:

$$\Pr[F(L_*, \sigma)/m \leq \eta + s/2] = LE(\eta + s, m, m(\eta + s/2) \leq \delta/(2N).$$

Thus the probability that any ϵ-bad hypothesis L_i has $F(L_i, \sigma)/m \leq \eta + s/2$ is at most $\delta/2$. (There are at most $N - 1$ hypothesis that are ϵ-bad.) Putting these two equalities together, the probability that some ϵ-bad hypothesis minimizes $F(L_i, \sigma)$ is at most δ. ■

Thus we know that by using the method of minimizing disagreements one can tolerate any noise rate strictly less than 1/2. Furthermore, if the hypothesis minimizing disagreements can be found in polynomial time then we obtain an efficient PAC algorithm for learning when there is random misclassification noise.

5.2.2 Handling Unknown η_b

Until now we have assumed the learner is given an upperbound η_b on the noise rate. What if such an upperbound is not known? We can solve this problem using the technique described in Topic 4 for handling an unknown size parameter. That is, just treat η as the unknown size parameter. The only detail we need to worry about here is how to perform the hypothesis testing. The basic idea is as follows, we draw some examples and estimate the failure probability of each of the hypotheses $L_1, ..., L_N$. The smallest estimate is compared to the current value of η_b. If the estimate $\hat{p}_i$ is less than the current value of η_b, we halt; otherwise we increase η_b and repeat. For details see the Angluin, Laird paper [7].

5.2.3 How Hard is Minimizing Disagreements?

How hard is to find an hypothesis L_i that minimizes $F(L_i, \sigma)$? Unfortunately, the answer is that is usually quite hard. For example consider the domain of all conjunctions of positive literals (monotone monomials). To find a monotone monomial that minimizes the disagreement is a NP-hard problem.

Theorem 5.2 *Given a positive integer n and c and a sample σ. The problem of determining if is there a monotone monomial ψ over n variables such that $F(\pi, \sigma) \leq c$ is NP-complete.*

The result indicates that even for a very simple domain, the approach of directly trying to minimize the disagreement is unlikely to be computationally feasible. However, in the next section we show that for some concept classes, we can bypass the minimization problem (which is hard) and efficiently PAC learn the concepts from noisy examples.

5.3 Learning k-CNF Under Random Misclassification Noise

In the previous section we described how to PAC learn any finite concept space if we remove the condition that our algorithm runs in polynomial time. However, we would like to have efficient algorithms for dealing with noise. In this section we use some of the ideas suggested by the method of minimizing disagreements to get a polynomial time algorithm for PAC learning k-CNF formulas using examples from EX_η.

5.3.1 Basic Approach

Instead of searching for the k-CNF formula with the fewest disagreements, we will test all potential clauses individually, and include those that are rarely false in a positive example. Of course, if a clause is false yet the example is reported as positive then either the clause is not in the target formula, or the label has been inverted by the noise process. Thus if a clause is false on a significant number of positive examples, then we do not want to include the clause in our hypothesis. Observe that we will not be solving an NP-complete problem, the k-CNF formula that is chosen may not minimize the disagreements with the examples, but it will (with probability $\geq 1-\delta$) have an error that is less then ϵ.

We now give some notation that we use in this section.

- Let M be the number of possible clauses of at most k literals ($M \leq (2n+1)^k$), and let C be any such clause.
- Let ϕ_* be the target k-CNF formula
- For all clauses C, let $P_{00}(C) = \text{Prob}[C \text{ is false and } \phi_* \text{ is false}]$.
- For all clauses C, let $P_{01}(C) = \text{Prob}[C \text{ is false but } \phi_* \text{ is true}]$.
- For all clauses C, let $P_0(C) = P_{00}(C) + P_{01}(C) = \text{Prob}[C \text{ is false}]$.

Finally, we need the following two definitions. We say that a clause C is *important* if and only if $P_0(C) \geq Q_I = \epsilon/(16M^2)$. We say a clause C is *harmful* if and only if $P_{01}(C) \geq Q_H = \epsilon/(2M)$. Note that $Q_H \geq Q_I$, so every harmful clause is important. Also, no clause contained in ϕ_* can be harmful, since if ϕ_* classifies an example as positive, the example satisfies all its clauses and C must be true (i.e., $P_{01}(C) = 0$).

The algorithm to PAC learn k-CNF formulas works as follows. We must construct an hypothesis h that is ϵ-good with high probability. To achieve this goal the hypothesis h must have all the important clauses that are not harmful. A non-important clause is almost always assigned the value "true" (by the examples in EX_η), and thus it does not mater if it is included in h or not. On the other hand, a harmful clause must not be included in h, since it is very likely to be falsified by a positive example.

Thus our goal is to find an hypothesis h that includes all important clauses and does not include any harmful clause. We first prove that if we find such a hypothesis h then it is ϵ-good. Then we show how to efficiently construct such a hypothesis with high probability.

Lemma 5.1 *Let D be a fixed unknown distribution, and let ϕ_* be a fixed unknown target. Let ϕ be any conjunction (product) of clauses that contains every important clause in ϕ_* and no harmful clauses. Then $error(\phi) \leq \epsilon$.*

Proof: The probability that ϕ misclassifies an example from D is equal to the probability that ϕ_* classifies the example as positive but ϕ classifies it as negative, or vice versa (the example is truly negative but ϕ classifies it as positive).
The probability of an error on a positive example is equal to the probability that any clause in $\phi - \phi_*$ is falsified by positive example; therefore,

$$\text{Prob}[\phi_* = 1 \wedge \phi = 0] \leq \sum_{C \in \phi - \phi_*} P_{01}(C) < MQ_H = \epsilon/2$$

since there are at most M clauses in ϕ and none are harmful.

The probability of error on a negative example is equal to the probability that a clause in $\phi_* - \phi$ is falsified and all clauses in ϕ are true. This probability is less or equal to the probability that a clause in $\phi_* - \phi$ is falsified. Since ϕ contains all the important but not harmful clauses and ϕ_* contains no harmful clauses, then $\phi_* - \phi$ contains only non-important clauses. Therefore,

$$\text{Prob}[\phi = 1 \wedge \phi_* = 0] \leq \sum_{C \in \phi_* - \phi} P_0(C) < MQ_I = \epsilon/(16M) < \epsilon/2$$

Thus,

$$error(\phi) \leq \text{Prob}[\phi = 1 \wedge \phi_* = 0] + \text{Prob}[\phi = 0 \wedge \phi_* = 1] \leq \epsilon/2 + \epsilon/2 = \epsilon.$$

■

To complete the algorithm for learning k-CNF under noise, we must construct an efficient algorithm to find a formula ϕ that contains all the important clauses and no harmful clauses (with high probability). Observe that we have no direct information about whether a clause is important or harmful. More specifically, we cannot directly compute $P_{00}(C)$ or $P_{01}(C)$, but rather must rely on the examples received from EX_η. However, since EX_η only modifies the label and not the values of the attributes in the example, $P_0(C)$ can be directly estimated by drawing a sample from EX_η, and calculating the fraction of the assignments that assign "false" to C. Thus we can accurately estimate $P_0(C)$ for every possible clause and thus decide (with high probability) which clauses are important. Let I be the set of clauses that are determined to be important. The only thing left, is to identify the harmful clauses in I and eliminate them.

Harmful clauses are those that are falsified by positive examples (examples that satisfy ϕ_*), but since the classification of the example is subject to noise we cannot directly estimate

$P_{01}(C)$. Let $P_{0+}(C)$ be the probability that a a clause is falsified by an example and that EX_η reports that the example is positive. This probability can be directly estimated by counting the number of examples from the sample that are reported as positive and falsify the clause C. We perform this estimate for every important clause. For the estimate of $P_{0+}(C)$, an example is counted if and only if the example is positive and no error occurred in the reporting, or the example is negative and there was an error in the reporting. Thus,

$$\begin{aligned} P_{0+}(C) &= (1-\eta)P_{01}(C) + \eta P_{00}(C) \\ &= \eta(P_{00}(C) + P_{01}(C)) + (1-2\eta)P_{01}(C) \\ &= \eta P_0(C) + (1-2\eta)P_{01}(C) \end{aligned}$$

If $P_0(C) \neq 0$, then the proportion of examples falsifying C that are reported as positive is:

$$\frac{P_{0+}(C)}{P_0(C)} = \eta + \frac{P_{01}(C)}{P_0(C)}(1-2\eta)$$

and since $\eta < 1/2$, we get:

$$\frac{P_{0+}(C)}{P_0(C)} \geq \eta.$$

We would like to separate between two cases: one is the case where C is a desired clause ($C \in \phi_*$) and the other is when C is harmful. If $C \in \phi_*$ then

$$\frac{P_{0+}(C)}{P_0(C)} = \eta,$$

while if C is harmful ($P_{01}(C) \geq Q_H$):

$$\frac{P_{0+}(C)}{P_0(C)} \geq \eta + Q_H(1-2\eta) \geq \eta + \frac{\epsilon}{2M}(1-2\eta).$$

Thus, we have a separation of at least $s = Q_H(1-2\eta)$ between the clauses that are included in ϕ_* and the harmful clauses. Since $\eta \leq \eta_b$, the separation is bounded bellow by $s = Q_H(1-2\eta_b)$.

If we knew the value of η, we could estimate P_{0+}/P_0 for all clauses and delete from I all the clauses with an estimate that is greater than $\eta + s/2$. However, we do not know η and thus we do not know where the cutoff is.

How can we estimate η? If I contains any clause C that is also in ϕ_* then the estimate of $P_{0+}(C)/P_0(C)$ will be close to η. So we can take the minimum of this value over all the clauses in I as a first estimate for η:

$$\hat{\eta} = \min_{c \in I} \frac{P_{0+}(C)}{P_0(C)}.$$

If no clause in I is contained also in ϕ_*, then the estimate $\hat{\eta}$ calculated above may not be a good estimate of η. However, since I contains *all* the important clauses it must be that

ϕ_* contains no important clauses. This means that most of the examples drawn from D satisfy these non-important clauses and therefore satisfy ϕ_*. In this case most examples are positive, and thus the observed overall rate of negative examples is sufficiently close to η. Thus, the estimate of η is taken to be the minimum of the two estimates:

$$\hat{\eta} = \min\left\{\text{fraction of negative examples}, \min_{c \in I}\left\{\frac{P_{0+}(C)}{P_0(C)}\right\}\right\}.$$

We use therefore $\hat{\eta} + s/2$ to decide which are the harmful clauses.

5.3.2 Details of the Algorithm

We now put these ideas together to get an efficient algorithm for PAC learning k-CNF formulas under random misclassification noise.

Algorithm Learn-Noisy-k-CNF(n,k,ϵ,δ,η_b)

1. $m = \lceil \frac{2^{10}M^4}{\epsilon^3(1-\eta_b)^2} \ln(\frac{6M}{\delta}) \rceil$, where M is the number of possible clauses
2. $Q_I = \epsilon/(16M^2); Q_H = \epsilon/2M; s_b = Q_H(1-2\eta_b)$
3. We draw m examples from the oracle EX_η, and set: $\hat{P}_-$ = number of negative examples in the sample
4. For each possible clause C, we compute:
   (a) $\hat{P}_0(C)$ = the number of examples that falsify the clause C
   (b) $\hat{P}_{0+}(C)$ = the number of positive (reported) examples that falsify C
   (c) If $P_0(C) \neq 0$ then $h(C) = \hat{P}_{0+}(C)/\hat{P}_0(C)$.
   (d) $\eta_1 = \hat{P}_-/m$, the observed fraction of negative examples
5. Form the set I by including all the important clauses C; i.e., $\hat{P}_0(C)/m \geq Q_I/2$
6. $\eta_2 = \min_{c \in I}\{h(C)\}$
7. $\hat{\eta} = \min\{\eta_1, \eta_2\}$
8. The final output ϕ is the conjunction (product) of all those clauses $C \in I$ such that $h(C) \leq \hat{\eta} + s_b/2$

To prove that the algorithm is correct, we need to prove that ϕ contains all important clauses that are not harmful (with probability greater than $1-\delta$). Only in the following cases the algorithm could go wrong:

- Some important clauses might not be selected for inclusion in I.

- the estimate $\hat{\eta}$ could be too large ($\hat{\eta} \geq \eta + s/4$) or too small ($\hat{\eta} \leq \eta - s/4$).
- Some harmful clauses may be included in ϕ.
- Some correct clauses may be excluded from ϕ.

The proof uses Chernoff bounds, showing that the second case has a probability of at most $\delta/2$, while the other cases, each has a probability of at most $\delta/6$. Therefore, the toatl probability of error is at most δ, and by the previous lemma, the output formula is ϵ-good with high probability. For the details of the proof, see the Angluin, Laird paper[7].

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 6: Occam's Razor

Lecturer: Sally Goldman *Scribe: Nilesh Jain*

"Entities should not be multiplied unnecessarily"
William of Occam, c. 1320

6.1 Introduction

The above quote is better known to us today as Occam's Razor or the Principle of Ontological Parsimony. Over the centuries, people from different fields have given it different interpretations. One interpretation used by the experimental scientists is: *given two explanations of the data, all other things being equal, the simpler explanation is preferable.*

This principle is consistent with the goal of machine learning: to discover the simplest hypothesis that is consistent with the sample data. However, the question still remains: why should one assume that the simplest hypothesis based on past examples will perform well on future examples. After all, the real value of a hypothesis is in predicting the examples that it has not yet seen.

It will be shown that, under very general assumptions, Occam's Razor produces hypotheses that with high probability will be predictive of future observations. As a consequence, when hypotheses of minimum or near minimum complexity can be produced in time polynomial in the size of the sample data, it leads to a polynomial PAC-learning algorithm. The material presented in the lecture comes from the paper "Occam's Razor," by Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred Warmuth [11]. Portion of the notes are taken from this paper.

6.2 Using the Razor

In this section, we define an Occam-algorithm and show that the existence of an Occam-algorithm implies polynomial learnability. First, we prove the *uniform convergence lemma*.

Lemma 6.1 (Uniform Convergence Lemma) *Given any function f in a hypothesis class of r hypotheses, the probability that any hypothesis with error larger than ϵ is consistent with a sample of size m is less than $(1-\epsilon)^m r$.*

Proof: Let h_i be any hypothesis with error larger than ϵ. The probability that h_i is consistent with one observation of f is less than $(1-\epsilon)$. Since all the m samples of f are drawn independent of each other, the probability that h_i is consistent with all m observations of f is less than $(1-\epsilon)^m$. Finally, since there are r hypothesis, the probability that any

hypothesis with error larger than ϵ is consistent with all m observations of f is less than $(1-\epsilon)^m r$. ■

For an infinite hypothesis class, the learning algorithm would have to choose a hypothesis carefully. Occam's Razor suggests that the learning algorithm should choose its hypothesis among those that are consistent with the sample and have the minimum complexity. But this may sometimes be intractable. For example, finding a minimum length DNF expression consistent with a sample of a Boolean function and finding a minimum size DFA consistent with a set of positive and negative examples of a regular language are known to be NP-hard [14]. To obtain polynomial algorithms, the criterion is weakened as follows.

Definition 6.1 *An Occam-algorithm for H with constant parameters $c \geq 1$ and compression factor $0 \leq \alpha < 1$ is a learning algorithm that given a sample of size m which is labeled according to some hypothesis $h \in H$:*

1. *produces a hypothesis consistent with the data, that is, all the observations can be explained by the hypothesis,*
2. *produces a hypothesis of complexity at most $n^c m^\alpha$ where n is the complexity of h, and*
3. *runs in time polynomial in n and m.*

We now show that the existence of an Occam-algorithm for H implies polynomial learnability.

Theorem 6.1 *Given independent observations of any function in H of complexity at most* n, *an Occam-algorithm with parameters $c \geq 1$ and $0 \leq \alpha < 1$ produces a hypothesis of error at most ϵ with probability at least $1-\delta$ using sample size polynomial in $n, 1/\epsilon$, and $1/\delta$, independent of the function and of the probability distribution. The sample size required is*

$$O\left(\frac{1}{\epsilon}\lg\left(\frac{1}{\delta}\right) + \left(\frac{n^c}{\epsilon}\right)^{1/(1-\alpha)}\right).$$

Proof: Let the size of the hypothesis space be r. We will show that the Occam-algorithm produces a good hypothesis after drawing a sample of size

$$m \geq \max\left\{\frac{2\lg(1/\delta)}{-\lg(1-\epsilon)}, \left(\frac{2n^c}{-\lg(1-\epsilon)}\right)^{1/(1-\alpha)}\right\}$$

Since the hypothesis under consideration are given by binary strings of length at most $n^c m^\alpha$, the size of the hypothesis space, r, is at most $2^{n^c m^\alpha}$.

We first consider the second lower bound on m,

$$m \geq \left(\frac{2n^c}{-\lg(1-\epsilon)}\right)^{1/(1-\alpha)}.$$

Raising both sides to the power of $(1-\alpha)$ yields

$$m^{1-\alpha} \geq \frac{2n^c}{-\lg(1-\epsilon)}.$$

Simplifying further we obtain

$$n^c m^\alpha \leq -\frac{1}{2} m \lg(1-\epsilon).$$

Finally, raising both sides to the power of 2 gives

$$2^{n^c m^\alpha} \leq (1-\epsilon)^{-m/2}.$$

Recall that $r \leq 2^{n^c m^\alpha}$ and thus it follows from above that:

$$r \leq (1-\epsilon)^{-m/2}.$$

From the uniform convergence lemma, we get that:

$$\begin{aligned} \Pr[\text{any } \epsilon\text{−bad hyp. is consistent with } m \text{ exs.}] &\leq (1-\epsilon)^m r \\ &\leq (1-\epsilon)^{-m/2}(1-\epsilon)^m \\ &= (1-\epsilon)^{m/2}. \end{aligned} \tag{6.1}$$

Finally, we consider the first lower bound on m,

$$m \geq \frac{2\lg(1/\delta)}{-\lg(1-\epsilon)}.$$

Multiplying both sides by $\lg(1-\epsilon)$ which is negative, we get

$$\frac{m}{2}\lg(1-\epsilon) \leq -\lg(1/\delta) = \lg\delta.$$

Raising both sides to the power of 2 gives

$$(1-\epsilon)^{m/2} \leq \delta \tag{6.2}$$

We complete the proof by combining Equations (6.1) and (6.2) gives that the probability that an ϵ-bad hypothesis is consistent with all the m examples is at most δ. Thus the Occam-algorithm produces a good hypothesis after drawing m examples where

$$m = O\left(\frac{1}{\epsilon}\lg\left(\frac{1}{\delta}\right) + \left(\frac{n^c}{\epsilon}\right)^{1/(1-\alpha)}\right)$$

By definition of an Occam-algorithm, it runs in time polynomial in n and m. Thus, we have a PAC-learning algorithm. ■

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 7: The Vapnik-Chervonenkis Dimension

Lecturer: Sally Goldman *Scribe: Nilesh Jain*

7.1 Introduction

In the notes on Occam's Razor, we defined the condition for uniform convergence on finite concept classes. We can state the general condition for uniform convergence as follows:

$$\Pr_{S \in D^m} [\exists h \in \mathcal{C} \ | error_D(h) > \epsilon \wedge h \text{ is consistent with } S] \leq \delta$$

where S is a set of m examples from the instance space X having probability distribution D. To determine whether a concept class $\mathcal{C}$ is uniformly learnable, we have to find out if there exists a learning function $\mathcal{A}$ satisfying the above condition. If so, we say that $\mathcal{C}$ is uniformly learnable and has sample complexity m.

For finite concept classes, such uniform convergence proofs are easy because we know $|\mathcal{C}|$. However, when the concept class is infinite, we need some other measure to replace $|\mathcal{C}|$. Such a measure is a combinatorial parameter known as the *Vapnik-Chervonenkis (VC) dimension.* The VC dimension of a concept class $\mathcal{C}$ is a measure of the complexity of the class.

In the next section, we will define the VC dimension and some other concepts needed to define the VC dimension. We shall then measure the VC dimension of some concept classes. The following section will contain the theorems relating the VC dimension to uniform learnability and the sample complexity. Finally, we will see some relations on the VC dimension. The material presented in the lecture comes from the paper "Learnability and the Vapnik-Chervonenkis Dimension," by Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred Warmuth [12]. Portions of the notes are taken from this paper.

7.2 VC Dimension Defined

The definition of VC dimension uses the definition of a *shattered* set. We now give two equivalent definitions for a shattered set. Let X be the instance space and $\mathcal{C}$ the concept class.

Definition 7.1 A finite set $S \subseteq X$ is *shattered* by $\mathcal{C}$ if for each subset $S' \subseteq S$, there is a concept $c \in \mathcal{C}$ which contains all of S' and none of $S - S'$.

In order to give the alternate definition of shattering, we first need the following definition.

Definition 7.2 Given $S \subseteq X$, $\Pi_{\mathcal{C}}(S)$ denotes the set of all subsets of S that can be obtained by intersecting S with a concept in $\mathcal{C}$. Thus,

$$\Pi_{\mathcal{C}}(S) = \{S \cap c : c \in \mathcal{C}\}.$$

For any integer $m > 0$, $\Pi_{\mathcal{C}}(m) = \max(|\Pi_{\mathcal{C}}(S)|)$ over all $S \subseteq X$ where $|S| = m$.

Definition 7.3 Given $S \subseteq X$, if $\Pi_{\mathcal{C}}(S) = 2^S$, the power set of S, then S is *shattered* by $\mathcal{C}$.

We can now define the VC dimension.

Definition 7.4 *VC dimension* of $\mathcal{C}$, denoted as $\text{VCD}(\mathcal{C})$, is the smallest d for which no set of $d+1$ instances is shattered by $\mathcal{C}$.

Definition 7.5 Equivalently, $\text{VCD}(\mathcal{C})$ is the cardinality of the largest finite set of points $S \subseteq X$ that is shattered by $\mathcal{C}$ (i.e., the largest integer d such that $\Pi_{\mathcal{C}}(d) = 2^d$).

7.3 Example Computations of VC Dimension

Consider a concept class $\mathcal{C}$ with a finite VC dimension. To show the lower bound on $\text{VCD}(\mathcal{C})$, (i.e., $\text{VCD}(\mathcal{C}) \geq d$), we have to show that there exists a set of size d that is shattered by $\mathcal{C}$. To show the upper bound on $\text{VCD}(\mathcal{C})$, (i.e., $\text{VCD}(\mathcal{C}) \leq d$), we have to show that no set of size $d+1$ is shattered by $\mathcal{C}$. To show that $\text{VCD}(\mathcal{C}) = d$, we have to show that there exists a set of size d that is shattered by $\mathcal{C}$, and no set of size $d+1$ is shattered by $\mathcal{C}$. Keeping this in mind, let us compute $\text{VCD}(\mathcal{C})$ for some concept classes.

Example 7.1 *Intervals on the real line*

The concepts are intervals on the real line. Points lying on or inside the interval are positive, and points lying outside the interval are negative.

We first show that there exists a set of size two that can be shattered by $\mathcal{C}$.

Consider Figure 7.1. Let $S = \{x_1, x_2\}$ be a subset of the instance space X. Consider the concepts $c_1 = [0, r_1], c_2 = [r_1, r_2], c_3 = [r_2, r_3], c_4 = [0, 1]$. Let the concept class $\mathcal{C} = \{c_1, c_2, c_3, c_4\}$. Then, we have $c_1 \cap S = \phi, c_2 \cap S = \{x_1\}, c_3 \cap S = \{x_2\}$ and $c_4 \cap S = S$. Thus, S is shattered by $\mathcal{C}$.

Finally, we must show that no set of size three can be shattered by $\mathcal{C}$.

Consider Figure 7.2. Let $S = \{x_1, x_2, x_3\}$ be a subset of the instance space X. There is no concept which contains x_1 and x_3 and does not contain x_2. Thus, S is not shattered by $\mathcal{C}$.

Thus, $\text{VCD}(\mathcal{C}) = 2$.

Example 7.2 *Axis-parallel rectangles in* $\mathbb{R}^2$

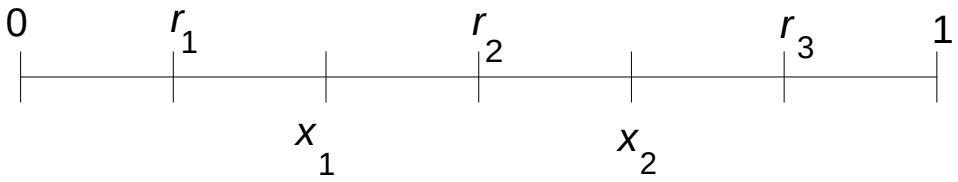

Figure 7.1: A line of reals with some instances and concepts.

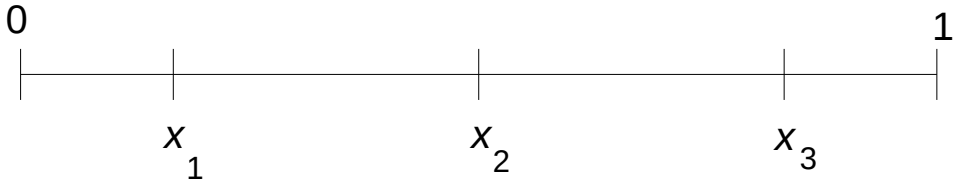

Figure 7.2: A line of reals with some instances.

The concepts are axis-parallel rectangles in $\mathbb{R}^2$. Points lying on or inside the rectangle are positive, and points lying outside the rectangle are negative.

We first show that there exists a set of size four that can be shattered by $\mathcal{C}$. Consider any four points, no three of which are collinear. Clearly, these points can be shattered by concepts from $\mathcal{C}$. Finally, we must show that no set of size five can be shattered by $\mathcal{C}$. Given any five points, one of the following two cases occur.

1. At least three of the points are collinear. In this case, there is no rectangle which contains the two extreme points, but does not contain the middle points. Thus clearly, the five points cannot be shattered.

2. No three of the points are collinear. In this case, consider the bounding rectangle formed by taking the maximum x-coordinate, maximum y-coordinate, minimum x-coordinate, and minimum y-coordinate. Clearly, one of the five points (possibly more) will be contained in this bounding rectangle. Note that there is no concept containing the points on this rectangle but not the internal points. Thus the five points cannot be shattered.

Thus we have demonstrated that $\text{VCD}(\mathcal{C}) = 4$.

Generalizing to axis-parallel rectangles in $\mathbb{R}^d$, we get $\text{VCD}(\mathcal{C}) = 2d$.

Example 7.3 *Half-spaces in* $\mathbb{R}^2$

The concepts are half-spaces in $\mathbb{R}^2$ formed by a line dividing $\mathbb{R}^2$ into two half-spaces. Points lying in one of the half-spaces or on the dividing line are positive, and points lying in the other half-space are negative.

We first show that there exists a set of size three that can be shattered by $\mathcal{C}$. Consider any three non-collinear points. Clearly, they can be shattered by concepts from $\mathcal{C}$. Finally we must show that no set of size four can be shattered by $\mathcal{C}$. Given any four points, one of the following two cases occur.

1. At least three of the points are collinear. In this case, there is no half-space which contains the two extreme points, but does not contain the middle points. Thus clearly the four points cannot be shattered.

2. No three of the points are collinear. In this case, the points form a quadrilateral. There is no half-space which labels one pair of diagonally opposite points positive, and the other pair of diagonally opposite points negative. Thus clearly the four points cannot be shattered.

Thus we have demonstrated that $\text{VCD}(\mathcal{C}) = 3$.

Generalizing to half-spaces in $\mathbb{R}^d$, we get $\text{VCD}(\mathcal{C}) = d + 1$.

Example 7.4 *Closed sets in* $\mathbb{R}^2$

The concepts are closed sets in $\mathbb{R}^2$. 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 $\mathcal{C}$. This is because the concepts can assume any shape in $\mathbb{R}^2$. Thus, the largest set that can be shattered by $\mathcal{C}$ is infinite.

Thus, $\text{VCD}(\mathcal{C}) = \infty$.

Example 7.5 *Convex d-gons in* $\mathbb{R}^2$

The concepts are convex polygons in $\mathbb{R}^2$ having d sides. All points lying in the convex d-gon or on the sides of the convex d-gon are positive, and all points lying outside the convex d-gon are negative.

We first show that there exists a set of $2d + 1$ points that can be shattered. Consider $2d + 1$ points evenly spaced around a circle. We claim that given any labeling of these points one can find a d-gon consistent with the labeling. If there are more negative points then use the positive points as the vertices of the d-gon. If there are more positive points use the tangents to the negative points as the edges.

Finally, we informally argue that no set of size $2d + 2$ points can be shattered. If the points are not in a circular arrangement then clearly they can't be shattered. And if there are in a circular arrangement switching between positive and negative points as one goes around the circle produces a labeling that cannot be obtained by any d-gon.

So, for this concept class, $\text{VCD}(\mathcal{C}) = 2d + 1$.

Generalizing to convex polygons in $\mathbb{R}^2$, we get $\text{VCD}(\mathcal{C}) = \infty$.

7.4 VC Dimension and Sample Complexity

In this section, we relate the VC dimension to the sample complexity required for PAC learning. First, we need a definition.

Definition 7.6 A concept class $\mathcal{C} \subseteq 2^X$ is *trivial* if $\mathcal{C}$ consists of one concept, or two disjoint concepts c_1 and c_2 such that $c_1 \cup c_2 = X$.

When $\mathcal{C}$ is trivial, it is clear that a sample size of at most 1 is required to learn $\mathcal{C}$. Now to the more general case.

Theorem 7.1 *Let $\mathcal{C}$ be a non-trivial, well-behaved* [3] *concept class.*

1. $\mathcal{C}$ is PAC learnable if and only if the VC dimension of $\mathcal{C}$ is finite [4]

[3] This relatively benign measure-theoretic assumption holds of all the concept classes we have seen. It is discussed in detail in the appendix of [12].

[4] This assumes that the learner is limited to static sampling. It also does not consider the time or sample complexity of the learner.

2. if $\text{VCD}(\mathcal{C}) = d$*, where* $d < \infty$*, any hypothesis from* $\mathcal{C}$ *that is consistent with*

$$m \geq \max\left(\frac{2}{\epsilon}\log\frac{2}{\delta}, \frac{8d}{\epsilon}\log\frac{13}{\epsilon}\right)$$

examples is ϵ*-good with probability* $\geq 1-\delta$*. The sample complexity is*

$$O\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{\text{VCD}(\mathcal{C})}{\epsilon}\ln\frac{1}{\epsilon}\right).$$

We will not prove this theorem here. It is given in detail in Blumer et al. [12]. However, we can note a consequence of the theorem. If $\text{VCD}(\mathcal{C})$ is finite, then $\mathcal{C}$ is PAC learnable with sample size $O\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{\text{VCD}(\mathcal{C})}{\epsilon}\ln\frac{1}{\epsilon}\right)$, and if $\text{VCD}(\mathcal{C})$ is infinite, then $\mathcal{C}$ is not PAC learnable at all. (See Topic 14 for a discussion of when dynamic sampling can be used to learn concept classes with infinite VC dimension.)

We now describe an information-theoretic lower bound that demonstrates that the above upper bound is almost tight.

Theorem 7.2 *For any concept class* $\mathcal{C}$ *with finite VC dimension, finding an* ϵ*-good hypothesis with probability* $\geq 1-\delta$ *requires*

$$\Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{\text{VCD}(\mathcal{C})}{\epsilon}\right)$$

examples.

Proof: We begin by proving that $\Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta}\right)$ examples are needed.

Consider a portion of X having ϵ-weight in the distribution D_X on X. The probability of not seeing an instance from that portion in one drawing is at most $(1-\epsilon)$. Thus the probability of not seeing an instance from that portion of X in m drawings is at most $(1-\epsilon)^m$. We want this probability to be at most δ. Thus we require that

$$(1-\epsilon)^m \leq \delta$$

Taking the natural logarithm of both sides, we get that

$$m\ln(1-\epsilon) \leq \ln\delta$$

Dividing by $\ln(1-\epsilon)$ which is negative, gives

$$m \geq \frac{\ln\delta}{\ln(1-\epsilon)}$$

Finally, since $\ln(1-\epsilon) < -\epsilon$ we can conclude that

$$\begin{aligned} m &\geq -\frac{\ln\delta}{\epsilon} \\ &= \frac{1}{\epsilon}\ln\frac{1}{\delta} \end{aligned}$$

Thus, $\Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta}\right)$ examples are needed so that we get an example from any portion of X having ϵ-weight with probability $\geq (1-\delta)$.

We now want to prove that $\Omega\left(\frac{\text{VCD}(\mathcal{C})}{\epsilon}\right)$ examples are needed.

Let $\text{VCD}(\mathcal{C}) = d$. We shall first prove that additional $\Omega(d)$ examples are needed. Then, we will improve it to $\Omega(\frac{d}{\epsilon})$.

Since $\text{VCD}(\mathcal{C}) = d$, there exists a shattered set of size d. Let $S = \{x_1, \ldots, x_d\}$ be such a set, and let D_S be a uniform distribution over S. Assume without loss of generality that $|\mathcal{C}| = 2^d$.

Run the following experiment.

1. Draw sample $Y = \{y_1, \ldots, y_m\}$ from D_S where $m \leq \frac{d}{2}$. Assume without loss of generality that $y_1 = x_1, y_2 = x_2, \ldots, y_m = x_m$.
2. Choose target c by flipping d fair coins. (Let $b_1, \ldots, b_d$ be the outcomes).
3. Run the PAC algorithm on the m pairs $(x_1, b_1), \ldots, (x_m, b_m)$ to output hypothesis h.
4. Measure the error of h.

Consider the following modified experiment.

1. Draw sample $Y = \{y_1, \ldots, y_m\}$ from D_S where $m \leq \frac{d}{2}$. Assume without loss of generality that $y_1 = x_1, y_2 = x_2, \ldots, y_m = x_m$.
2. Choose target c by flipping m fair coins. (Let $b_1, \ldots, b_m$ be the outcomes).
3. Run the PAC algorithm on the m pairs $(x_1, b_1), \ldots, (x_m, b_m)$ to output hypothesis h.
4. Flip $d - m$ coins to determine the rest of c.
5. Measure the error of h.

Both the experiments have the same results. However, it is easier to measure the expected error in the second experiment. On each point not in the sample Y, the probability of h being correct is $\frac{1}{2}$. Thus, the total expected error $\geq (\frac{d}{2})(\frac{1}{2}) = \frac{d}{4}$. This implies that ϵ and δ can no longer be chosen arbitrarily because we know that the total expected error is at least $\frac{d}{4}$. Thus, the PAC algorithm needs at least $\frac{d}{2}$ examples. This shows that we need an additional $\Omega(d)$ examples.

We now modify the upper bound to prove the $\Omega(\frac{d}{\epsilon})$ bound. Let $S' = \{x_1, \ldots, x_{d-1}\}$. Let $S = S' \cup \{x_0\}$ and D_S be the distribution on S. Let D_S put weight $1 - 2\epsilon$ on x_0 and weight $\frac{2\epsilon}{d-1}$ on each of $x_1, \ldots, x_{d-1}$. Let $\mathcal{C}'$ be the concept class obtained by taking the concepts that shatter S' and let x_0 be positive. So $|\mathcal{C}'| = 2^{d-1}$.

Any PAC algorithm will quickly learn that x_0 is positive, but it cannot ignore S' since $\sum_{x \in S'} D_S(x) = 2\epsilon > \epsilon$. So, the PAC algorithm must see at least half the examples in S'. With probability $\frac{2\epsilon}{d-1}$ of seeing any of the examples in S', the PAC algorithm would need at

least $\Omega(\frac{d}{\epsilon})$ examples to see $\frac{d}{2}$ of the examples in S'. This is because we want $\frac{d}{2}$ successful Bernoulli trials, each with $\frac{2\epsilon}{d-1}$ chance of success.

Thus combining this with the first part of the proof, we get that the PAC algorithm needs

$$\Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta}+\frac{\text{VCD}(\mathcal{C})}{\epsilon}\right)$$

examples to find an ϵ-good hypothesis with probability at least $(1-\delta)$. ■

7.5 Some Relations on the VC Dimension

In this section, we will see some inequalities describing the relationships between the VC dimension of two or more concept classes, the relationships between the VC dimension of a concept class and the size of the concept class, etc. These bounds are very useful to compute the VC dimension of complicated concept classes that are constructed with simpler concept classes for which the VC dimension is known.

If $\mathcal{C}_1 \subseteq \mathcal{C}_2$, then clearly

$$\text{VCD}(\mathcal{C}_1) \leq \text{VCD}(\mathcal{C}_2).$$

Next we consider the relation between the VC dimension of $\mathcal{C}$ and the cardinality of $\mathcal{C}$. Given $\text{VCD}(\mathcal{C})$ points, we need $2^{\text{VCD}(\mathcal{C})}$ concepts to shatter them. Thus,

$$|\mathcal{C}| \geq 2^{\text{VCD}(\mathcal{C})}$$

Taking the base-2 logarithm of both sides

$$\lg|\mathcal{C}| \geq \text{VCD}(\mathcal{C})$$

For finite $\mathcal{C}$, we had earlier shown that

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

examples were sufficient to learn $\mathcal{C}$. Note that using $\text{VCD}(\mathcal{C})$ instead of $\ln|\mathcal{C}|$ gives a better bound.

Consider the complement $\bar{\mathcal{C}}$ of the concept class $\mathcal{C}$, defined as $\bar{\mathcal{C}} = \{X - c : c \in \mathcal{C}\}$. Then

$$\text{VCD}(\bar{\mathcal{C}}) = \text{VCD}(\mathcal{C}).$$

Consider the union of two concept classes. That is, let $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ where $\mathcal{C}$, $\mathcal{C}_1$ and $\mathcal{C}_2$ are defined over the same instance space. Then,

$$\text{VCD}(\mathcal{C}) \leq \text{VCD}(\mathcal{C}_1) + \text{VCD}(\mathcal{C}_2) + 1.$$

Consider the concept class $\mathcal{C}_s$ formed by the union (or intersection) of up to s concepts from $\mathcal{C}$ where $\textsc{vcd}(\mathcal{C}) = d$. For the union, $\mathcal{C}_s = \{\cup_{i=1}^{s} c_i : c_i \in \mathcal{C}\}$, and similarly for intersection. Then, for all $s \geq 1, \textsc{vcd}(\mathcal{C}_s) \leq 2ds \lg(3s)$. Thus,

$$\textsc{vcd}(\mathcal{C}_s) = O(s \ln(s) \textsc{vcd}(\mathcal{C})).$$

Earlier, we have defined $\Pi_{\mathcal{C}}(S)$ as the distinct ways that $\mathcal{C}$ can classify the instances in S. We shall now bound $|\Pi_{\mathcal{C}}(S)|$ by $|S|$ and $\textsc{vcd}(\mathcal{C})$.

Definition 7.7 For all $d \geq 0, m \geq 0$,

$$\begin{array}{rcll} \Phi_d(m) & = & \sum_{i=0}^{d} \binom{m}{i} & \text{if } m \geq d \\ & = & 2^m & \text{otherwise} \end{array}$$

For concept class $\mathcal{C}$, let $\textsc{vcd}(\mathcal{C}) = d$ and let S be a set of m distinct instances, then

$$|\Pi_{\mathcal{C}}(S)| \leq \Phi_d(m).$$

There are some inequalities relating $\Phi_d(m)$ using combinatorial arguments and Stirling's approximation.

$$\Phi_d(m) = \Phi_{d-1}(m-1) + \Phi_d(m-1).$$

For all $d \geq 0$ and $m \geq 0$,

$$\Phi_d(m) \leq m^d + 1.$$

For all $d \geq 2$ and $m \geq 2$,

$$\Phi_d(m) \leq m^d.$$

For all $m \geq d \geq 1$,

$$\Phi_d(m) \leq 2\left(\frac{m^d}{d!}\right) \leq \left(\frac{em}{d}\right)^d.$$

Thus, whenever $\textsc{vcd}(\mathcal{C})$ is finite, then $\Pi_{\mathcal{C}}(m)$ grows only polynomially in m.

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 8: Representation Independent Hardness Results

Lecturer: Sally Goldman

8.1 Introduction

In Topic 3 we discussed a hardness result for learning k-term-DNF when the learner is restricted to use a hypothesis class of k-term-DNF. However, we then showed that k-term-DNF is learnable using the hypothesis class of k-CNF. While such *representation-dependent hardness 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. In this lecture we will briefly introduce a representation-independent hardness result for learning several simple 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. The material presented here is just a brief introduction to the paper, "Cryptographic Limitations on Learning Boolean Formulae and Finite Automata," by Michael Kearns and Leslie Valiant [26]. We only give the intuition behind the hardness results—no details are described here. For the complete proofs we refer to reader to the Kearns and Valiant paper. Also a more complete discussion of these results is contained in Chapter 7 of Kearns' thesis [27].

8.2 Previous Work

Before describing the results of Kearns and Valiant we first briefly review the only previously known representation-independent hardness result. This previous result follows from the work of Goldreich, Goldwasser and Micali [19]. Let $\text{CKT}_n^{p(n)}$ denote the class of Boolean circuits over n inputs with at most $p(n)$ gates and let $\text{CKT}^{p(n)} = \bigcup_{n \geq 1} \text{CKT}_n^{p(n)}$. Goldreich, Goldwasser, and Micali showed that if there exists a one-way function, then for some polynomial $p(n)$, $\text{CKT}^{p(n)}$ is not polynomially learnable by *any* polynomially evaluatable hypothesis class.

Observe that the most powerful (polynomially evaluatable) hypothesis that can be output by any polynomial-time learning algorithm is a hypothesis that is itself a polynomial-time algorithm (or equivalently a polynomial-size Boolean circuit). Thus we cannot find efficient learning algorithms for concept classes that do not have small circuits—no learning algorithm even has time to just write down the representation. Schapire [37] has formally shown that any representation class that is not polynomially evaluatable cannot be learned in

polynomial time. With this in mind, observe that the result of Goldreich, et al. shows that not everything with a small representation is efficiently learnable (assuming the existence of one-way functions). However, there is a large gap between the computational power of $\text{CKT}^{p(n)}$ and the classes for which we have efficient learning algorithms. Thus we would like to prove representation-independent hardness results for less powerful concept class such as the classes of Boolean formulas.

8.3 Intuition

In this section we describe the intuition behind the hardness results of Kearns and Valiant. We begin by very informally describing the type of cryptographic scheme on which this hardness result is based.

Suppose that Alice and Bob wish to communicate privately inspite of an eavesdropper Eve who has bugged the communication line between Alice and Bob. (We make no assumptions about Eve except that she has a polynomial bound on her computing resources.) Furthermore, we want a scheme which does not require Alice and Bob to meet privately ahead of time to set up their encoding scheme. Such encryption scheme can be devised using a *trapdoor function*. Informally, a trapdoor function is one that can be computed in polynomial time (i.e., it is easy to compute $f(x)$ on input x) but cannot be inverted in polynomial time (i.e., it is hard to compute x on input $f(x)$) unless one is the "creator" of the function and thus possesses a piece of "trapdoor" information that makes inversion possible in polynomial time.

Cryptographic schemes based on such trapdoor functions are known as *public-key cryptographic systems*. In such a scheme each user creates a trapdoor function f and publishes a program for computing f. (This program must reveal no information about f^{-1}.) Then anyone can send messages to the given user over the nonsecure communication line and only that user can decode such messages. So Alice and Bob can communicate with each other using such a scheme where they have both created their own trapdoor function. We say that this system is secure if Eve cannot do noticeably better than random guessing in trying to decode a bit that has been encoded using this encryption scheme. More formally, if

$$\Pr[\text{Eve predicts correct decoded bit}] \geq \frac{1}{2} + \frac{1}{p(n)}$$

for some polynomial $p(n)$ then the system is not secure.

We now describe how we can use the existence of such a public-key cryptographic system, such as RSA, to obtain a representation-independent hardness result for learning. For the hardness result view Eve as a learning algorithm. Since a program for f is available she can create any polynomially number of pairs of the form $(f(x), x)$ that she likes by simply choosing x and then computing $f(x)$. If we set $y = f(x)$ observe that such pairs have the form $(y, f^{-1}(y))$ and thus can be viewed as labeled examples of f^{-1}. Thus public-key cryptography assumes the existence of functions that are not learnable from examples. In fact, unlike the PAC learning model in which we ask the learner to be able to make

arbitrarily good predications, in the cryptographic scheme we only ask the learner, Eve, to make predictions that are noticeably better than random guessing. (This "weak learning" model will be discussed further in the next topic.)

Observe that in the learning problem described above f^{-1} is "simple" in the sense that it has a small circuit (determined by the trapdoor used for decoding.) So the theory of cryptography provides simple functions that are difficult to learn. Kearns and Valiant show how to refine the functions provided by cryptography to find the simplest functions that are difficult to learn. Using this basic approach (of course, with lots of details which we will not discuss here) they show that polynomial-sized Boolean formulas, and constant-depth threshold circuits are not even weakly learnable by any polynomially evaluatable hypothesis class. Then using a prediction-preserving reduction (described below) of Pitt and Warmuth [34] it follows that deterministic finite automata are also not learnable.

8.4 Prediction Preserving Reductions

Given that we now have a representation-independent hardness result (assuming the security of the 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. Such a complexity theory for predictability has been provided by Pitt and Warmuth [34]. They formally define a *prediction-preserving* reduction ($A \trianglelefteq B$) that enables a prediction algorithm for concepts of type B to solve a prediction problem for concepts of type A. This reduction consists of the following two mapping:

1. A polynomial time computable function f that maps unlabeled examples of A to unlabeled examples of B.

2. A function g that maps representations of A to representations of B. Furthermore, this mapping g need not be computable—it is only required to be length preserving within a polynomial.

Then given a polynomial prediction algorithm for concepts of type B one can use it to obtain a polynomial prediction algorithm for concepts of type A in the obvious manner. Thus if A is known not to be learnable then B also cannot be learnable. They describe several such reductions in their paper, one of which is a reduction from Boolean formula predictability to DFA predictability (Boolean formula $\trianglelefteq$ DFA). Thus since Kearns and Valiant have shown that Boolean formula are not predictable (under cryptographic assumptions) it immediately follows that DFA are not predictable.

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 9: Weak Learning

Lecturer: Sally Goldman *Scribe: Andy Fingerhut*

9.1 Introduction

In the first lecture we introduced the *probably approximately correct* (PAC) or *distribution free* model of learning. In this model, the learner is presented with examples which are randomly and independently drawn from an unknown but fixed distribution. The learner must, with arbitrarily high probability, produce a hypothesis that is arbitrarily close to the target concept.

As we saw in the last lecture, the representation-independent hardness results based on the security of various cryptographic schemes motivated the study of learning algorithms which do not attain arbitrarily high accuracy, but do just slightly better than random guessing (by an inverse polynomial in the size of the problem). This leads to the notion of *weak learning* which is the subject of this lecture. In particular we describe an algorithm to convert any weak learning algorithm into a PAC-learning algorithm. The material presented here comes from the paper, "The Strength of Weak Learnability," by Robert Schapire [37].

9.2 Preliminaries

We begin by formally defining the weak learning model. As in the PAC model, for a given size parameter n, there is a set of instances X_n. A *concept* c is a Boolean function $c : X_n \to \{0,1\}$. A *concept class* $C_n \subseteq 2^{X_n}$ is a set of concepts. The learner has access to a source EX of labeled examples. Each time EX is called, an example is drawn randomly and independently according to a fixed but unknown distribution D on X_n, and returned in unit time.

After drawing some examples and running for a time, the learning algorithm must output an *hypothesis* h. This is a polynomially (in n) evaluatable hypothesis which, when given an instance $v \in X_n$, returns a prediction. The hypothesis may be randomized (this is important for weak learning).

We write $\text{Prob}[\pi(v)]$ to denote the probability that the predicate π holds for a particular instance v. This probability may be between 0 and 1 because of randomness in evaluating $\pi(v)$. For example, think of $\pi(v)$ as the event that $h(v) = 1$ for some randomized hypothesis h. By $\text{Prob}_{v \in D}[\pi(v)]$ we denote the probability that, after drawing v randomly from distribution D, $\pi(v)$ holds. Thus, assuming that these two random events (i.e., choice of v and evaluation of $\pi(v)$) are independent, we have $\text{Prob}_{v \in D}[\pi(v)] = \sum_{v \in X_n} D(v)\text{Prob}[\pi(v)]$ where $D(v)$ denotes the probability of instance v being chosen under distribution D.

Finally, we can define the *error* of an hypothesis h on c under distribution D as

$$error_D(h) = \text{Prob}_{v \in D}[h(v) \neq c(v)].$$

If the $error_D(h) \leq \epsilon$, we say h is ϵ-close to c under D. The *accuracy* of h is one minus its error.

We say that a concept class C is *strongly learnable* (called simply learnable or PAC-learnable elsewhere) if there is an algorithm A such that, for all $n \geq 1$, for all target concepts $c \in C_n$, for all distributions D and parameters $0 < \delta, \epsilon \leq 1$, algorithm A, given n, δ, ϵ, and access to oracle EX, outputs a hypothesis h such that, with probability $\geq 1 - \delta$, $error_D(h)$ is at most ϵ. Algorithm A should run in time polynomial in n, $1/\epsilon$, and $1/\delta$.

To be *weakly learnable*, there must be a polynomial p in addition to the algorithm A such that, for all $n \geq 1$, $c \in C_n$, for all distributions D, and for all $0 < \delta \leq 1$, algorithm A, given n, δ, and access to oracle EX, 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 $1/\delta$.

9.3 The Equivalence of Strong and Weak Learning

It should be clear that if 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. In particular, Kearns and Valiant [26] have shown that under a uniform distribution monotone Boolean functions are weakly, but not strongly, learnable. Thus it will be important to take advantage of the requirement that the weak learning algorithm must work for all distributions. The remainder of this section will be devoted to proving the main result of these notes.

Theorem 9.1 *If concept class C is weakly learnable, then it is strongly learnable.*

Note that weak learning, even though it may be "weak", is not necessarily easy to do. Consider the simple-minded algorithm which draws and memorizes a polynomial $q(n, 1/\delta)$ number of examples. It then outputs a hypothesis h which looks up known results and otherwise flips a fair coin to determine the answer. Suppose that $|X_n| = 2^n$ and the distribution D is the uniform distribution. Then the $error_D(h) = (1/2 - q(n, 1/\delta)/2^n)$, which is larger than $(1/2 - 1/p(n))$ for any polynomial p. Thus it is non-trivial to devise an algorithm that weakly learns a concept class.

9.3.1 Hypothesis Boosting

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.

Suppose that we have a learning algorithm A which produces hypotheses with error at most α (for $\alpha < 1/2$). Simply running A twice to produce hypotheses h_1 and h_2 may not help at all—they may err in a very similar way in their classification. After obtaining h_1, we need a way to force the next hypothesis output to be more accurate. The answer lies in the power of A to work given any distribution D on the instances. If we give A a distribution D' such that the error of h_1 is exactly $1/2$, then the hypothesis h_2 produced must have an error $\leq \alpha$ on it, which means that h_2 has learned something about the target concept which h_1 did not. Unfortunately, the probability that h_1 and h_2 classify a random instance the same (their "overlap") may be very low; in this case, it would seem that h_1 and h_2 learned almost completely different parts of the D. To handle this, we ask A to learn a "tie-breaker" hypothesis h_3. This time, A is given a distribution D'' on the instances on which h_1 and h_2 disagree.

Thus to improve the accuracy of the weak learning algorithm A, the algorithm A' simulates A on distributions D, D', and D'' and outputs a hypothesis that takes the majority vote of the three hypothesis obtained. Of course, algorithm A' must use D to simulate the distributions D' and D''. We now describe how this task is achieved. These two modified distributions are created by *filtering* the original distribution. In other words, A' will draw samples from D and discard undesirable samples to obtain the desired distribution. While this requires additional samples (thus increasing both the time and sample complexity) it can be made to run in polynomial time. The key observation is that we will only apply the filter if the probability of obtaining an example which passes through the filter is $\Omega(\epsilon)$. The distribution D' given to learn h_2 is produced by the procedure EX_2 defined below. Likewise, the distribution D'' for learning h_3 is produced by EX_3.

$EX_2(EX, h_1)$
    flip fair coin
    if heads then return the first instance v from EX for which $h_1(v) = c(v)$
        else return the first instance v from EX for which $h_1(v) \neq c(v)$

$EX_3(EX, h_1, h_2)$
    return the first instance v from EX for which $h_1(v) \neq h_2(v)$

Thus we can now formally state the hypothesis boosting procedure as follows:

1. Run the weak learning algorithm using EX. Let h_1 be the hypothesis output.
2. Run the weak learning algorithm using $EX_2(EX, h_1)$. Let h_2 be the hypothesis output.
3. Run the weak learning algorithm using $EX_2(EX, h_1, h_2)$. Let h_3 be the hypothesis output.
4. Return $h = \text{MAJORITY}(h_1, h_2, h_3)$.

An important question to address is: How much better is the accuracy of h than the accuracy of h_1? Suppose that the given weak learning algorithm is guaranteed to output a

Figure 9.1: A graph of the function $g(x) = 3x^2 - 2x^3$.

α-close hypothesis. (So $error_D(h_1) \leq \alpha$.) We shall show that for the hypothesis h output by the above hypothesis boosting procedure, $error_D(h) \leq g(\alpha) = 3\alpha^2 - 2\alpha^3$. See Figure 9.1 for a graph of $g(\alpha)$. Observe that both 0 and 1/2 are fixed points for the function g. Thus, as must be the case, for this boosting procedure to improve the error of the hypothesis, $\alpha < 1/2$. Note that for small α, $error_D(h) = O(\alpha^2) << \alpha$ and thus the new hypothesis is much better than the initial hypothesis. On the other hand, as α nears 1/2 the improvement is not as significant. However, as we shall show the number of iterations required to improve the error to be at most ϵ is polynomial in n and $1/\epsilon$.

9.3.2 The Learning Algorithm

In this section we describe the complete learning algorithm for converting a weak learning algorithm into a strong learning algorithm. The algorithm is shown in Figure 9.2. The basic idea is to recursively boost the hypothesis. The lowest level of recursion only requires the accuracy that `WeakLearn` (the weak learning algorithm given) can provide. Each higher level produces hypotheses with higher accuracy, until it has been boosted to be at least $1 - \epsilon$.

A technical difficulty which arises is that we may get lucky early and obtain an ϵ-close hypothesis for h_1 or h_2. If this happens, we are in trouble since either EX_2 or EX_3 would take too long. Therefore, the algorithm performs some hypothesis testing to prevent this from happening.

9.3.3 Correctness

We now prove that the algorithm of Figure 9.2 will produce a hypothesis meeting the PAC criterion.

`Learn`(ϵ, δ, EX)

Input:
- ϵ - allowed error
- δ - confidence desired
- EX - oracle for examples
- n - (implicit) size

Output:
- h - hypothesis that with probability $\geq 1 - \delta$ is ϵ-close to target

Procedure:

if $\epsilon \geq (1/2 - 1/p(n))$ then return `WeakLearn`(δ, EX)
else $\alpha := g^{-1}(\epsilon)$
  $h_1 :=$ `Learn`$(\alpha, \delta/5, EX)$
  estimate $error_D(h_1)$ to within $\epsilon/3$ of true value with confidence $\geq 1 - \delta/5$
  if h_1 is ϵ-good then return h_1
  else construct EX_2
    $h_2 :=$ `Learn`$(\alpha, \delta/5, EX_2)$
    estimate $error_D(h_2)$ to within $(1 - 2\alpha)\epsilon/8$ of true value with confidence $\geq 1 - \delta/5$
    if h_2 is ϵ-good then return h_2
    else construct EX_3
      $h_3 :=$ `Learn`$(\alpha, \delta/5, EX_3)$
      return $h = MAJ(h_1, h_2, h_3)$
    endif
  endif
endif

Figure 9.2: An algorithm to convert a weak learning algorithm into a strong learning algorithm.

Theorem 9.2 *If $0 < \epsilon < 1/2$ and $0 < \delta \leq 1$, then with probability $\geq 1 - \delta$, the hypothesis returned by calling* `Learn`(ϵ, δ, EX) *is ϵ-close to the target c under D.*

Proof: Define a *good run* of `Learn` to be one in which everything goes right. That is, every call of `WeakLearn` produces a hypothesis with the required accuracy, and every error estimation is within its prescribed tolerance of being correct. We first show that the probability of having a good run is $\geq 1 - \delta$. We then show that whenever there is a good run, the hypothesis returned is ϵ-close to the target. These two facts imply the theorem.

We now argue by induction on the depth of the recursion that the probability of having a good run is at least $1-\delta$. The base case is depth $= 0$, which occurs when $\epsilon \geq (1/2 - 1/p(n))$. In this case, the allowed error is high enough that `WeakLearn` can handle the situation. It produces a good run with probability $\geq 1-\delta$. The induction step occurs if $\epsilon < (1/2-1/p(n))$. In this case, there are (at most) three recursive calls to `Learn` and (at most) two error estimations. In the worst case, all of the calls are performed. Since each of the calls and estimations succeeds (independently) with probability $\geq 1 - \delta/5$ (the calls succeed with this probability by the induction hypothesis), the probability that they all succeed is $\geq 1 - \delta$.

From now on, we assume that we have a good run, and our goal is to prove that the returned hypothesis is ϵ-good. We again proceed by induction on the depth of recursion. The base case is trivial, assuming a good run, since the call to `WeakLearn` produces an ϵ-good hypothesis.

The inductive step is trickier. In his paper Schapire gives a very detailed and formal proof—what follows here is a simpler but less formal (i.e. important details are ignored) proof.

If either h_1 or h_2 is tested to be ϵ-good, then it is returned and the Theorem is proved. If $h = MAJ(h_1, h_2, h_3)$, then there are two cases for h to be incorrect.

1. Both h_1 and h_2 give the wrong answer. Since this is a good run, $error_{D'}(h_2) \leq \alpha$. (Recall that D' is the distribution of examples given by EX_2). To make the probability of $h_1 = h_2 \neq c$ as high as possible, all of this error should coincide with h_1's error. With the filtering that occurs to get from D to D', the portion on which h_1 is incorrect expands from probability α to $1/2$. If h_2 has all of its error on that $1/2$ portion, it shrinks by multiplying it by 2α when translated back into D. Therefore h_2's error on D is $\leq 2\alpha(\alpha) = 2\alpha^2$.

2. The hypotheses h_1 and h_2 give different answers, and h_3 breaks the tie incorrectly. Since h_3 is the deciding vote, it should be wrong as often as possible to make the overall error large. Therefore we should make the distribution D'' (produced by EX_3) "cover" as much of the original distribution D as possible. This is done by making h_1 and h_2 disagree as often as possible. So, make h_2 wrong on the portion α of D' on which h_1 is correct. When translating back to D, this portion expands by a factor of $2(1-\alpha)$. Therefore, h_2's error on D is $\leq 2(1-\alpha)\alpha = 2\alpha - 2\alpha^2$, and the portion of D on which exactly one of h_1 and h_2 is wrong is $\leq 2\alpha - 2\alpha^2 + \alpha = 3\alpha - 2\alpha^2$. Thus h_3 can be wrong on a fraction α of that, so the error of h_3 on D is $\leq 3\alpha^2 - 2\alpha^3$.

The above two cases are mutually exclusive, and they exhaust all possibilities that h is wrong. Case (1) covers when h_1 and h_2 agree as often as possible, and case (2) covers when they disagree as often as possible. Thus h's real error will be somewhere in between these values. The larger value (when $0 \leq \alpha \leq 1/2$) is $g(\alpha) = 3\alpha^2 - 2\alpha^3 = \epsilon$, proving the theorem. ■

9.3.4 Analysis of Time and Sample Complexity

Although in the previous section we proved that `Learn` is correct, we have said nothing about its time or sample complexity. To complete the proof that strong and weak learning are equivalent, we must prove that `Learn` has time and sample complexity that are polynomial in n, $1/\epsilon$, and $1/\delta$.

First, consider the shape of the recursion tree obtained by the recursive calls occurring during `Learn`'s execution. If there are no early returns caused by serendipitously discovering an ϵ-good hypothesis, then the recursion tree is a rooted full ternary tree. Every node has either zero or three children, and every leaf occurs at the same depth. Denote this depth by $B(\epsilon, p(n))$, since it depends only on the arguments given.

If early returns do occur, then a node at depth less than B will have either one or two children and their corresponding subtrees "clipped". It is simpler to ignore such cases when determining worst-case time complexity, since the run time is greatest when there is no clipping.

The number of leaves (which correspond with calls to `WeakLearn`) is exactly 3^B. Therefore we would like to bound B to be logarithmic in n, $1/\epsilon$, and $1/\delta$. That is the result of Lemma 9.1 below.

At each level of recursion, ϵ is replaced by $g^{-1}(\epsilon)$, with $\epsilon \geq (1/2 - 1/p(n))$ at the bottom level. Therefore

$$B(\epsilon, p(n)) = \min_{i \geq 0} \left\{ g^i \left(\frac{1}{2} - \frac{1}{p(n)} \right) \leq \epsilon \right\}.$$

Note that $g^i(\alpha)$ is monotonically increasing with α for $0 \leq \alpha \leq 1/2$ for all i, and $g(\alpha) \leq \alpha$ on the same range.

Lemma 9.1 $B(\epsilon, p(n)) = O(\log(p(n)) + \log\log(1/\epsilon))$

Proof: Note that if $g^b(1/2 - 1/p(n)) \leq d$ and $g^c(d) \leq \epsilon$ for some values of b, c, and d, then $B(\epsilon, p(n)) \leq a + b$.

For $0 \leq x$, it is clear that $g(x) \leq 3x^2$. From this it can be proved easily by induction on i that $g^i(x) \leq (3x)^{2^i}$ for all $0 \leq x$. Therefore $g^c(1/4) \leq \epsilon$ if $c \geq \lg\log_{4/3}(1/\epsilon)$.

If $1/4 \leq x \leq 1/2$, then $1/2 - g(x) = (1/2 - x)(1 + 2x - 2x^2) \geq (11/8)(1/2 - x)$ (Note that $11/8$ is the minimum of $(1 + 2x - 2x^2)$ on the given interval). It can be proved by induction on i that $g^i(x) \geq (11/8)^i(1/2 - x)$, as long as x, $g(x)$, ..., $g^{i-1}(x)$ are all at least $1/4$. Therefore $g^b(1/2 - 1/p(n)) \leq 1/4$ if $b \geq \log_{11/8}(p(n)/4)$. ■

Quantities which are relevant to the time and sample complexity of `Learn` are $T(\epsilon, \delta)$, the expected running time of `Learn`(ϵ, δ, EX), $U(\epsilon, \delta)$, the time needed to evaluate a hypothesis

returned by `Learn`, and $M(\epsilon, \delta)$, the expected number of samples needed by `Learn`. There are corresponding quantities $t(\delta)$, $u(\delta)$, and $m(\delta)$ for `WeakLearn`(δ, EX). All of these functions also depend (implicitly) on n.

These functions can be bounded quite effectively by the use of recurrence relations. These recurrences can be obtained by examination of `Learn`, with the base case being `WeakLearn` and its complexity. The final results will be mentioned here as lemmas, with proofs omitted. See Schapire's paper for the complete proofs.

Lemma 9.2 *The time to evaluate a hypothesis returned by* `Learn`(ϵ, δ, EX) *is* $U(\epsilon, \delta) = O(3^B \cdot u(\delta/5^B))$.

Lemma 9.3 *Let r be the expected number of examples drawn from EX by any oracle EX_i simulated by* `Learn` *on a good run when asked to provide a single example. Then $r \leq 4/\epsilon$.*

Lemma 9.4 *On a good run, the expected number of examples $M(\epsilon, \delta)$ needed by* `Learn`(ϵ, δ, EX) *is*

$$O\left(\frac{36^B}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right)$$

Lemma 9.5 *On a good run, the expected execution time of* `Learn`(ϵ, δ, EX) *is given by*

$$T(\epsilon, \delta) = O\left(3^B \cdot t(\delta/5^B) + \frac{108^B \cdot u(\delta/5^B)}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right)$$

The fact that nearly all of the expected values above are only taken over good runs of `Learn` can be taken into account by "borrowing" some of the confidence δ for that purpose.

9.4 Consequences of Equivalence

There are many very interesting consequences that follow from this main result (Theorem 9.1. We shall just briefly mention a few here. We refer the reader to Schapire's paper for more details on these consequences and a discussion of other interesting corollaries.

One of the more notable corollaries of Theorem 9.1 is a partial converse of Occam's Razor (see Topic 6). That is, if a concept class is learnable, then any sample can be compressed with high probability. More specifically, if C is PAC learnable then there exists a polynomial time algorithm which with confidence $1 - \delta$ outputs a consistent hypothesis of size polynomial in n and $/logm$ for any sample of size m labeled according to a concept c/inC. This corollary leads to the result that any representation class that is not polynomially evaluatable cannot be learned in polynomial.

Another interesting result discussed is an improved version of the learning algorithm given here, which achieves asymptotic performance which is poly-logarithmic in terms of $1/\epsilon$. Thus using this improved version of `Learn` an existing strong learning algorithm performance can be made to be poly-logarithmic in terms of 1ϵ by first fixing the error of the given strong learning algorithm at, say, $\epsilon = 1/4$ to obtain a weak learning algorithm. Then use this improved version of `Learn` to convert the weak learning algorithm back into a strong learning algorithm.

CS 582T Computational Learning Theory **Spring Semester, 1998**

Topic 9.5: Bayesian Learning

Lecturer: Stephen Scott Scribe: Sanjin Loncaric

9.5.1 Introduction

Bayes learning algorithms that calculate explicit probabilities for hypotheses (like the naive Bayes classifier), are among the most practical approaches to certain types of learning problems (for ex. learning to classify text documents such as electronic news articles.) They combine prior knowledge with observed data. Unlike PAC learning, Bayesian learning looks at the average performance (with respect to assumptions on the hypothesis class) instead of worst-case performance. Bayesian methods also provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities. One of such algorithms is the Monomial algorithm covered in section 1.4 of the scribe notes. In this way they provide the basis for developing general purpose algorithms.

One practical difficulty in applying Bayesian methods is that they typically require initial knowledge of many probabilities. Another difficulty is the significant computational cost required to determine the Bayes optimal hypothesis in the general case.

In this discussion, we will be interested at the probability that a given hypothesis h is correct (i.e. equal to a target function) given some observed data. We will cover Bayes theorem, maximum a posteriori hypothesis (h_{MAP}), maximum likelihood hypothesis (h_{ML}), different h_{MAP} learners, Bayes optimal classifier, Gibbs algorithm and the naive Bayes learner. The material presented here is taken from my lecture notes and chapter 6 of *Machine Learning*, by Tom Mitchell.

9.5.2 Bayes theorem

Since we are going to deal with probabilities here, let us first introduce some notation. P(h) will denote the initial probability (before we observe the training data) that hypothesis h holds. It is often called the *prior probability* of h. Similarly, P(D) will denote the prior probability that training data D will be observed, given no knowledge about which hypothesis holds. For convenience, we can think of D as labels of a set of instances fixed in advance. P(D|h) will denote the probability of observing data D given that hypothesis h holds. Similarly, P(h|D) will be the probability that h holds given the observed training data D. This probability is also called the *posterior probability* of h. Bayes theorem can now be expressed as:

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)} \quad .$$

When we choose a hypothesis, we usually want the most probable hypothesis given our data D. Any such maximally probable hypothesis is called a *maximum a priori* (MAP) hypothesis. We can determine the MAP hypothesis by using Bayes theorem:

$$h_{MAP} \equiv \arg\max_{h \in H} P(h \mid D)$$

$$= \arg\max_{h \in H} \frac{P(D \mid h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D \mid h)P(h)$$

where H is a set of hypotheses.

In the final step above we dropped the term P(D) because it is a constant independent of *h*. In cases where $P(h_i) = P(h_j)$ for all h_i and h_j in H, we need to consider the term P(D|*h*) only, to find the most probable hypothesis. Any hypothesis that maximizes P(D|*h*) (the *likelihood* of the data D given *h*) is called a *maximum likelihood* (ML) hypothesis, h_{ML}:

$$h_{ML} \equiv \arg\max_{h \in H} P(D \mid h) .$$

9.5.3 Brute-Force MAP Learning algorithm

- For each hypothesis *h* in H, compute posterior probability and output the one that maximizes P(*h*|D).

This algorithm may require significant computation, especially for large hypothesis spaces (for ex. consider axis parallel rectangles in a plane). However, the algorithm is still of interest because it provides a standard against which we may judge the performance of other concept learning algorithms.

The remainder of this section studies when certain learning algorithms output MAP hypotheses. We make the following assumptions:

1. The training data D is noise free
2. The target concept *c* is contained in the hypothesis space H
3. We have no a priori reason to believe that any hypothesis is more probable than any other.

Given these assumptions what values should we specify for P(*h*) and P(D|*h*)? It is reasonable to assign the same prior probability to every hypothesis *h* in H and also we should require that they sum to 1. Therefore:

$$P(h) = \frac{1}{|H|} \qquad \text{for all } h \text{ in } H.$$

Also, given a world in which hypothesis *h* holds, the probability of observing classification d_i given *h* is just 1 if $d_i = h(x_i)$ and 0 if $d_i \neq h(x_i)$. Therefore:

$$P(D \mid h) = \begin{Bmatrix} 1 \text{ if } d_i = h(x_i) \text{ for all } d_i \text{ in } D \\ 0 \text{ otherwise} \end{Bmatrix} .$$

Now we consider the first part of the Brute-force algorithm: computing the posterior probability P(*h*|D) of each hypothesis *h* given the observed training data D. First we look at the case where *h* is inconsistent with D. Above, we showed that P(D|*h*) = 0 when *h* is inconsistent with D. So, using Bayes theorem we have:

$$P(h \mid D) = \frac{0 \cdot P(h)}{P(D)} = 0 \qquad \text{if } h \text{ is inconsistent with } D.$$

Now, we look at the case where *h* is consistent with D. Above, we define P(D|*h*) to be 1 when *h* is consistent with D. We have:

$$\begin{aligned} P(h \mid D) &= \frac{1 \cdot \frac{1}{|H|}}{P(D)} \\ &= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} \\ &= \frac{1}{|VS_{H,D}|} \qquad \text{if } h \text{ is consistent with } D \end{aligned}$$

where $VS_{H,D}$ is the subset of hypotheses from H that are consistent with D (the *version space* of H given D). Now it is easy to derive the expression for P(D), because the sum over all hypotheses of P(*h*|D) must be 1 and because the number of hypotheses from H consistent with D is by definition $|VS_{H,D}|$. Therefore,

$$P(D) = \frac{|VS_{H,D}|}{|H|}.$$

From the above analysis we can conclude that in a given setting (uniform prior probabilities over H and noise-free data) every consistent hypothesis is a MAP hypothesis. Also, a learning algorithm is a *consistent learner* if it outputs a hypothesis that commits no errors over the training examples. Every consistent learner outputs a MAP hypothesis given a uniform prior distribution and deterministic, noise-free training data. Since all PAC algorithms can be made to output consistent hypotheses with high probability, all PAC algorithms are MAP learners under our assumptions, including for ex.: monomials, k-CNF, unions of boxes, etc.

However, MAP hypothesis is not the best that we can do. Consider the next example:

H={h_1, h_2, h_3}
P(h_1|D) = 0.4, P(h_2|D) = 0.3, P(h_3|D) = 0.3
For an instance x: h_1(x) = +, h_2(x) = -, h_3(x) = -
Most probable label for x is therefore -, but MAP outputs + .

9.5.4 Bayes optimal classifier

To improve upon MAP algorithms we will now maximize over *classifications*, not probabilities. In general, the most probable classification of the new instance is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities. If the possible classification of the new example can take on any value v_j from some set V, then the maximum probability P(v_j|D) of the correct classification over all instances v_j can be expressed as:

Bayes optimal classification:

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j \mid h_i) P(h_i \mid D).$$

Any system that classifies new instances according to this classification is called a *Bayes optimal classifier* (or Bayes optimal learner.) Our example above now becomes:

P(h_1|D) = 0.4, P(-|h_1) = 0, P(+|h_1) = 1
P(h_2|D) = 0.3, P(-|h_2) = 1, P(+|h_2) = 0
P(h_3|D) = 0.3, P(-|h_3) = 1, P(+|h_3) = 0 .

So,

$$\sum_{h_i \in H} P(+ \mid h_i) P(h_i \mid D) = 0.4 \ ,$$

$$\sum_{h_i \in H} P(- \mid h_i) P(h_i \mid D) = 0.6 \ ,$$

and

$$\arg\max_{v_j \in \{+,-\}} \sum_{h_i \in H} P(v_j \mid h_i) P(h_i \mid D) = - \quad .$$

No other classifier, given the same H and D and prior distribution, can do better on average than the Bayes optimal classifier. Note that it does not make any guarantees outside the data and probabilities given, unlike the PAC model.

9.5.5 Gibbs algorithm

Although the Bayes optimal classifier obtains the best performance that can be achieved from the given training data, it can be quite costly to apply, due to computing the posterior probability for every hypothesis in H and then combining the predictions of each hypothesis to classify each new instance. An alternative is a less optimal method called the *Gibbs algorithm*, defined as:

1. Choose one hypothesis from H at random according to P(*h*|D)
2. Use this *h* to predict the classification of the new instance

It can be shown that the average error of he Gibbs algorithm is guaranteed to be **at most twice** that of the Bayes optimal algorithm (Haussler et al. 1994). If we have a uniform prior hypothesis and noise-free data, then the Gibbs algorithm is as follows:

1. Pick any hypothesis from $VS_{H,D}$ (uniformly at random)
2. Use this *h* to predict the classification of the new instance

Classifying the next instance according to a hypothesis drawn at random from the current version space (according to a uniform distribution), will have expected error at most twice that of the Bayes optimal classifier:

$$E[ERROR_{Gibbs}] \leq 2E[ERROR_{\text{Bayes optimal}}] \ ,$$

assuming a uniform prior and noise-free data. Note that this still leaves the algorithmic problem of selecting a hypothesis from $VS_{H,D}$ uniformly at random.

Here are some other results (taken from Haussler et al. 1994) relating Gibbs and bayes algorithms:

The probabilities of mistake for both the Bayes and the Gibbs algorithms are between $H^{-1}(E_{f\in P}[I_{m+1}(f)])$ and $1/2E_{f\in P}[I_{m+1}(f)]$, where f is the target concept, P is the posterior distribution and I is the information gain. These upper and lower bounds are equal (and therefore tight) at both extremes $E_{f\in P}[I_{m+1}(f)]=1$ (maximal information gain) and $E_{f\in P}[I_{m+1}(f)]=0$ (minimal information gain). As the information gain becomes smaller, the difference between the upper and lower bounds shrinks, but the ratio of the two bound grows logarithmically in the inverse of the information gain.

We can also tie Bayesian analysis with the VC dimension to give bounds on the instantaneous probability of mistakes that are independent of the prior P and the distribution D on the instance space X. We have that for all P,

$$E_{f\in P, x\in D^*}[Bayes_m(x, f)] \le E_{f\in P, x\in D^*}[Gibbs_m(x, f)]$$

$$\le (1+o(1))\frac{\dim(F)}{2m}\log\frac{m}{\dim(F)}$$

where we use the notation $x\in D^*$ to indicate that each element of the infinite sequence *x* is drawn independently according to D and we use dim(F) to denote the VC dimension of the concept class F.

It can be shown that the bound given by the above equation is tight to within a factor of $1/\ln(2) \approx 1.44$ and hence cannot be improved by more than this factor in general. It also follows that the expected total number of mistakes of the Bayes and the Gibbs algorithms differ by a factor of at most about 1.44.

In case of an incorrect prior, our best lower bounds for both instantaneous mistake probability and the cumulative number of mistakes are obtained by observing that the mistake probability is minimized by the Bayes algorithm when P = Q (where Q is the *true prior* an P is *the perceived prior*). Thus

$$c_0 E_{f\in Q}[I^Q_{m+1}(f)]/\log(2/E_{f\in Q}[I^Q_{m+1}(f)]$$

is a lower bound on the instantaneous mistake probability, and

$$c_0 H^Q_m/\log(2m/H^Q_m)$$

is a lower bound on the cumulative number of mistakes for both the Bayes and Gibbs algorithms, for any perceived prior P.

9.5.6 Naïve Bayes classifier

This is a highly practical Bayesian learning method with high performance. The naïve Bayes classifier applies to moderate and large training sets and to attributes that are conditionally independent given the classifications. The Bayesian approach to classifying the new instance is to assign the most probable target value, v_{MAP}, given the attribute values $<a_1, a_2 \ldots a_n>$ that describe the instance:

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j \mid a_1, a_2 ... a_n) = \arg\max_{v_j \in V} \frac{P(a_1, a_2 ... a_n \mid v_j) P(v_j)}{P(a_1, a_2 ... a_n)}$$

$$= \arg\max_{v_j \in V} P(a_1, a_2 ... a_n \mid v_j) P(v_j) .$$

Now we need to estimate the two terms in the final expression above based on the training data. However, we can't realistically have any estimates on many of these terms because it is very unlikely we have seen a specific example before. Therefore, we need to see every instance in the instance space many times in order to obtain reliable estimates.

The naïve Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, given the target value of the instance, the probability of observing the conjuction a_1, $a_2 \ldots a_n$ is just the product of the probabilities for the individual attributes: $P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$. Therefore we have:

Naïve bayes classifier:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i \mid v_j) ,$$

where v_{NB} denotes the target value output by our classifier.
Now we can express the naïve Bayes algorithm using the classifier we stated above,

Alg:

for each label v_j

$\hat{P}(v_j) = estimate\ p(v_j)$

for each attribute value a_i *of each attribute,* $\hat{p}(a_i \mid v_j) = estimate\ p(a_i \mid v_j)$

where *estimate* is typically evaluated by counting the occurences of v_j and $a_i | v_j$ in training data.

If conditional independence holds, this naïve Bayes classification is identical to the MAP classification. However, technically we don't necessarily need to have the independence assumption hold to get MAP classification. If independence assumption holds, we will get correct posterior probabilities, but even if the assumption doesn't hold, if

$$\arg\max_{v_j \in V} \hat{p}(v_j) \prod_{i=1}^{n} \hat{p}(a_i \mid v_j) = \arg\max_{v_j \in V} p(v_j) P(a_1, a_2 ..., a_n \mid v_j) ,$$

then naïve Bayes still outputs a MAP hypothesis. This situation occurs when H is the class of *monomials* or class of *disjunction of literals* (Domingos and Pazzani 1996).

To summarize, the naïve Bayes learning method involves a learning step in which the various $P(v_j)$ and $P(a_i|v_j)$ terms are estimated, based on their frequencies over the training data. The set of these estimates corresponds to the learned hypothesis, which is then used to classify each new instance (using the classifier). Also, the hypothesis is formed without searching, simply by counting the frequency of various data combinations within the training examples.

References:

David Haussler, Michael Kearns, and Robert Schapire. Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension. Machine Learning, 14(1):83-113, 1994. Short version in COLT '91.

Machine Learning, Tom Mitchell, McGraw Hill, 1997.

Pedro Domingos and Michael Pazzani. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In Proceedings of ICML '96: The 13th International Conference on Machine Learning, pp. 105-112, Bari, Italy, July 1996.

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 10: Learning With Queries

Lecturer: Sally Goldman *Scribe: Kevin Ruland*

10.1 Introduction

In these notes we study the model of learning with queries. The material presented in this lecture comes from the paper, "Queries and Concept Learning," by Dana Angluin [3]. In the model of learning with queries, the learner must identify an unknown hypothesis L_* from some countable concept space of subsets of a universal set of instances. Unlike in the PAC model, the algorithm is not given a stochastically generated sequence of labeled instances. It is instead allowed to actively explore its environment by asking questions of an *oracle*.

Throughout these notes we use the following notation.

- $\mathcal{C} = \{L_1, L_2, \ldots\}$ is the set of concepts
- L_* is the target concept
- U is the instance space

Angluin [3] defines many different kinds of queries. We now define the types of queries we will consider here.

Membership Query The oracle will respond to input $x \in U$ "yes" if $x \in L_*$, otherwise it replies "no".

Equivalence Query The oracle will respond to input hypothesis $h \in \mathcal{C}$ "yes" if $h = L_*$, otherwise it will return some counterexample $x \in h \oplus L_*$. The counterexample returned by the oracle is selected by an all-powerful adversary, and no assumptions can be made about it.

Restricted Equivalence Query The oracle will respond to input hypothesis $h \in \mathcal{C}$ "yes" if $h = L_*$, otherwise it responds "no." It does *not* return a counterexample.

Generalized Equivalence Query The oracle will respond as an equivalence query to any input hypothesis that is a subset of U.

Unlike the PAC model in which the learner need only output a hypothesis that is a good approximation to the target concept, in the model of learning with queries, the learner must achieve *exact identification*. We say an algorithm *exactly identifies* the target concept L_* with access to certain types of queries if it always halts and outputs a concept equivalent to L_*. That is, for each element of $x \in U$ the concept output classifies x as L_* does.

10.2 General Learning Algorithms

In this section we explore some generic algorithms for learning with membership and equivalence queries. Then in the next section, we explore the relationship between this learning model and the PAC model. Finally, we will consider learning algorithms for particular concept classes.

10.2.1 Exhaustive Search

The most trivial algorithm that uses only equivalence queries is exhaustive search where each concept in $\mathcal{C}$ is tried in turn until the correct hypothesis is found. This method clearly requires, in the worst case, $|\mathcal{C}| - 1$ equivalence queries. It is interesting to note that for some classes this algorithm is no worse than any other. For example, consider the *singletons*, where for fixed instance space U, $\mathcal{C} = \{ \{x\} \mid x \in U \}$. In this class, an adversary can respond to an equivalence query by returning the element in the hypothesis as the counterexample. This strategy allows the learner to eliminate at most one concept for each equivalence query. Also, $\mathcal{C}$ even requires $|\mathcal{C}| - 1$ membership queries because the adversary could reply "no" to each query.

This observation is generalized in the following lemma.

Lemma 10.1 *Suppose $\mathcal{C} = \{L_1, \ldots, L_N\}$, and there exists a set $L_\cap \notin \mathcal{C}$ such that*

$$L_i \cap L_j = L_\cap, \qquad i \neq j$$

Then any algorithm that achieves exact identification for any $L_* \in \mathcal{C}$ must make at least $N - 1$ equivalence and membership queries in the worst case.

Proof: We must exhibit an adversary which answers each query in such a way that only a single concept can be eliminated. Keep in mind that the adversary can choose the target concept as the learning session progresses. Namely, the adversary will keep a list of target concepts that are consistent with all previous queries. The learner cannot achieve exact identification until only one concept remains in the adversary's list.

For a membership query on instance x, if $x \in L_\cap$ then reply "yes," otherwise reply "no." In the first case, no concepts from the adversary's list are eliminated because x is in every concept. In the second, at most one concept, the concept containing x, is eliminated—if two were eliminated, L_i and L_j, then $L_i \cap L_j \supseteq L_\cap \cup \{x\}$, a contradiction.

For an equivalence query on hypothesis $L \neq L_*$, reply "no" and return any $x \in L \cap L_\cap$. Such an x exists since $L_\cap \notin \mathcal{C}$, so $L \cap L_\cap \neq \emptyset$. Also, if $x \in L_\cap$ then no concepts are eliminated, and if $x \notin L_\cap$ then at most one concept is eliminated. ■

There is a dual result for union.

10.2.2 The Halving Algorithm

If the learner is allowed a more powerful equivalence query, better results are obtainable. Recall that the *generalized equivalence query* is similar to the equivalence query but allows

as input any subset of U. Using such a query, any finite concept class can be learned in no more than $\lfloor \lg |\mathcal{C}| \rfloor$ queries using the following algorithm.

Halving Algorithm($\mathcal{C}$)
If $\mathcal{C}$ contains one element L, then $L = L_*$. Halt.
Else
  Let $M_{\mathcal{C}} = \{ x \mid x \in \mathcal{C}$ for at least $\frac{|\mathcal{C}|}{2}$ concepts $\}$.
  Make generalized equivalence query with hypothesis $M_{\mathcal{C}}$
  If "yes" then $M_{\mathcal{C}}$ is equivalent to L_*. Halt and output $M_{\mathcal{C}}$.
  Else let x be the counterexample.
    If $x \in M_{\mathcal{C}}$ then $\mathcal{C}' = \mathcal{C} \setminus \{ L \in \mathcal{C} \mid x \in L \}$
    Else $\mathcal{C}' = \{ L \in \mathcal{C} \mid x \in L \}$
    Halving Algorithm($\mathcal{C}'$)

Theorem 10.1 *The number of queries performed by the halving algorithm is at most* $\lfloor \lg |\mathcal{C}| \rfloor$.

Proof: A query is made if $|\mathcal{C}| > 1$ and every query eliminates at least half of the concepts. ■

The upper bound is often not tight. For example, for the class of singletons the halving algorithm makes only one mistake versus the upper bound of $\lg n$ given in Theorem 10.1. Littlestone [29] gives an algorithm whose performance on any class is shown to be optimal. (This algorithm is discussed in Topic 12.)

A major drawback of the halving algorithm is that it is often not computationally feasible—often exponential computation time is needed to construct $M_{\mathcal{C}}$. However, it can sometimes be efficiently implemented. As an example note that Valiant's algorithm for learning k-CNF [43] indirectly implements the halving algorithm. Let h be defined as the conjunction of all clauses that have been true in all counterexamples. (Recall that if $h \subseteq L_*$, then h is consistent with all negative examples and thus for any counterexample x to h, $L_*(x) = +$.) Now let $\mathcal{C}' = \{$ concepts consistent with all counterexamples $\}$, so $\mathcal{C}'$ consists of every k-CNF formula that is a conjunction of some subset of the clauses in h. The hypotheses h constructed in this manner predicts according to the halving algorithm. If for a particular instance all clauses in h are true, the majority is true. Similarly, if for an instance some clause l in h is false, for every $L \in \mathcal{C}'$ that is true, there is some other hypothesis, $L' \in \mathcal{C}'$ (namely $L \wedge l$), that is false. So, if ties are false, the majority is false.

See Topic 20 for a discussion on how approximate counting schemes can be used to efficiently implement a variant of the halving algorithm.

10.3 Relationship Between Exact Identification and PAC Learnability

In this section we describe how an algorithm that uses equivalence queries can be converted to a PAC-learning algorithm. We then give an example to demonstrate the converse is not true.

Theorem 10.2 *Any algorithm that uses equivalence queries to achieve exact identification can be modified to achieve the PAC criterion (i.e.,* $\Pr[\text{error}(h) \geq \epsilon] \leq \delta$*) using calls to EX instead of equivalence queries. Furthermore, if the algorithm for learning with equivalence queries runs in polynomial time, then so will the PAC-learning algorithm.*

Proof: We need to simulate an equivalence query with calls to EX. For the i^{th} equivalence query, make $q_i = \left\lceil \frac{1}{\epsilon}(\ln \frac{1}{\delta} + i \ln 2) \right\rceil$ calls to EX. If h_i (the i^{th} hypothesis) is not consistent with a particular example, then that is the counterexample. Given that h_i is consistent with all q_i examples,

$$\Pr[\text{error}(h_i) > \epsilon] \leq (1-\epsilon)^{q_i}$$

Let h be the final hypothesis. So

$$\begin{aligned} \Pr[\text{error}(h) > \epsilon] &\leq \sum_{i=1}^{\infty} (1-\epsilon)^{q_i} \\ &\leq \sum_{i=1}^{\infty} e^{-\epsilon q_i} \\ &\leq \sum_{i=1}^{\infty} \frac{\delta}{2^i} \\ &= \delta \end{aligned}$$

where the second inequality holds because for all x, $1 - x \leq e^{-x}$. Finally, observe that the number of calls to EX made by the simulation is polynomial in the number of equivalence queries made by the original algorithm. ■

One can consider a variation of the PAC model in which the learner can either ask for a random example from EX or ask a membership query. We say a concept class is PAC-learnable with membership queries if the leaner can meet the PAC-criterion using a polynomial number of calls to EX and polynomial number of membership queries. As an immediate corollary to the above theorem we obtain.

Corollary 10.1 *A concept class that is learnable with equivalence and membership queries is PAC learnable with membership queries. Furthermore, if the algorithm for learning with equivalence and membership queries uses polynomial time, then so will the PAC-learning algorithm.*

The converse to the preceding theorem is false if efficiency must be preserved. Consider the class of singletons defined on the instance space of all binary strings of length n. We have already seen that the class of singletons requires $|\mathcal{C}| - 1 = 2^n - 1$ equivalence queries, but it only requires $\left\lceil \frac{1}{\epsilon}(n \ln 2 - \ln \delta) \right\rceil$ calls to EX in the PAC model.

10.4 Examples of Exactly Identifiable Classes

In this section we will exhibit examples, some with proofs, of classes that can be exactly identified by membership and equivalence queries.

10.4.1 k-CNF and k-DNF Formulas

As we saw in the last section, Valiant's algorithm for learning k-CNF can be used to efficiently implement the halving algorithm. Thus it follows that this class is learnable using a polynomial number of equivalence queries. There is a logically dual method for k-DNF formulas. Littlestone [29] describes an algorithm for learning these classes that may use significantly fewer queries. (This algorithm is described in the notes for Topic 12.) Finally, we note that by applying Lemma 10.1 to the class of 1-CNF formulas one gets that 2^{n-1} restricted equivalence and membership queries are needed in the worst case. A similar hardness result can be obtained for 1-DNF.

10.4.2 Monotone DNF Formulas

In this section we describe an efficient algorithm for learning any monotone DNF formula using membership and equivalence queries.

Theorem 10.3 *The class of monotone DNF formulas over n variables is exactly identifiable by equivalence and membership queries in time polynomial in n and the number of terms in the target concept.*

Before we prove this we give a definition and some observations.

Definition 10.1 *A prime implicant t of a propositional formula ϕ is a satisfiable product of literals such that t implies ϕ but no proper subterm of t implies ϕ.*

Example 10.1 *$\phi = ac \vee b\bar{c}$ has prime implicants ac, $b\bar{c}$, and ab.*

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 not greater than the number of terms in the formula.

Proof: [of Theorem 10.3] In the following let $\{y_1, y_2, \ldots, y_n\}$ be the variables.

Learn-Monotone-DNF
$\phi \leftarrow$ FALSE
Forever
    Make equivalence query with ϕ
    If "yes," output ϕ; halt.
    Else Let $x = b_1 b_2 \cdots b_n$ be the counterexample
        Let $t = \bigwedge_{b_i=1} y_i$
        For $i = 1, \ldots, n$
            If $b_i = 1$ perform membership query on x with i^{th} bit flipped
            If "yes," $t \leftarrow t \setminus \{y_i\}$ and $x = b_1 \cdots \bar{b_i} \cdots b_n$
        Let $\phi \leftarrow \phi \vee t$

We need to show that at each iteration, the term t is a new prime implicant of ϕ_*, the target concept. The proof will proceed by induction. For notation, let ϕ_i be the value of ϕ after the i^{th} iteration. Firstly, $\phi_0 = \text{FALSE}$ is trivially a prime implicant of ϕ_*. Assuming that ϕ_i is a prime implicant, the counterexample produced by the equivalence query is an instance x for which $\phi_*(x) = 1$ and $\phi_i(x) = 0$. So t is an implicant of ϕ_* but not of ϕ_i. Clearly the "trimming" procedure leaves a prime implicant.

Finally, the loop iterates exactly once for each prime implicant, and as stated above, this is bounded above by the number of terms in ϕ_*. Exactly n membership queries are performed during each iteration. So, assuming queries take polynomial time, the algorithm runs in time polynomial in both n and the number of terms in ϕ_*. ■

We now show that the counterexamples returned are essential to learn monotone DNF formulas efficiently.

Theorem 10.4 *For any $n > 0$ there is a class $\mathcal{D}$ of monotone DNF formulas with $2n$ variables and $n+1$ terms such that any algorithm that exactly identifies every formula in $\mathcal{D}$ using restricted equivalence queries and membership queries must make $2^n - 1$ queries in the worse case.*

Proof: Given $n > 0$, let $\phi_n = x_1y_1 + x_2y_2 + \cdots + x_ny_n$ and define $\mathcal{D}$ to be the 2^n formulas of the form $T + \phi_n$, where $T = \ell_1 \cdot \ell_2 \cdots \ell_n$ and $\ell_i = x_i$ or y_i.

Now, since for any formula ϕ in $\mathcal{D}$ there is exactly one assignment that satisfies ϕ but does not satisfy ϕ_n, any pair of distinct formulas $\phi_1, \phi_2 \in \mathcal{D}$ the assignment that satisfies both formulas are exactly those that satisfy ϕ_n. That is, $\phi_i \cap \phi_j = \phi_n \notin \mathcal{D}$. By Lemma 10.1 and $|\mathcal{D}| = 2^n$, at least $2^n - 1$ queries are needed. ■

We now consider the question: Is the class of monotone DNF formulas learnable with a polynomial number of equivalence queries? To prove that a concept class is not efficiently learnable by equivalence queries alone, Angluin [4] developed the "Method of Approximate Fingerprints". We now briefly describe this technique and apply it to the class of monotone DNF formulas.

The goal is to generalize the hardness result obtained for the class of singletons. Recall that for this class the adversary can reply to each equivalence query in such a way that only one concept is eliminated. This phenomenon is too much to ask, but also it is stronger than needed to prove a superpolynomial number of equivalence queries are needed.

Definition 10.2 *The instance w_n is an approximate fingerprint for hypothesis h in the concept class $\mathcal{C}$ if few (a superpolynomial fraction) concepts in $\mathcal{C}$ classify w_h the same as h.*

Approximate fingerprints can be used by the adversary to generate uninformative counterexamples to equivalence queries. Given any $h \in \mathcal{C}$, $h \neq L_*$, the adversary would respond "no", and return w_h, eliminating at most a superpolynomial fraction of the concept class. Thus the learner would require a superpolynomial number of equivalence queries in order to correctly eliminate all concepts but the target.

Angluin [4] has proven that monotone DNF formulas have approximate fingerprints, thus proving that exact identification cannot be achieved with a polynomial number of equivalence

queries. The key property used to prove this result is that for every monotone DNF formula ϕ there is an assignment with few 1's that satisfies ϕ or an assignment with few 0's that falsifies ϕ. Moreover, not many formulas share the value of ϕ on this assignment. This assignment serves as the approximate fingerprint. See Angluin's paper for the complete proof.

Combining these two hardness results, we get that the both equivalence and membership queries are required to learn DNF formulas. Thus the algorithm described in the beginning of the section, is the best one can expect.

10.4.3 Other Efficiently Learnable Classes

We now briefly mention a few of the many other concept classes that are learnable using membership and equivalence queries.

1. **k-Term DNF, k-Clause CNF.** Angluin [1] gives an algorithm that using equivalence and membership queries identifies any k-term DNF formula using time polynomial in n^k. A dual result holds for k-clause CNF formulas. Furthermore, note that a simple modification of the proof of Pitt and Valiant [33] that k-term DNF is not PAC-learnable by k-term DNF can be used to show that for $k > 1$, the class of k-term DNF or k-clause CNF formulas cannot be exactly identified by any algorithm that uses just equivalence queries and runs in time polynomial in n^k unless $P = NP$. Thus membership queries appear to be essential to Angluin's algorithm.

2. **Regular Sets.** Angluin [2] shows that if the minimum deterministic finite automaton that generates a given regular set has n states, then the DFA can be determined in time polynomial in n and in the length of the longest counterexample. This algorithm it covered in Topic 13.

3. **Read-once Formulas.** Angluin, Hellerstein, and Karpinski [6] give an efficient algorithm to exactly identify any read-once formula using membership and equivalence queries. (A read-once formula is one in which every literal appears at most once.) Furthermore, they show that monotone read-once formulas can be learned using only membership queries.

4. **Conjunction of Horn Clauses.** A *Horn clause* is a disjunction of literals at most one of which is negated. A *Horn sentence* is a conjunction of Horn clauses. Angluin, Frazier, and Pitt [5] show that Horn sentences are efficiently learnable using membership and equivalence queries. This algorithm is described in the notes for Topic 11. Note that the class of monotone DNF formulas is properly contained in the class of Horn sentences, thus it follows that neither equivalence queries nor membership queries alone suffice.

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 12: Learning With Abundant Irrelevant Attributes

Lecturer: Sally Goldman *Scribe: Marc Wallace*

12.1 Introduction

In these notes we first introduce the on-line (or mistake-bound) learning model. Then we consider when this model is applied to concept classes that contain a large number of irrelevant attributes. Most of this lecture comes from the paper, "Learning when Irrelevant Attributes Abound: A New Linear-threshold Algorithm," by Nick Littlestone [29].

12.2 On-line Learning Model

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

We now give an example from Goldman's thesis [18] to motivate the *on-line learning model*. This model is also known as the *mistake-bound learning model*. Suppose that when arriving at work (in Boston) you may either park in the street or park in a garage. In fact, between your office building and the garage there is a street on which you can always find a spot. On most days, street parking is preferable since you avoid paying the \$10 garage fee. Unfortunately, when parking on the street you risk being towed (\$50) due to street cleaning, snow emergency, special events, etc. When calling the city to find out when they tow, you are unable to get any reasonable guidance and decide the best thing to do is just learn from experience. There are many pieces of information that you might consider in making your prediction; e.g. the date, the day of the week, the weather. We make the following two assumptions: enough information is available to make good predictions if you know how to use it, and after you commit yourself to one choice or the other you learn of the right decision. In this example, the city has rules dictating when they tow; you just don't know them. If you park on the street at the end of the day you know if your car was towed; otherwise when walking to the garage you see if the street is clear (i.e. you learn if you would have been towed).

The on-line model is designed to study algorithms for learning to make accurate predictions in circumstances such as these. Formally, an on-line learning algorithm for C is an algorithm that runs under the following scenario. A *learning session* consists of a set of *trials*. In each trial, the learner is given an unlabeled instance $x \in X$. The learner uses its current hypothesis to predict if x is a positive or negative instance of the target concept $c \in C$ and then the learner is told the correct classification of x. If the prediction was incorrect, the learner has made a *mistake*. Note that in this model there is no training phase. Instead, the learner receives *unlabeled instances* throughout the entire learning session. However, after each prediction the learner "discovers" the correct classification. This feedback can then be used by the learner to improve its hypothesis. Observe that in this model it is beneficial for the learning algorithm to calculate hypothesis incrementally rather than starting from scratch each time.

In this learning model we shall evaluate the performance of a learning algorithm by the number of prediction mistakes made by the learning algorithm. We now describe the two most common ways in which this notion has been formalized.

> **Probabilistic Mistake Bound:** Let D be some arbitrary and unknown probability distribution on the instance space. The *probabilistic mistake bound* is the probability that the learner's hypothesis disagrees with c on the t^{th} randomly drawn instance from D. Formally, given any $n \geq 1$ and any $c \in C_n$, the learner's goal is to output a hypothesis h such that the probability that h makes a mistake on the $t + 1^{st}$ trial is at most $p(n)t^{-\beta}$ for some polynomial $p(n)$ and $0 < \beta$.
>
> **Absolute Mistake Bound:** The *absolute mistake bound* is the worst-case total number of mistakes made when the learner must make predictions for any, possibly infinite, sequence of instances. (Even if the instance space is finite, repetitions may occur.) Formally, given any $n \geq 1$ and any $c \in C_n$, the learner's goal is to make at most $p(n)$ mistakes for some polynomial $p(n)$.

Throughout the remainder of these notes we shall consider an on-line learning model where the absolute mistake bound criterion is used. In other words, 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 bound 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. See Haussler et al. [21] for a discussion of the relationship between the PAC model and on-line learning with the probabilistic mistake bound criterion.

After studying some general results about this on-line learning model we will focus on the situation in which the instances are drawn from the Boolean domain and proper classification for each instance can be determined by only a small fraction of the attribute space. Pattern recognition falls under this situation since a feature detector might extract a large number

of features for the learner's consideration not knowing which few will prove useful. Another example is building new concepts as Boolean functions of previously learned concepts that are stored in a library. For this problem, the learner may need to sift through a large library of available concepts to find the suitable ones to use in expressing each new concept. (The concept class of k-DNF fits into this model where the terms come from the library.)

12.3 Definitions and Notation

In this section we describe the notation used by Littlestone [29]. A concept class C consists of concepts c which are Boolean functions $c : \{0,1\}^n \rightarrow \{0,1\}$.

- For any algorithm A and target concept $c \in C$, let $M_A(c)$ be the maximum over all possible sequences of examples of the number of mistakes that algorithm A makes while learning the concept c.
- Define $M_A(C) = \max_{c \in C} M_A(c)$. If C is the empty set then by definition $M_A(C) = -1$.
- Define $opt(C) = \min_A M_A(C)$, the minimum over all possible algorithms of the worst-case number of mistakes.
- An algorithm A is *optimal* for C if and only if $M_A(C) = opt(C)$.

12.4 Halving Algorithm

One algorithm which often yields a good mistake bound is the halving algorithm (as described in Section 10.2.2). We now review the halving algorithm using the notation of Littlestone. We shall then look at a variant of it that can be shown to perform optimally.

Let CONSIST be a subset of the concepts in C that are consistent with all previous examples. So initially, CONSIST $= C$. Given a target concept class C and an instance x, we define $\xi_i(C, x) = \{c \in C | c(x) = i\}$ for $i = 0$ or $i = 1$. Thus the sets $\xi_0(C, x)$ and $\xi_1(C, x)$ are the set of concepts that are 0 at x, and the set of concepts that are 1 at x, respectively.

Upon receiving an instance x, the halving algorithm computes the sets $\xi_0(\text{CONSIST}, x)$ and $\xi_1(\text{CONSIST}, x)$. If $|\xi_1(\text{CONSIST}, x)| > |\xi_0(\text{CONSIST}, x)|$ then the algorithm predicts 1, otherwise it predicts 0. After receiving feedback the learner updates CONSIST: if $c(x) = 0$ then set CONSIST $= \xi_0(\text{CONSIST}, x)$, and if $c(x) = 1$ then set CONSIST $= \xi_1(\text{CONSIST}, x)$.

Theorem 12.1 *For any nonempty target class C, $M_{HALVING}(C) \leq \log_2 |C|$.*

Proof: Since the halving algorithm predicts according to the majority, its response is consistent with at least half of CONSIST. Therefore the size of CONSIST drops by a factor of at least two at each mistake. And since there is a consistent function, $|\text{CONSIST}| \geq 1$ always, so the algorithm can make no more than $\log_2 |C|$ mistakes. ■

The halving algorithm tells us that we can always choose a concept class such that $\log_2 |C|$ mistakes are actually made. Hence we have

Theorem 12.2 *For C such that $|C|$ is finite, $opt(C) \leq \log_2 |C|$.*

Before considering an algorithm that often makes fewer mistakes than the halving algorithm, we briefly consider the relation between this on-line learning model and the model of learning with equivalence queries. If both an equivalence query algorithm and a mistake-bound algorithm are applied to learning the same concept (over the same representation class), we have:

$$\begin{aligned} \#\text{ equiv. queries for exact id.} &\leq \#\text{ mistakes} + 1 \\ \#\text{ mistakes} &\leq \#\text{ equiv. queries for exact id.} \end{aligned}$$

These inequalities follow from the simple observation that each mistake made in the on-line model serves as a counterexample to an equivalence query. Furthermore, an equivalence query with the correct hypothesis corresponds to a hypothesis for which no additional mistakes will occur.

12.5 Standard Optimal Algorithm

Along with the problem that the halving algorithm often requires an exponentially amount of time and space, it is not always an optimal algorithm. After giving some more definitions, we shall describe a modification of the halving algorithm that always performs optimally (although is still not computationally feasible in most cases).

A *mistake tree* for C over X is defined to be a binary tree each of whose nodes is a non-empty subset of C and each of whose internal nodes is labeled with an $x \in X$ which satisfies:

1. The root node is C (along with a label).
2. Given any internal node C' labeled with x, the left child (if present) is $\xi_0(C', x)$ and the right child (if present) is $\xi_1(C', x)$.

A *complete k-mistake tree* is a mistake tree that is a complete binary tree of height k. (The height of a binary tree is the number of edges in the longest path from the root to a leaf.) Finally, we define $K(C)$ as the largest integer k such that there exists a complete k-mistake tree for the concept class C. We shall use the convention that $K(\emptyset) = -1$.

In Figure 12.1 is an example of a complete 2-mistake tree. The concept class used is a simple one; $X = \{0,1\}^5$ and C consists of the five concepts, $f_i(x_1, \ldots, x_5) = x_i$ for $i = 1, \ldots, 5$.

As we shall show, these trees characterize the number of mistakes made by the optimal learning algorithm.

We now define the *standard optimal algorithm* (SOA):

> Let CONSIST contain all $c \in C$ consistent with all past instances.
> Predict 1 if $K(\xi_1(\text{CONSIST}, x)) > K(\xi_0(\text{CONSIST}, x))$.
> Predict 0 otherwise.

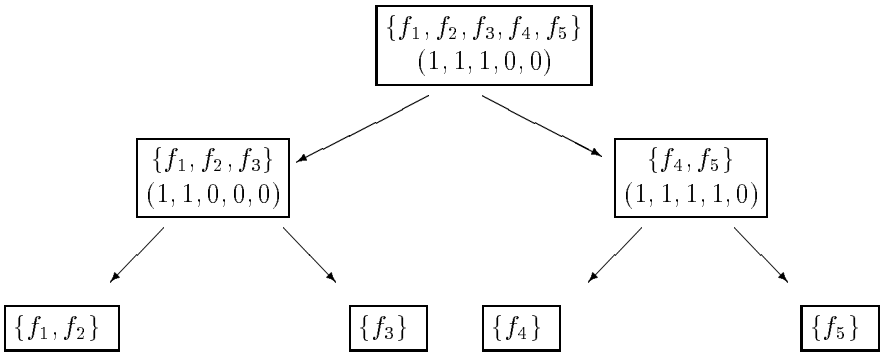

Figure 12.1: An example of a complete 2-mistake tree for a concept class C over instance space $X = \{0,1\}^5$ where C consists of the five concepts $f_i(x_1, \ldots, x_5) = x_i$ for $i = 1, \ldots, 5$.

So if a mistake is made the remaining consistent functions have the smaller maximal complete mistake tree. As we shall show this yields an optimal algorithm. However, we first prove the following two lemmas.

Lemma 12.1 *$opt(C) \geq K(C)$.*

Proof: If $C = \emptyset$ then by definition $K(C) = -1$ and the lemma trivially follows. So assume that $C \neq \emptyset$ and $k = K(C)$. Given any algorithm A we show how the adversary can choose a target concept and a sequence of instances such that A makes at least k mistakes. If $k = 0$ the lemma is trivially satisfied. Otherwise, the adversary chooses the instance that is the root of a complete k-mistake tree for C. Regardless of A's prediction the adversary replies that the prediction is incorrect. The remaining consistent concepts form a complete $(k-1)$-mistake tree, so by induction we are done. ■

Lemma 12.2 *Suppose that we run SOA in order to learn a concept in C where $x_1, \ldots, x_t$ is the sequences of instances given to SOA. Let* CONSIST$_i$ *be the value of the variable* CONSIST *at the beginning of the ith trial. Then for any $k \geq 0$ and $i \in \{1, \ldots, t\}$, if $K(\text{CONSIST}_i) = k$, then SOA will make at most k mistakes during the trials $i, \ldots, t$.*

Proof: This will be a proof by induction on k. For the base case observe that by construction, the target function is always in CONSIST$_i$. If CONSIST$_i$ has two elements, we can always use an instance on which they differ as the root node of a 1-mistake tree, so $K(\text{CONSIST}_i) = 0$ implies that CONSIST$_i$ has only the target function. Since $K(0) = -1$ (by definition), SOA

cannot make any mistakes when only the target function is left. Hence the base case $k = 0$ is proven.

We now prove the lemma for arbitrary $k > 0$, assuming it holds for $k-1$. If SOA makes no mistakes during trials $i, \ldots, t-1$ then the lemma is trivially true. So let j be the first trial among $i, \ldots, t-1$ in which a mistake is made. We now prove by contradiction that $\xi_0(\text{CONSIST}_j, x_j)$ and $\xi_1(\text{CONSIST}_j, x_j)$ cannot both be complete k-mistake trees. Suppose there are k-mistake trees for both $\xi_0(\text{CONSIST}_j, x_j)$ and $\xi_1(\text{CONSIST}_j, x_j)$. Then we can combine these into $(k+1)$-mistake tree by using x_j as a root node. But by hypothesis, such a tree cannot exist (k is the largest size of a mistake tree). Therefore one of $K(\xi_0(\text{CONSIST}_j, x_j))$ or $K(\xi_1(\text{CONSIST}_j, x_j))$ is less than k. Since SOA always picks the larger of the two, and it made a mistake, then $K(\text{CONSIST}_{j+1})$ will be less than k. Using induction only $k-1$ mistakes will be made from this point on. So only k mistakes can be made completing the inductive step. ■

Now we are ready to prove the main result of this section.

Theorem 12.3 $opt(C) = M_{SOA}(C) = K(C)$.

Proof: By setting $i = 1$ and $k = K(C)$ in the Lemma 12.2, we get $M_{SOA}(C) \leq K(C)$. Furthermore, by Lemma 12.1, $K(C) \leq opt(C)$. Combining these two inequalities it follows that $M_{SOA}(C) \leq opt(C)$. Finally, by definition, $opt(C) \leq M_{SOA}(C)$. ■

Let us now consider some lower bounds on $opt(C)$. The Vapnik-Chervonenkis dimension is useful in this respect.

Theorem 12.4 $opt(C) \geq VCD(C)$.

Proof: Let $\{v_1, \ldots, v_k\} \subseteq X$ be any set shattered by C. Then clearly a complete k-mistake tree can be constructed for C, where all internal nodes at a depth i are labeled with v_{i+1}. Apply this procedure to $k = VCD(C)$. ■

We note however that this is not a tight lower bound. Let C be the a concept class of the instance space $X = \{1, \ldots, 2^n - 1\}$ where $C = \{c_i : X \rightarrow \{0,1\} | c_i(x_j) = 1$ if and only if j < i$\}$. Thus, these are concepts of "all things less than i". Clearly $VCD(C) = 1$ since for two concepts c_i and c_j, the greater of $\{i, j\}$ cannot be covered by one concept without covering the other. Yet, we can show that $opt(C) \geq n$ by constructing a complete n-mistake tree as follows: label the root node 2^{n-1}. Each branch of the tree is the same type but half as big, so by inductively constructing the binary tree, we have a complete n-mistake tree.

12.6 The Linear Threshold Algorithm: WINNOW1

In this section we describe an algorithm that efficiently deals with a large number of irrelevant attributes when learning in a Boolean domain. If desired this algorithm can be implemented within a neural network framework as a simple linear threshold function. This algorithm is similar to the classical perceptron algorithm, except that it uses a multiplicative weight-update scheme that permits it to perform much better than classical perceptron training

prediction	correct	name	update scheme
1	0	elimination	if $x_i = 1$ then set $w_i = 0$.
0	1	promotion	if $x_i = 1$ then set $w_i = \alpha \cdot w_i$.

Figure 12.2: The update scheme used by WINNOW1.

algorithms when many attributes are irrelevant. From empirical evidence it appears that for the perceptron algorithm the number of mistakes grows linearly with the number of irrelevant attributes. Here we shall describe an algorithm for which the number of mistakes only grows logarithmically with the number of irrelevant attributes.

A *linearly separable Boolean function* 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 point 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 $f(x_1,\ldots,x_n) = x_{i_1} \vee \ldots \vee x_{i_k}$, then the hyperplane $x_{i_1} + \ldots + x_{i_k} = 1/2$ is a separating hyperplane.

We present a limited form, WINNOW1, and will later generalize it. WINNOW1 is a linear threshold algorithm over the Boolean space $X = \{0,1\}^n$ that is designed for learning monotone disjunctions. There are n real valued weights $w_1,\ldots,w_n$ maintained by the algorithm. Initially each weight is set to 1. Also, a real number θ, called the *threshold*, is utilized. When WINNOW1 receives an instance $x = (x_1,\ldots,x_n)$, it predicts as follows:

- if $\sum_{i=1}^n w_i x_i > \theta$ then it predicts 1.
- if $\sum_{i=1}^n w_i x_i \leq \theta$ then it predicts 0.

When a mistake is made, the weights with non-zero x_i are updated as shown in Figure 12.2. Note that the threshold θ is never altered. Good values for the parameters θ and α are $\theta = n/2$ and $\alpha = 2$.

We now present three lemmas which will be used to prove a later theorem. All three have as preconditions that WINNOW1 is run with $\alpha > 1$ and $\theta \geq 1/\alpha$ for the learning of k-literal monotone disjunctions.

Lemma 12.3 *Let u be the number of promotion steps that have occurred in some sequence of trials, and v the number of elimination steps in the same trials. Then $v \leq (n/\theta)+(\alpha-1)u$.*

Proof: Consider the sum $\sum_{i=1}^n w_i$. Initially this sum is n since all of the weights are initially 1. At each promotion, the sum can increase by no more than $(\alpha-1)\theta$, since the sum (over all x_i that are on) must be less than θ for a promotion to occur. Similarly, at each elimination step, the sum must be decreased by at least θ. Hence

$$0 \leq \sum_{i=1}^{n} w_i \leq n + \theta(\alpha - 1)u - \theta v.$$

Thus

$$\theta v \leq n + \theta(\alpha - 1)u$$

giving the desired result. ■

Lemma 12.4 *For all i, $w_i \leq \alpha\theta$ (after any number of trials).*

Proof: Since $\theta \geq 1/\alpha$, for all i, initially $w_i \leq \alpha\theta$. We now proceed by induction on the number of steps. Note that w_i is only increased by a promotion step when $x_i = 1$ and $\sum_{i=1}^{n} w_i x_i \leq \theta$. These conditions can only occur together if $w_i \leq \theta$ prior to promotion. Thus $w_i \leq \alpha\theta$ after the promotion step. So after any step the claim is true, hence it is always true. ■

Lemma 12.5 *After u promotion steps and any number of eliminations, there exists an i such that $\log_\alpha w_i \geq u/k$.*

Proof: Let the k-literal disjunction we are learning be of the form

$$f(x_1, \ldots, x_n) = x_{i_1} \vee \ldots \vee x_{i_k}$$

Let the set $R = \{i_1, \ldots, i_k\}$, and consider the product $P = \prod_{j \in R} w_j$. Since $f = 0$ if and only if $x_j = 0$ for all $j \in R$, and elimination occurs when $f = 0$, elimination cannot affect the product P at all. Also, at each promotion step, P is increased by at least a factor of α, since at least one of the x_is was on for $i \in R$. At first we have $P = 1$. After u promotions (and any number of eliminations), $P \geq \alpha^u$. Taking logs of both sides, we have $\sum_{i \in R} \log_\alpha w_i \geq u$. Since $|R| = k$, for some $i \in R$ we must have that $\log_\alpha w_i \geq u/k$. ■

We are finally ready to prove an upperbound on the number of mistakes made by WINNOW1 when learning a k-literal monotone disjunction.

Theorem 12.5 *For the learning of k-literal monotone disjunctions, if WINNOW1 is run with $\alpha > 1$ and $\theta \geq 1/\alpha$, then the total number of mistakes is at most $\alpha k(\log_\alpha \theta + 1) + n/\theta$.*

Proof: The total number of mistakes is clearly the number of promotion steps plus the number of elimination steps $(u + v)$. Lemmas 12.4 and 12.4 yield the following bound on v:

$$u/k \leq \log_\alpha w_i \leq \log_\alpha \alpha\theta = \log_\alpha \theta + 1$$

So

$$u \leq k(\log_\alpha \theta + 1)$$

Plugging in this value of u into the inequality of Lemma 12.3, we obtain the following bound on v:

$$v \leq (n/\theta) + (\alpha - 1)k(\log_\alpha \theta + 1)$$

Adding the two bounds together gives the desired result. ■

Observe that WINNOW1 need not have prior knowledge of k although the number of mistakes will depend on k. Littlestone's paper also discusses how to optimally choose θ and α if an upperbound on k is known a priori.

Noticing that k-literal monotone disjunctions are 1-term monotone k-DNF formulas, we state (without proof) the following lower bound on the VC dimension and thus on the number of mistakes made in the on-line model. See Littlestone's paper for the proof.

Lemma 12.6 *For $1 \leq k \leq n$ and $1 \leq l \leq \binom{n}{k}$, let C be the class of l-term monotone k-DNF formulas. Let m be any integer with $k \leq m \leq n$ with $\binom{m}{k} \geq l$. Then $VCD(C) \geq kl\lfloor \log_2(n/m) \rfloor$.* ■

We now apply this theorem to two special cases. If $l = 1$ and $m = k$ then we get that $VCD(C) \geq k\lfloor \log_2 n/k \rfloor$ where C contains all conjunctions of at most k variables chosen from n variables. Likewise, if $k = 1$ and $m = l$ then $VCD(C) \geq l\lfloor \log_2 nl \rfloor$ where C contains all disjunctions of at most l variables chosen from n variables.

12.7 Extensions: WINNOW2

Let $X = \{0,1\}^n$ be the instance space, and $0 < \delta \leq 1$. We define $F(X, \delta)$ to be the set of all functions $X \to \{0,1\}$ with the following property: for each $f \in F(X, \delta)$ there exist $\mu_1, \ldots, \mu_n \geq 0$ such that for all instances $x = (x_1, \ldots, x_n) \in X$,

$$\sum_{i=1}^{n} \mu_i x_i \geq 1 \text{iff(x)} = 1$$

and

$$\sum_{i=1}^{n} \mu_i x_i < 1 - \delta \text{iff(x)} = 0$$

That is, the inverse images of 0 and 1 are separated by at least δ.

An example of such a class of functions would be the r-of-k threshold functions. These functions are defined by selecting k variables which are "important". Then, $f(x) = 1$ whenever r or more of these k significant variables are on. Let the selected variables be $x_{i_1}, \ldots, x_{i_k}$.

$$f(x) = 1 \iff x_{i_1} + \ldots + x_{i_k} \geq r$$

Thus,

$$f(x) = 1 \iff (1/r)x_{i_1} + \ldots + (1/r)x_{i_k} \geq 1$$

Also,

$$f(x) = 0 \iff x_{i_1} + \ldots + x_{i_k} \leq r - 1$$

Thus,

$$f(x) = 0 \iff (1/r)x_{i_1} + \ldots + (1/r)x_{i_k} \leq 1 - (1/r)$$

Hence r-of-k threshold functions are contained in $F(\{0,1\}^n, 1/r)$.

We describe the algorithm WINNOW2 here. It is basically the same as WINNOW1, but updates made in response to mistakes is slightly different: instead of eliminating weights entirely, they are divided by α. One could call this altered step a demotion. See Figure 12.3 for the update scheme used by WINNOW2.

A theorem similar to the one for WINNOW1 exists to give an upper bound on the number of mistakes that WINNOW2 will make. We state without proof the theorem here. For a proof see Littlestone's paper.

prediction	correct	name	update scheme
1	0	demotion	if $x_i = 1$ then set $w_i = w_i/\alpha$.
0	1	promotion	if $x_i = 1$ then set $w_i = w_i * \alpha$.

Figure 12.3: The update scheme used by WINNOW2.

Theorem 12.6 *Let $0 < \delta \leq 1$, and the target function be in $F(X, \delta)$ for $X = \{0,1\}^n$. If the appropriate $\mu_1, \ldots, \mu_n$ are chosen so that the target function satisfies the definition for $F(X, \delta)$, and WINNOW2 is run with values $\alpha = 1 + \delta/2$ and $\theta \geq 1$ and the algorithm gets its instances from X, then the number of mistakes is bounded above by*

$$\frac{8}{\delta^2}\frac{n}{\theta} + \left(\frac{5}{\delta} + \frac{14 \ln \theta}{\delta^2}\right) \sum_{i=1}^{n} \mu_i$$

For r-of-k threshold functions we have $\delta = 1/r$ and $\sum \mu_i = k/r$. So setting $\theta = n$ yields a mistake bound of $8r^2 + 5k + 14kr \ln n$.

12.8 Transformations to Other Concept Classes

Suppose we had a learnable concept class, and wanted to learn a second class. If morphisms existed which map instances between the two concept spaces, and mapped from the answers of one to the answers of the second, then it would not seem to be difficult to learn the second class by simply pretending to be learning the first. This is the basic idea behind these transformations.

Let the instance space of the derived algorithm be X_1, and that of the original algorithm be X_2. The transformations will take the form of $T_i : X_1 \to X_2$ and $T_p : \{0,1\} \to \{0,1\}$. The mapping T_i maps between the instance spaces, and T_p, the map between predictions, is either the identity function or the function which interchanges 0 and 1. Let A_1 be the algorithm desired to learn concepts over X_1, and let A_2 be the algorithm provided that learns concepts over X_2. When A_1 gets an instance $x \in X_1$, it sends the instance $T_i(x)$ to A_2. Then A_2 generates a prediction y. Next, A_2 sends this prediction to A_1 which outputs $T_p(y)$. Finally, any reinforcements are passed directly to A_2.

So suppose we have an algorithm A_2, and we want to derive an algorithm to learn C_1. We need not only a C_2 that A_2 can learn, but also two mappings T_i, T_p such that for all $g \in C_1$, there exists $f \in C_2$ such that $T_p \circ f \circ T_i = g$.

Theorem 12.7 *Suppose we are given maps $T_i : X_1 \to X_2$ and $T_p : \{0,1\} \to \{0,1\}$, an original algorithm A that takes instances from X_2, and a derived algorithm B (defined as above). Suppose also that we have a function $g : X_1 \to \{0,1\}$ that we want B to learn. If $f : X_2 \to \{0,1\}$ is a function that A can learn with some bounded number of mistakes such that $T_p \circ f \circ T_i = g$, then B will learn g with at most $M_A(f)$ mistakes.*

Proof: This proof is trivial. Basically, B can only make a mistake when A has. ∎

We now go through several example transformations.

Example 12.1 *Arbitrary Disjunctions*

We first consider learning arbitrary disjunctions, using WINNOW1 as A_1. We can determine the correct signs (negated variable or not) for all variables by finding a single negative example. Predict positive until we get a negative example (one mistake at most). Let $(z_1, \ldots, z_n)$ be the negative example. All future instances are sent to WINNOW1 using

$$T_i(x_1, \ldots, x_n) = (x_1 \oplus z_1, \ldots, x_n \oplus z_n).$$

Such a mapping makes sense, since if z_i is off in the negative example, it cannot be a negated variable, and vice versa, so only the negated variables are negated by the mapping. (What is done to irrelevant variables does not matter.) Finally, let T_p be the identity function. It is easily shown that the conditions of Theorem 12.7 are satisfied. So the mistake bound for learning non-monotone disjunctions with this technique is just one more than the corresponding mistake bound for learning monotone disjunctions.

Example 12.2 *Arbitrary Conjunctions*

The example of arbitrary disjunctions can be extended to learning k-literal monotone conjunctions. Let A_2 be the algorithm described in the above example, let $T_i = (1 - x_1, \ldots, 1 - x_n)$, and let $T_p(r) = 1 - r$. Again, using DeMorgan's law it is easily verified that the conditions of Theorem 12.7 hold. Thus, the number of mistakes will be bounded by $2k \log_2 n + 2$.

Example 12.3 *k-DNF for fixed k*

As another example, consider learning k-DNF for a fixed k. We notice that this class is more difficult, since it is not a linearly separable class. Let the original algorithm be WINNOW1 over k-literal disjunctions. Let n_1 be the number of variables in the derived concept class, and let n_2, the number of variables over which the original algorithm is run, be set to $n_2 = \sum_{i=0}^{k} 2^i \binom{n}{i}$. Then we have that

$$T_i(x) = (c_1(x), \ldots, c_{n_2}(x))$$

where $x = (x_1, \ldots, x_{n_1})$ and the c_i's range over all possible conjunctions which could be terms in a k-DNF formula. Notice that given any k-DNF we can represent it as such. If such a g is given we can construct an f in the derived space which is a disjunction of all the variables whose corresponding terms are in the expansion of g.

By expanding the summation for n_2, we can see that $n_2 \leq (2n)^k + 1$. Since the original algorithm will be learning an l-literal monotone disjunction, we will make $O(l \log n^k) =$

$O(lk \log n)$ mistakes. Compare this to the algorithm Valiant presented which can be forced to make $\binom{n}{k} - l$ mistakes in the worst case.

By using the VC dimension, we can find a lower bound on the mistake bound. Taking $m = \lceil kl^{1/k} \rceil$, we have

$$\binom{m}{k} \geq \frac{m^k}{k^k} \geq 1$$

Thus a lower bound would be given, when $kl^{1/k} \leq n$, by:

$$kl \left\lfloor \log_2 \frac{n}{\lceil kl^{1/k} \rceil} \right\rfloor$$

If, however, l is known, an even better bound of $4l + 2kl \log_2 \left(\frac{2n}{l^{1/k}} \right)$ can be achieved.

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 13: Learning Regular Sets

Lecturer: Sally Goldman *Scribe: Ellen Witte*

13.1 Introduction

In this lecture we consider the problem of learning an unknown regular set using membership and equivalence queries. The material presented in this lecture comes from the paper, "Learning Regular Sets from Queries and Counterexamples," by Dana Angluin [2]. Since deterministic finite-state acceptors (dfa's) recognize the same languages as regular sets, this is equivalent to the problem of learning an unknown dfa. We will express our hypothesis in the form of a dfa by giving the initial state, final (accepting) states and transition function. Our goal will be to find a dfa with the *minimum* number of states that recognizes the unknown regular set.

Gold [15] has shown that finding a minimum dfa consistent with an arbitrary set of positive and negative examples is NP-hard. The learning algorithm has the advantage of being able to select the examples for the membership queries, thus the set of examples used to help construct the dfa is not arbitrary. To be certain we are not asking for too much with membership and equivalence queries, we would like both types of queries to be computable in time polynomial in the number of states in a minimum dfa recognizing the regular set and polynomial in the length of the hypothesis dfa. The teacher has access to the minimum dfa, thus a membership query can be answered by simply tracing the dfa using the given string. In addition, there is a polynomial time algorithm for testing the equivalence of two dfa's which returns a counterexample if the dfa's are not equivalent.

13.2 The Learning Algorithm

In the following discussion we use the notation L^* to denote the learning algorithm, U to denote the unknown regular set to be learned, and A to denote the alphabet of the regular language. (The alphabet is known to the learner.)

The learning algorithm develops a hypothesis by using membership queries to determine whether or not certain strings are in the set U. The results of these queries are stored in an *observation table* maintained by the learner. Periodically the learner will construct a hypothesis dfa from the observation table and perform an equivalence query to see if the hypothesis is correct. If the hypothesis is not correct then the counterexample will be used to modify the observation table. Next we describe the structure of the observation table. Later we will see how the hypothesis dfa is constructed from the table.

T	λ
λ	1
a	0
b	0

Table 13.1: Initial observation table.

The observation table represents a mapping T from a set of finite strings to {0,1}. The function T is such that $T(u) = 1$ if and only if $u \in U$. The strings in the table are formed by concatenating an element from the set $S \cup S \cdot A$ with an element from the set E, where S is a nonempty finite prefix-closed set of strings and E is a nonempty finite suffix-closed set of strings. $S \cdot A = \{s \cdot a : \ s \in S, a \in A\}$, where $\cdot$ is the concatenation operator. The table can be represented by a two-dimensional array with rows labeled by elements of $S \cup S \cdot A$ and columns labeled by elements of E. The entry in row $s \in S \cup S \cdot A$ and column $e \in E$ contains $T(s \cdot e)$. The observation table will be denoted (S, E, T).

In the initial observation table, $S = E = \{\lambda\}$, where λ represents the empty string. This table has one column and $1 + |A|$ rows. For example, if $A = \{a, b\}$ and the regular language $U = \{u : \ u$ contains an even number of both a's and b's$\}$ then Table 13.1 is the initial table. The double horizontal line separates the rows labeled with elements of S from the rows labeled with elements of $S \cdot A$. We will denote the row of the table labeled by $s \in S \cup S \cdot A$ by row(s). In the example, row$(a) = (0)$.

We define two properties of an observation table. An observation table is *closed* if for all $t \in S \cdot A$, there exists an $s \in S$ such that row$(t) =$ row(s). An observation table is *consistent* if whenever $s_1, s_2 \in S$ satisfy row$(s_1) =$ row(s_2), then for all $a \in A$, row$(s_1 \cdot a) =$ row$(s_2 \cdot a)$. Table 13.1 is not closed since $a \in S \cdot A$ and row$(a) = (0)$, but there is no $s \in S$ such that row$(s) = (0)$. The table is consistent.

We now define the dfa, denoted $M(S, E, T)$, corresponding to the closed, consistent observation table (S, E, T). (The observation table must be closed and consistent otherwise the dfa is undefined.) $M(S, E, T)$ is the acceptor over alphabet A, with state set Q, initial state q_0, accepting state set F and transition function δ where:

$$
\begin{aligned}
Q &= \{\text{row}(s) : \ s \in S\} \\
q_0 &= \text{row}(\lambda) \\
F &= \{\text{row}(s) : \ s \in S, T(s) = 1\} \\
\delta(\text{row}(s), a) &= \text{row}(s \cdot a)
\end{aligned}
$$

We can show that $M(S, E, T)$ is a well defined acceptor. First, it is always true that $\lambda \in S$ since S is nonempty and prefix-closed. Thus $q_0 = \text{row}(\lambda)$ is well defined. Second, $F = \{\text{row}(s) : s \in S, T(s) = 1\}$ is well defined. For F to be ill defined there must exist $s_1, s_2 \in S$ such that row$(s_1) =$ row(s_2) but $T(s_1) \neq T(s_2)$. Since E is nonempty and suffix-closed, $\lambda \in E$. Thus row(s_1) contains $T(s_1 \cdot \lambda) = T(s_1)$, and row$(s_2)$ contains $T(s_2 \cdot \lambda) = T(s_2)$. Since row$(s_1) =$ row(s_2), it must be true that $T(s_1) = T(s_2)$. Thus, F is well defined. Third, $\delta(\text{row}(s), a) = \text{row}(s \cdot a)$ is well defined. There are two ways for δ to be ill defined. First,

there could exist $s_1, s_2 \in S$ such that $\text{row}(s_1) = \text{row}(s_2)$ but $\text{row}(s_1 \cdot a) \neq \text{row}(s_2 \cdot a)$. Second, it could be that $\text{row}(s \cdot a)$ is not in Q. Because the table is consistent, the first condition cannot occur. Because the table is closed, the second condition cannot occur. Thus δ is well defined.

We now give the learning algorithm L^*, which maintains an observation table (S, E, T). The table is modified to reflect responses to membership and equivalence queries. The hypothesis dfa posed by each equivalence query is the dfa $M(S, E, T)$ corresponding to the current observation table.

Algorithm L^*

Initialize S and E to $\{\lambda\}$.
Ask membership queries for λ and a, $\forall a \in A$.
Construct the initial observation table (S, E, T).
Repeat:
    While (S, E, T) is not closed or not consistent:
        If (S,E,T) is not consistent,
            find $s_1, s_2 \in S, a \in A, e \in E$ such that $\text{row}(s_1) = \text{row}(s_2)$
                and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$,
            add $a \cdot e$ to E,
            extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.
        If (S,E,T) is not closed,
            find $s_1 \in S, a \in A$ such that $\text{row}(s_1 \cdot a) \neq \text{row}(s) \forall s \in S$,
            add $s_1 \cdot a$ to S,
            extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.
    Perform an equivalence query with $M = M(S, E, T)$.
    If answer is "no" with counterexample t,
        add t and its prefixes to S,
        extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.
Until answer is "yes" from equivalence query.

We note that the runtime of the algorithm depends upon the length of the longest counterexample, since the table must include each counterexample t and all its prefixes. The algorithm will never remove a row or column from the table. The only adjustments to the table are the addition of rows and columns. Also, the algorithm must test the closure and consistency of the current observation table. The closure is easy to check; the consistency is a bit more time consuming, but not too difficult. Before analyzing the correctness of the algorithm we consider an example execution of L^*.

13.3 An Example

T_1	λ
λ	1
a	0
b	0

Table 13.2: Initial observation table.

T_2	λ
λ	1
a	0
b	0
aa	1
ab	0

Table 13.3: Second observation table.

In this section we trace the execution of L^* as it learns the regular set U defined in the previous section. That is, $U = \{u : u \text{ contains an even number of both } a\text{'s and } b\text{'s}\}$. The alphabet $A = \{a, b\}$. The initial table was given in the previous section and is repeated as Table 13.2. To distinguish the versions of the observation table as the algorithm progresses, we label the i^{th} table in the upper left corner with T_i.

We noted earlier that this table is consistent, but not closed since $a \in S \cup A$ but $\text{row}(a) = 0 \neq \text{row}(s)$ for any $s \in S$. Accordingly, L^* moves a to S, extends $S \cup A$ to include extensions of a and fills in the new entries in the first column of the table. Table 13.3 is the second observation table.

This table is closed and consistent, so the algorithm poses an equivalence query with the dfa $M(S, E, T)$. This dfa has two states, $q_0 = \text{row}(\lambda)$ and $q_1 = \text{row}(a)$. The starting state is q_0, the accepting state is q_0 and the transition function is as shown in Table 13.4. To distinguish the various hypothesis dfa's, we label the transition function corresponding to observation table i with δ_i.

This dfa does not recognize U thus the equivalence query returns a counterexample t. Let us assume the counterexample is $t = bb$. It is easy to see that $t \in U$, but t is not accepted by the hypothesis dfa. Accordingly, L^* adds bb and all its prefixes to S and adds the extensions of bb to $S \cdot A$. L^* then fills in the new entries in the first column. Table 13.5 is the resulting observation table.

The third observation table is closed, but not consistent, since $\text{row}(a) = \text{row}(b)$ but

δ_2	a	b
q_0	q_1	q_1
q_1	q_0	q_1

Table 13.4: Dfa for second observation table.

T_3	λ
λ	1
a	0
b	0
bb	1
aa	1
ab	0
ba	0
bba	0
bbb	0

Table 13.5: Third observation table.

T_4	λ	a
λ	1	0
a	0	1
b	0	0
bb	1	0
aa	1	0
ab	0	0
ba	0	0
bba	0	1
bbb	0	0

Table 13.6: Fourth observation table.

δ_4	a	b
q_0	q_1	q_2
q_1	q_0	q_2
q_2	q_2	q_0

Table 13.7: Dfa for fourth observation table.

T_5	λ	a
λ	1	0
a	0	1
b	0	0
bb	1	0
ab	0	0
abb	0	1
aa	1	0
ba	0	0
bba	0	1
bbb	0	0
aba	0	0
$abba$	1	0
$abbb$	0	0

Table 13.8: Fifth observation table.

row$(aa) \neq$ row(ba). The algorithm adds a to E and fills in the values of T for the new column. Table 13.6 is the result.

Table 13.6 is a closed, consistent observation table. The algorithm constructs the dfa $M(S, E, T)$, which has three states, $q_0 =$ row(λ), $q_1 =$ row(a) and $q_2 =$ row(b). The starting state is q_0, the accepting state is q_0 and the transition function is as shown in Table 13.7.

This dfa is not quite correct, thus the equivalence query returns a counterexample t. Let us assume the counterexample is $t = abb$, which is not in U but is accepted by the dfa. The algorithm adds abb and all its prefixes and extensions to the observation table and fills in the values of T. Table 13.8 shows the result.

Table 13.8 is closed, but not consistent since row$(b) =$ row(ab) but row$(bb) \neq$ row(abb). The algorithm adds b to E and fills in the values of T for this new column. The result is Table 13.9.

This table is closed and consistent, so the algorithm constructs $M(S, E, T)$. This dfa has four states, $q_0 =$ row(λ), $q_1 =$ row(a), $q_2 =$ row(b) and $q_3 =$ row(ab). The starting state is q_0, the accepting state is q_0 and the transition function is as shown in Table 13.10.

This dfa accepts exactly the language U, thus the equivalence query returns "yes" and the algorithm halts. This is the minimum size dfa to recognize the language consisting of all strings with an even number of a's and an even number of b's.

T_6	λ	a	b
λ	1	0	0
a	0	1	0
b	0	0	1
bb	1	0	0
ab	0	0	0
abb	0	1	0
aa	1	0	0
ba	0	0	0
bba	0	1	0
bbb	0	0	1
aba	0	0	1
$abba$	1	0	0
$abbb$	0	0	0

Table 13.9: Sixth observation table.

δ_6	a	b
q_0	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

Table 13.10: Dfa for sixth observation table.

13.4 Algorithm Analysis

There are three requirements of the algorithm which we address in this section. First, we show that the algorithm is correct, that is, L^* finds a *minimum* dfa to recognize U. In addressing this issue we assume the algorithm terminates and show that, assuming termination, the algorithm finds a minimum dfa to recognize U. This leads to the second issue, showing that the algorithm terminates. Third, we show that the time complexity of the algorithm is polynomial in the number of states in the minimum dfa and polynomial in the length of the longest counterexample. We consider these requirements in the next three subsections.

13.4.1 Correctness of L^*

Assuming L^* terminates, the claim that L^* produces a dfa that recognizes U is trivial. The condition for termination is that the equivalence query returns "yes", indicating that the dfa recognizes U. The claim that L^* produces a *minimum* dfa to recognize U is more complicated. The key to this claim is the following theorem about the acceptor $M(S,E,T)$ constructed from a closed, consistent observation table (S,E,T).

Theorem 13.1 *If (S,E,T) is a closed, consistent observation table, then the acceptor M(S,E,T) is consistent with the finite function T. Any other acceptor consistent with T but inequivalent to M(S,E,T) must have more states.*

Proof: This theorem is proven by several lemmas.

Lemma 13.1 *Assume that (S,E,T) is a closed, consistent observation table. For the acceptor M(S,E,T) and every* $s \in (S \cup S \cdot A)$, $\delta(q_0, s) = \text{row}(s)$.

Proof: The proof is by induction on the length of s.

The base case is $|s| = 0$ implying that $s = \lambda$. By definition, $q_0 = \text{row}(\lambda)$, thus

$$\delta(q_0, s) = \delta(\text{row}(\lambda), \lambda) = \text{row}(\lambda) = \text{row}(s).$$

For the induction step we assume the lemma holds for $|s| \leq k$ and show it holds for strings of length $k+1$. Let $t \in (S \cup S \cdot A)$ with $|t| = k+1$. Clearly, $t = s \cdot a$ for some string s of length k and some $a \in A$. By the construction of the observation table, $s \in S$. Thus,

$$\begin{aligned}
\delta(q_0, t) &= \delta(q_0, s \cdot a), \\
&= \delta(\delta(q_0, s), a), \\
&= \delta(\text{row}(s), a), && \text{by induction hypothesis,} \\
&= \text{row}(s \cdot a), && \text{by definition of } \delta, \\
&= \text{row}(t).
\end{aligned}$$

■

Lemma 13.2 *Assume that* (S,E,T) *is a closed, consistent observation table. The acceptor* $M(S,E,T)$ *is consistent with the finite function* T. *That is, for every* $s \in (S \cup S \cdot A)$ *and* $e \in E$, $\delta(q_0, s \cdot e) \in F$ *if and only if* $T(s \cdot e) = 1$.

Proof: The proof is by induction on the length of e.

In the base case, $|e| = 0$, thus $e = \lambda$. By the preceding lemma, $\delta(q_0, s \cdot e) = \text{row}(s)$. If $s \in S$ then by the definition of F, $\text{row}(s) \in F$ if and only if $T(s) = 1$. If $s \in S \cup A$ then since the table is closed, $\text{row}(s) = \text{row}(s_1)$ for some $s_1 \in S$. Now $\text{row}(s_1) \in F$ if and only if $T(s_1) = 1$, which is true if and only if $T(s) = 1$, since $\text{row}(s) = \text{row}(s_1)$.

In the induction step we assume the lemma holds for all e with $|e| \leq k$. Let $e \in E$ with $|e| = k + 1$. Since E is suffix closed, $e = a \cdot e_1$ for some $a \in A$ and some $e_1 \in E$ of length k. Let s be any element of $(S \cup S \cdot A)$. Because the observation table is closed, there exists a string $s_1 \in S$ such that $\text{row}(s) = \text{row}(s_1)$. Then,

$$\begin{aligned}
\delta(q_0, s \cdot e) &= \delta(\delta(q_0, s), a \cdot e_1), \\
&= \delta(\text{row}(s), a \cdot e_1), && \text{by preceding lemma,} \\
&= \delta(\text{row}(s_1), a \cdot e_1), && \text{since } \text{row}(s) = \text{row}(s_1), \\
&= \delta(\delta(\text{row}(s_1), a), e_1), \\
&= \delta(\text{row}(s_1 \cdot a), e_1), && \text{by definition of } \delta, \\
&= \delta(\delta(q_0, s_1 \cdot a), e_1), && \text{by preceding lemma,} \\
&= \delta(q_0, s_1 \cdot a \cdot e_1).
\end{aligned}$$

By the induction hypothesis on e_1, $\delta(q_0, s_1 \cdot a \cdot e)$ is in F if and only if $T(s_1 \cdot a \cdot e) = 1$. Since $\text{row}(s) = \text{row}(s_1)$ and $a \cdot e_1 = e$ is in E, $T(s_1 \cdot a \cdot e) = T(s \cdot a \cdot e) = T(s \cdot e)$. Therefore, $\delta(q_0, s \cdot e) \in F$ if and only if $T(s \cdot e) = 1$, as claimed by the lemma.

■

Lemma 13.3 *Assume that (S,E,T) is a closed, consistent observation table. Suppose M(S,E,T) has n states. If $M' = (Q', q_0', F', \delta')$ is any dfa consistent with T that has n or fewer states, then M' is isomorphic to $M(S, E, T)$.*

Proof: The proof consists of exhibiting an isomorphism.

For each $q' \in Q'$, define $\text{row}(q')$ to be the function f from E to {0,1} such that $f(e) = 1$ if and only if $\delta'(q', e) \in F'$.

Since M' is consistent with T, for each $s \in (S \cup S \cdot A)$ and each $e \in E$, $\delta'(q_0', s \cdot e) \in F'$ if and only if $T(s \cdot e) = 1$. Also, since $\delta'(q_0', s \cdot e) = \delta'(\delta'(q_0', s), e)$ we know that $\delta'(\delta'(q_0', s), e) \in F'$ if and only if $T(s \cdot e) = 1$. Therefore, $\text{row}(\delta'(q_0', s)) = \text{row}(s)$ in $M(S, E, T)$. As s ranges over all of S, $\text{row}(\delta'(q_0', s))$ ranges over all elements of Q, implying that M' must have at least n states. Since the statement of the lemma presumed that M' had n or fewer states we can conclude that M' has exactly n states.

Thus, for each $s \in S$ there is a a unique $q' \in Q'$ for which $\text{row}(s) = \text{row}(q')$, namely, $\delta'(q_0', s)$. We can define a one-to-one and onto mapping between the states of $M(S, E, T)$ and the states Q' of M'. Specifically, for each $s \in S$, we define $\phi(\text{row}(s)) = \delta'(q_0', s)$. It remains to be shown that this mapping takes q_0 to q_0', that it preserves the transition function and that it takes F to F'. Each of these can be verified in a straightforward way using the definitions above. The details can be found in the paper by Angluin [2].

■

Lemma 2 and Lemma 3 together prove the two parts of Theorem 1. Namely, Lemma 2 shows that $M(S,E,T)$ is consistent with T and Lemma 3 shows that any dfa consistent with T is either isomorphic to $M(S,E,T)$ or has more states. Thus $M(S,E,T)$ is a smallest dfa consistent with T.

13.4.2 Termination of L^*

In the previous subsection we ignored the question of whether or not L^* will terminate and concentrated on showing that if it terminates, the output will be correct. In this subsection we address the issue of termination.

To see that the algorithm will terminate we need the following lemma. At first glance the lemma appears similar to the last lemma of the previous section. The key difference is that the following lemma does not assume that the observation table is closed and consistent. It is important to consider this more general case because as the algorithm proceeds, there will be times when the observation table is not closed and consistent.

Lemma 13.4 *Let (S,E,T) be an observation table. Let n denote the number of different values of* row(s) *for $s \in S$. (Note that if (S,E,T) is closed and consistent, then this will be the number of states in $M(S,E,T)$.) Any dfa consistent with T must have at least n states.*

Proof:

Let $M = (Q, \delta, q_0, F)$ be a dfa consistent with T. Define $f(s) = \delta(q_0, s)$ for every $s \in S$. In other words, $f(s)$ is the final state of M when run with input s. Suppose s_1 and s_2 are elements of S such that row$(s_1) \neq$ row(s_2). Then there exists an $e \in E$ such that $T(s_1 \cdot e) \neq T(s_2 \cdot e)$. Since M is consistent with T, exactly one of $\delta(q_0, s_1 \cdot e)$ and $\delta(q_0, s_2 \cdot e)$ is in F. Thus, $\delta(q_0, s_1 \cdot e)$ and $\delta(q_0, s_2 \cdot e)$ must be distinct states, implying that $f(s_1 \cdot e) \neq f(s_2 \cdot e)$. Since there are n different values of row(s), $f(s)$ must take on at least n distinct values. Thus M has at least n states.

■

Let n be the number of states in a minimum dfa M_U for the unknown language U. To prove termination we show that the number of distinct values of row(s) for $s \in S$ increases monotonically up to n as L^* runs.

First, consider what happens when a string is added to E because the table is not consistent. The number of distinct values of row(s) must increase by at least one. Two previously equal values, row(s_1) and row(s_2), are no longer equal after E is augmented. Any two unequal values will remain unequal.

Second, consider what happens when a string $s_1 \cdot a$ is added to S because the table is not closed. By definition, row$(s_1 \cdot a)$ differs from row(s) for all $s \in S$. Thus the number of distinct values of row(s) is increased by at least one.

From these two situations we can conclude that the total number of operations of either type over the entire run of L^* is at most $n-1$. (There is initially one value of row(s) and there cannot be more than n.) Thus the algorithm will enter the while loop at most $n-1$ times. This means that L^* always eventually finds a closed, consistent observation table (S,E,T) and makes an equivalence query with $M(S,E,T)$.

We must show that the number of equivalence queries is limited (i.e., that the algorithm does not get stuck in the repeat loop). If an equivalence query $M(S, E, T)$ is incorrect with counterexample t, then by Theorem 1, M_U must have at least one more state than $M(S, E, T)$. Furthermore, L^* must eventually make another equivalence query $M(S', E', T')$ which is consistent with T (since T' extends T) and also classifies t the same as M_U (since $t \in S'$ and $\lambda \in E$). This implies that $M(S', E', T')$ is inequivalent to $M(S, E, T)$ and thus has at least one more state that $M(S, E, T)$.

From this we conclude that L^* can make at most $n - 1$ incorrect equivalence queries, since the number of states in the successive queries is monotonically increasing from one and cannot exceed $n - 1$. Since L^* will eventually make another equivalence query, it will terminate with a correct query.

13.4.3 Runtime complexity of L^*

The runtime of the algorithm depends in part of the length of the longest counterexample. We let m be the length of the longest counterexample and analyze the runtime in terms of m and n, the number of states in a minimum dfa for the unknown language. In addition, we let k denote the cardinality of the alphabet A.

First we determine the space needed by the observation table. Initially, $|S| = |E| = 1$. Each time (S, E, T) is discovered to be not closed, $|S| \rightarrow |S| + 1$. Each time (S, E, T) is discovered to be not consistent, $|E| \rightarrow |E| + 1$. For each counterexample of length at most m, at most m strings are added to S.

From this and the analysis of the termination of L^* we see that $|E| \leq n$ and for all $e \in E$, $|e| \leq n - 1$. Also, $|S| \leq n + m(n - 1)$. The first term results because the observation table may be discovered to be not closed at most $n - 1$ times. The second term results because there may be at most $n - 1$ counterexamples, each of which adds at most m strings to S. The maximum length of any string in S is increased by one each time the table is found not be to closed. Thus for all $s \in S$, $|s| \leq m + n - 1$.

Thus, the table size, $|(S \cup S \cdot A) \cdot E|$ is at most

$$(k + 1)(n + m(n - 1))n = O(mn^2).$$

The maximum length of any string in $(S \cup S \cdot A) \cdot E$ is at most

$$(m + n - 1) + 1 + n - 1 = m + 2n - 1 = O(m + n).$$

Thus the observation table takes space $O(m^2n^2 + mn^3)$.

Now we determine the time needed for the computation performed by L^*. Checking if the observation table is closed and consistent can be done in time polynomial in the size of the table. This is done at most $n - 1$ times. Adding a string to S or E requires at most $O(mn)$ membership queries of strings of length at most $O(m + n)$. The total number of membership queries is $O(mn^2)$. Given a closed and consistent table, $M(S, E, T)$ can be constructed in time polynomial in the size of the table. This must be done at most $n - 1$ times. Thus the computation time is polynomial.

Finally, note that if the counterexamples are always of the minimum possible length then $m \leq n$ and all results are polynomial in n. The algorithm which tests for the equivalence of two dfa's can identify a counterexample of minimum length. Also, the bound on the number of membership queries was improved to $O(kn^2 + n \log m)$ by Rivest and Schapire [36].

CS 582T Computational Learning Theory **Spring Semester, 1991**

Topic 18: The Weighted Majority Algorithm

Lecturer: Sally Goldman *Scribe: Marc Wallace*

18.1 Introduction

In these notes we study the construction of a prediction algorithm in a situation in which a learner faces a sequence of trials, with a prediction to be made in each, and the goal of the learner is to make few mistakes. In particular, consider the case in which the learner has reason to believe that one of some pool of known algorithms will perform well, but the learner does not know which one. A simple and effective method, based on weighted voting, is introduced for constructing a compound algorithm in such a circumstance. This algorithm is shown to be robust with respect to errors in the data. We then discuss other situations in which such a compound algorithm can be applied. The material presented here comes from the paper "The Weighted Majority Algorithm," by Nick Littlestone and Manfred Warmuth [30].

Before describing the weighted majority algorithm in detail we briefly discuss some learning problems in which the above scenario applies. The first case is one that we have seen before. Suppose one knows that the correct prediction comes from some target concept selected from some 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. As we shall see for such situations, the weighted majority algorithm is just the natural generalization of the halving algorithm. Another situation in which the weighted majority algorithm can be used in the situation in which one has a set of algorithms for making predictions, one of which will work well on a given data set. However, it may be that the best prediction algorithm to use depends on the data set and thus it is not known a priori which algorithm to use. Finally, as we shall discuss in more detail below 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.

The basic idea behind the weighted majority algorithm is to give each algorithm in the pool the input instance, get all the predictions, combine these into one prediction, and then if wrong pass that data along to the algorithms in the pool. Observe that the halving algorithm could be considered as a weak master algorithm by simply taking the majority prediction and throwing out all bad predictors at each mistake. Unfortunately this algorithm is not robust against noise. If one algorithm is perfect, but that source is noisy, then the main algorithm will reject it and may never converge to one solution (it will run out of algorithms).

We define the number of *anomalies* in a sequence of trials as the least number of inconsistent trials of any algorithm in the pool. Notice if there are any anomalies, then the halving

algorithm will fail to converge by eliminating all functions.

18.2 Weighted Majority Algorithm

In this section we introduce the weighted majority algorithm.

Weighted Majority Algorithm (WM)

Initially assign non-negative weights (say 1) to each algorithm in the pool
To make a prediction for an instance:
Let q_0 be the total weight of algs that predict 0
Let q_1 be the total weight of algs that predict 1
Predict 0 iff $q_0 \geq q_1$
If a mistake is made, multiply the weights of the algorithms agreeing with the incorrect prediction by some fixed non-negative $\beta < 1$.

Notice that using $\beta = 0$ would yield the halving algorithm. However, it is more interesting to consider what happens as $\beta \to 1$. Suppose the pool is called F, and a given sequence of trials has m anomalies with respect to F. Then

$$\#\text{ mistakes } \leq c(\log |F| + m)$$

where c is some constant dependent on β.

In general the maximum number of mistakes will be:

- $O(\log |F| + m)$ if one algorithm in F makes at most m mistakes.
- $O(\log(|F|/k) + m)$ if each of a set of k algorithms in F makes at most m mistakes.
- $O(\log(|F|/k) + (m/k))$ if the total number of mistakes of a set of k algorithms in F is at most m mistakes.

As an application of WM (weighted majority), we return to the use of WINNOW1 to learn r-of-k threshold functions. Normally the mistake bound is $O(kn \log n)$. However, if the learner has prior knowledge of a close upper bound for r, the mistake bound can be reduced to $b_r = O(kr \log n)$ by carefully selecting the parameters for WINNOW1. Let A_r denote the algorithm WINNOW1 with a guess of r. Now any sequence that cannot be represented by an r'-of-k function for $r' \leq r$ will fail phenomenally, and make far too many mistakes. However, we can apply WM to the set of algorithms $\{A_{2^i}\}$ for all i satisfying $2^i \leq n$. Then the number of mistakes will be $O(\log \log n + b_r) = O(kr \log n)$.

Now clearly if the size of our algorithm pool is not polynomial the weighted majority algorithm is not computationally feasible. Still, many reasonable pools of algorithms will be of polynomial size, or can be reduced to such without a dramatic increase in the average number of mistakes.

18.3 Analysis of Weighted Majority Algorithm

Let $A = \{A_1, \ldots, A_{|A|}\}$ be the pool of algorithms, and w_i the weight of A_i. The basic structure of the proofs that follow will be: First, show that after each trial, if a mistake occurs then the sum of the weights is decreases by at least a factor of u for some $u < 1$. Second, let W_{init} be the total initial weight and W_{fin} be a lower bound for the total final weights; then $W_{init}u^m \geq W_{fin}$, where m is the number of mistakes. This implies that

$$m \leq \frac{\log \frac{W_{init}}{W_{fin}}}{\log \frac{1}{u}}.$$

We now prove a general theorem bounding the number of mistakes made by the weighted majority algorithm. We can then apply to obtain the three results given above.

Theorem 18.1 *Let S be a sequence of instances and m_i the number of mistakes made by A_i on S. Let m be the number of mistakes made by WM on S when applied to the pool A. Then*

$$m \leq \frac{\log \frac{W_{init}}{W_{fin}}}{\log \frac{2}{1+\beta}}$$

where $W_{fin} = \sum_{i=1}^{n} w_i \beta^{m_i}$.

Notice that if the initial weights are all 1, we have

$$m \leq \frac{\log \frac{|A|}{\sum_{i=1}^{n} \beta^{m_i}}}{\log \frac{2}{1+\beta}}$$

Proof: We notice that the weights are updated only when their algorithm makes a mistake; and since A_i can make only m_i mistakes, we must have $w_{i_{final}} \geq w_{i_{init}}\beta^{m_i}$. Furthermore, at the end the total current weight must be greater than or equal to W_{fin}.

Now, during each trial, let $q_0 + q_1$ be the current total weight, where q_0 and q_1 are as defined in the definition of WM. Suppose we have a trial in which a mistake was made. Without loss of generality we assume the prediction was zero. Then the total weight after the trial will be:

$$\beta q_0 + q_1 \leq \beta q_0 + q_1 + \frac{1-\beta}{2}(q_0 - q_1) = \frac{1+\beta}{2}(q_0 + q_1)$$

If no mistake is made then the weights remain the same, of course. So we have $u = \frac{1+\beta}{2} < 1$.

Finally, since we know $W_{init}u^m \geq W_{fin}$ we take logs of both sides and get

$$m \leq \frac{\log \frac{W_{init}}{W_{fin}}}{\log \frac{2}{1+\beta}}$$

■

Now we provide some (more useful) corollaries. Assume $\beta > 0$ and all initial weights are one. Let $|A| = n$.

Corollary 18.1 *Assume there is a subpool of A (of size k) such that each algorithm in it makes no more than m mistakes on a set of instances S. Then WM applied to A makes no more than*

$$\frac{\log \frac{n}{k} + m \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$$

mistakes on S.

Proof: $W_{init} = n$ and $W_{fin} \geq k\beta^m$. Apply the theorem. ■

The theorem and the corollary give us the first two relations stated above concerning the order of the mistake bounds: $O(\log |F| + m)$ if one algorithm in F makes at most m mistakes, and $O(\log(|F|/k) + m)$ if each of a set of k algorithms in F makes at most m mistakes. The next corollary will yield the third relation. However, before proving this corollary we review some basic definitions and lemmas.

Definition 18.1 *A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is concave (respectively convex) over an interval D of $\mathbb{R}$ if for all $x \in D$, $f''(x) \geq 0$ ($f''(x) \leq 0$).*

The following to standard lemmas [20, 32] are often useful.

Lemma 18.1 *Let f be a function from $\mathbb{R}$ to $\mathbb{R}$ that is concave over some interval D of $\mathbb{R}$. Let q be a natural number, and let $x_1, x_2, \ldots, x_q \in D$. Then*

$$\sum_{i=1}^{q} x_i \leq S \;\Rightarrow\; \sum_{i=1}^{q} f(x_i) \geq q f(S/q).$$

Lemma 18.2 *Let f be a function from $\mathbb{R}$ to $\mathbb{R}$ that is convex over some interval D of $\mathbb{R}$. Let q be a natural number, and let $x_1, x_2, \ldots, x_q \in D$. Then*

$$\sum_{i=1}^{q} x_i \leq S \;\Rightarrow\; \sum_{i=1}^{q} f(x_i) \leq q f(S/q).$$

We are now ready to prove the corollary.

Corollary 18.2 *Assume there is a subpool of A (of size k) such that all algorithms (together) make no more than m mistakes on a set of instances S. Then WM applied to A makes no more than*

$$\frac{\log \frac{n}{k} + \frac{m}{k} \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}$$

mistakes on S.

Proof: Without loss of generality we can assume the first k algorithms of A are the subpool in question. Then $\sum_{i=1}^{k} m_i \leq m$. Since the sum of the final weights is greater than the sum of k of the final weights (or equal), we have

$$w_{fin} \geq \sum_{i=1}^{k} w_{i_{fin}} \geq \sum_{i=1}^{k} \beta^{m_i}$$

Finally, from Lemma 18.1 it follows that

$$\sum_{i=1}^{k} \beta^{m_i} \geq k\beta^{\frac{m}{k}}$$

■

As a final note, if $\beta = 0$ and $m = 0$ the number of mistakes made will be no more than $\log_2 \frac{n}{k}$.

18.4 Variations on Weighted Majority

In this section we demonstrate the generality of the weighted majority algorithm by discussing some useful learning situation in which it can be applied.

18.4.1 Shifting Target

We modify WM so that it can recover more easily if the target is changed. For example, suppose a particular subset of algorithms predicts well in the beginning, but after some period of time some other set of algorithms has the best performance (and the initial set may yield terribly wrong predictions). We want the overall performance to remain good.

For simplicity of notation and mathematics, assume that the subpools which do well (either initially or after a while) are all of size one: it is not too difficult to generalize but the basic ideas can easily be lost in the dredge of notation.

Our new algorithm will be called WML. WML has two basic parameters, β and γ, with $0 < \beta < 1$ and $0 \leq \gamma < 1/2$. Each algorithm in A has a non-negative weight associated with it. Everything is basically the same as WM except for the updating: whenever WM would multiply a weight by β, WML will multiply by β if and only if the weight is larger than $\gamma/|A|$ times the total weight of all algorithms in A.

The idea, therefore, is to follow the same updating scheme, unless a weight is already extremely low. If it is then it is probably negligible, so do not bother updating it. This allows us to take a previously poorly performing algorithm (which has just started predicting extremely well) and put lots of weight on it, since its weight has not been allowed dropped too low.

Now suppose that some algorithm makes m_1 mistakes in the first segment of some sequence of instances, another makes m_2 mistakes in the next segment, and so on. Of course, WML does not know when the segments start and finish. Still, the number of mistakes will be no greater than

$$c(S \log |A| + \sum_{i=1}^{s} m_i)$$

where S is the number of segments there are and c is some constant.

This mistake bound makes sense, since the $S \log |A|$ part is really $\sum_{i=1}^{S} \log |A|$. When there is a change between segments, the WML will make some constant times $\log |A|$ mistakes

while the weights are updated so that the (currently) accurate algorithm receives the majority of the weight. The other m_i is from the accurate algorithms themselves, as according to the bounds for WM.

18.4.2 Selection from an Infinite Pool

Suppose we have a (countably) infinite pool of algorithms from which to choose. Clearly we cannot ask for input from all of them. But by using ever increasing sets of the algorithms one can approximate the standard WM model, with only an additional term of order $\log i$ where it is the ith algorithm which predicts well.

Previously (in homework 3, problem 1) we saw that one could apply the halving algorithm to such an infinite set, and achieve a mistake bound of order $\log i$. Since the WM is merely an extension of the halving algorithm, it is possible to construct a variant of the WM algorithm which increases its pool over time, in which the number of mistakes is bounded by $c(\log i + m_i)$, where the constant c depends not only on the initial parameters, but on how big the initial set of algorithms grows.

18.4.3 Pool of Functions

We now consider the application of WM to a pool of Boolean functions on some domain. This is a true generalization of the halving algorithm. Now, suppose there is a function consistent with the sequence of trials. We wish to obtain better bounds for the number of mistakes on this function, at the possible expense of greater numbers of mistakes on the other functions. Hence, the following theorem:

Theorem 18.2 *Let $f_1, \ldots, f_n$ be a pool of functions, and $m_1, \ldots, m_n$ be integers such that $\sum_{i=1}^{n} 2^{-m_i} < 2$. If WM is applied to the pool with initial weights $w_j = 2^{m_i}$ and $\beta = 0$, and if the sequence of trials is consistent with some f_i, then WM will make no more than m_i mistakes.*

Proof: Recall from the discussion of WM that for m the number of mistakes, we have

$$m \leq \frac{\log \frac{W_{init}}{W_{fin}}}{\log \frac{2}{1+\beta}}$$

where $W_{fin} = \sum_{i=1}^{n} w_i \beta^{m_i}$, and m_i is the number of mistakes made by the ith algorithm on a given sequence.

For $\beta = 0$ the denominator drops out. Now, $W_{init} = \sum_{i=1}^{n} 2^{-m_i} < 2$. And since f_i makes no mistakes at all, $W_{fin} \geq$ initial weight of $f_i = 2^{-m_i}$. Thus the number of mistakes m is no more than:

$$m \leq \log W_{init} - \log W_{fin} \leq \log 2 - \log 2^{-m_i} = 1 + m_i$$

which yields the desired result. ■

18.4.4 Randomized Responses

Notice that we can consider WM to be a deterministic algorithm which predicts 1 whenever $\frac{q_1}{q_0+q_1} \geq \frac{1}{2}$.

Consider a randomized version in which 1 is predicted with probability $\frac{q_1}{q_0+q_1}$. This algorithm may make more initial mistakes when the probability is near 1/2, but it can be shown that we can update the weights so that the rate of mistakes (in the long run) can be made arbitrarily close to the rate of mistakes of the best prediction algorithm in the pool. This would yield an improvement of a factor of two over the limiting bounds for the learning rate of the deterministic version of WM.

CS 582T Computational Learning Theory **January 25, 1991**

Homework Assignment 1

Solutions Due: February 8

Please write up all solutions clearly, concisely, and legibly.

1. Consider Boolean functions f that can by defined on $\{0,1\}^n$ by a nested **if–then–else** statement of the form:

   $$f(x_1, x_2, \ldots, x_n) = \textbf{if } l_1 \textbf{ then } c_1 \textbf{ elseif } l_2 \textbf{ then } c_2 \cdots \textbf{elseif } l_k \textbf{ then } c_k \textbf{ else } c_{k+1}$$

   where the l_j's are literals (either one of the variables or their negations), and the c_j's are either **T** (true) or **F** (false). (Such a function is said to be computed by a *simple decision list*.)

   (a) Suppose you have a set of labeled examples that are consistent with a function computed by a simple decision list. Show how to find in polynomial time some simple decision list that is consistent with the examples.

   (b) Show how to generalize your algorithm to handle k-decision lists, where the condition in each **if** statement may be the conjunction of up to k literals. (Here k is a fixed constant.)

   (c) Argue the k-decision lists are PAC learnable.

2. We have seen in class that an algorithm that is capable of finding a hypothesis consistent with a given set of labeled examples can be turned into a PAC learning algorithm. Argue that the converse is true: a PAC learning algorithm can be used (with high probability) to find a hypothesis consistent with a given set of labeled examples. (Hint: Show how to define an appropriate probability distribution on the given set of examples, and how to set ϵ and δ, so that the PAC learning algorithm returns a hypothesis consistent with the given set of examples, with high probability.)

3. Describe a PAC learning algorithm for the class of concepts k-RS(n) (the RS is for "ring-sum"); this class contains all boolean formula over the n boolean variables $\{x_1, \ldots, x_n\}$, where each such formula is the exclusive-or of a set of terms, and each term is the conjunction of at most k literals. (Once again, think of k as a fixed constant.) For example, the formula

   $$x_1\overline{x_3}x_5 \oplus x_2 \oplus \overline{x_4}$$

   is a member of 3-RS(5). Be sure to describe how many examples are required as a function of n, k, ϵ, and δ, and how the learning algorithm computes its hypothesis. (Hints: For the former, base your computation on the number of formula in k-RS(n). For the later, define a variable for each possible term, and solve linear equations modulo 2.)

4. Consider the class of concepts defined on the interval on the interval $(0, 1)$ where each concept is of the form $(0, r]$ for some real $r \in (0, 1)$. For the concept $(0, r]$, all real numbers x, $0 < x \leq r$, are *positive* instances while all real numbers y, $r < y < 1$, are *negative* instances. There is one such concept for each real number $r \in \{0, 1\}$.

   (a) Show that this concept class is PAC learnable under the assumption that the examples are drawn *uniformly* from the range $(0, 1)$. Specify the number of examples needed as a function of δ and ϵ, and explain how you construct your hypothesis from the given data.

   (b) Show that this concept class is PAC learnable when the examples are drawn according to some *unknown* distribution. Specify the number of examples needed as a function of δ and ϵ, and explain how you construct your hypothesis from the given data.

5. Let y be a Bernoulli random variable that is 1 with probability p. Our goal is to compute a good estimate for p from observing a set of independent draws of y. Let $\hat{p}$ be an estimate of p obtained from m independent trials. That is $\hat{p}$ is just:

   $$\text{(number of trials in which } y = 1) \text{ / (total number of trials)}$$

   How large must m be so that $\Pr[|\hat{p} - p| \geq \gamma] \leq \delta$. That is we want our estimate for p to be within γ of the true value for p with probability at least $1 - \delta$.

CS 582T Computational Learning Theory **February 20, 1991**

Homework Assignment 2

Solutions Due: March 6

Please write up all solutions clearly, concisely, and legibly.

1. We return to the class of concepts defined on the interval $(0, 1)$ where each concept is of the form $(0, r]$ for some real $r \in (0, 1)$. Suppose that the source of labeled examples is subject to random misclassification noise of a rate $\beta < 1/4$. Give a PAC learning algorithm for this problem. Be sure to specify how many examples are needed as a function of ϵ and δ (and β if you wish).

2. If C_1 and C_2 are concept classes, and $c_1 \in C_1$ and $c_2 \in C_2$, then $c_1 \vee c_2$ is the concept whose positive examples are exactly the set $c_1 \cup c_2$. Note that $c_1 \vee c_2$ may not be an element of either C_1 or C_2. We can then define the concept class $C_1 \vee C_2 = \{c_1 \vee c_2 : c_1 \in C_1, c_2 \in C_2\}$. The definition for the class $C_1 \wedge C_2$ is analogous.

   (a) Prove that if C_1 is PAC learnable, and C_2 is PAC learnable from negative examples, then $C_1 \vee C_2$ is PAC learnable.

   (b) Prove that if C_1 is PAC learnable from positive examples and C_2 is PAC learnable from positive examples then $C_1 \wedge C_2$ is PAC learnable from positive examples.

3. Let C, C_1, C_2 be concept classes of finite VC-dimension over instance space X.

   (a) Define $\overline{C} = \{X - c : c \in C\}$. What is $VCD(\overline{C})$ in terms of $VCD(C)$?

   (b) For concept class C, let $VCD(C) = d$ and let S be any set of m distinct instances. Prove that $|\Pi_C(S)| \leq \Phi_d(m) = \sum_{i=0}^{d} \binom{m}{i}$. (Recall that $\Pi_C(S)$ includes all distinct ways that C can classify the S instances.)

   Hint: Use induction on m and d. (Think about $m = d$ as the base case for any given d.) Also, it may be useful to show that $\Phi_d(m) = \Phi_d(m-1) + \Phi_{d-1}(m-1)$.

   (c) For concept classes C_1, C_2, prove that $VCD(C_1 \cup C_2) \leq VCD(C_1) + VCD(C_2) + 1$. Also provide an example to demonstrate that this result is the best possible such result.

4. Show how to exactly identify any unknown read-once-DNF formula using a polynomial number of equivalence and membership queries. In a read-once formula each variable occurs at most once. (The one instance of a variable may be either negated or unnegated—so a read-once formula need not be a monotone formula.) The number of queries should be polynomial in the number of terms and variables in the unknown formula.

Big Hint: Prove as a lemma that if f is the unknown read-once-DNF formula over the variables $X_1, \ldots, X_n$, and $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ are two assignments to these variables such that x and y agree in all but one variable assignment, yet $f(x) = 1$ and $f(y) = 0$, then $S(x)$ is a term of f, where $S(x)$ is the conjunction of the literals in the set

$$\{X_i : f(x \oplus i) = 0 \text{ and } x_i = 1\} \cup \left\{\overline{X_i} : f(x \oplus i) = 0 \text{ and } x_i = 0\right\}$$

where $x \oplus i$ is the assignment x with the ith variable flipped. (For example $00110 \oplus 3 = 00010$.)

EXTRA CREDIT:

Let C be a concept class of finite VC-dimension over instance space X. Let

$$C^{(k)} = \{\cup_{i=1}^{k} c_i : c_i \in C\}.$$

Prove that $VCD(C^{(k)}) = O(k \ln(k) \cdot VCD(C))$. (Note the same result holds for k-way intersection.)

CS 582T Computational Learning Theory **March 27, 1991**

Homework Assignment 3

Solutions Due: April 9

Please write up all solutions clearly, concisely, and legibly.

1. Suppose that $f_1, f_2, \ldots$ is an enumeration of some set of computable total 0-1-valued functions. More precisely, this enumeration is computed by some program M; the output of M on input i is a program P_i that computes f_i.

   Show how to learn an unknown function f, which is guaranteed to be one of the f_i's, in the mistake-bound model described in the Littlestone paper in such a way that you make only $O(\lg t)$ mistakes, where t is the least index such that $f = f_t$.

2. In this problem we considered a simple case of learning with queries where the feedback can be erroneous. The learner and adversary agree on a number n, and then the adversary thinks of a number between 1 and n, inclusive. The learner must find out which number the adversary has selected by asking questions of the form, "Is your number less than t?" for various t. A binary-search approach allows you to ask at most $\lg n$ questions before finding the number.

   To make this an interesting problem, suppose that the adversary is allowed to incorrectly respond to at most *one* question. How many questions must the learner now ask? (A bound of $3 \lg n$ is easy: the learner can just ask each question three times and take the majority vote of the adversary's responses.) Give a learning algorithm that uses a number of queries of the form $\lg n + f(n)$ where $f(n) = o(\lg n)$, that is, f grows asymptotically more slowly than the logarithm function.

3. Argue that functions representable as 1-decision lists are threshold functions. That is, given any 1-decision list $f(x_1, x_2, \ldots, x_n) =$ **if** l_1 **then** c_1 **elseif** l_2 **then** $c_2 \cdots$ **elseif** l_k **then** c_k **else** c_{k+1}, show how to construct a threshold function equivalent to this decision list. One way to achieve this goal is to do the following:

   (a) Show that, without loss of generality, you can assume that $c_k = 1$ and $c_{k+1} = 0$.

   (b) Suppose that the threshold circuit not only has access to the x_i for all i, but also has access to the negated variables $\overline{x_i}$ for all i. (We will remove this additional information in the next step.) Given this assumption, describe a threshold function equivalent to f.

   (c) Now modify the previous part so that the threshold function does not have access to the negated variables.

   Observe that now we can use Littlestone's Winnow2 combined with the transformation described in Example 6 from Littlestone's paper (pg. 312) to learn 1-decision lists in the mistake-bound model.

4. In this problem we consider r-of-k threshold functions. (Recall that for this concept class, $X = \{0,1\}^n$. For a chosen set of k ($k \leq n$) relevant variables and a given number r ($1 \leq r \leq k$), an r-of-k threshold function is 1 if and only if at least r of the k relevant variables are 1.

   Assuming that both r and k are unknown to the learner, show that the class of r-of-k threshold functions can be learned in the mistake-bound model using the halving algorithm. What mistake bound do you obtain? How does this compare to the mistake bound obtained by Winnow2? (That is, does one algorithm always perform better than the other algorithm? If not, state under what conditions each algorithm is best.)

Bibliography

[1] Dana Angluin. Learning k-term DNF formulas using queries and counterexamples. Technical Report YALEU/DCS/RR-559, Yale University Department of Computer Science, 1987.

[2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

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

[4] Dana Angluin. Equivalence queries and approximate fingerprints. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 134–145, August 1989.

[5] Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of horn clauses. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, pages 186–192, October 1990.

[6] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning read-once formulas with queries. Technical Report UCB/CSD 89/528, University of California Berkeley Computer Science Division, 1989.

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

[8] Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155–193, April 1979.

[9] Javed A. Aslam and Ronald L. Rivest. Inferring graphs from walks. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 359–370, 1990.

[10] Avrim Blum. Learning boolean functions in an infinite attribute space. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 64–72, May 1990.

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

[12] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[13] M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for estimating the volumes of convex bodies. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 375–381, May 1989.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[15] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[16] Sally A. Goldman, Ronald L. Rivest, and Robert E. Schapire. Learning binary relations and total orders. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 46–51, October 1989.

[17] Sally A. Goldman and Robert H. Sloan. The difficulty of random attribute noise. Technical Report WUCS–91–29, Washington University, Department of Computer Science, June 1991.

[18] Sally Ann Goldman. *Learning Binary Relations, Total Orders, and Read-once Formulas.* PhD thesis, MIT Dept. of Electrical Engineering and Computer Science, July 1990.

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

[20] G.H. Hardy, J.E. Littlewood, and G. Pólya. *Inequalities.* Cambridge University Press, Cambridge, 1959.

[21] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation.* To appear. A preliminary version is available in *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 42–55, 1988.

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

[23] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewritting systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 30–45, 1977.

[24] Jeff Kahn and Michael Saks. Balancing poset extensions. *Order 1*, pages 113–126, 1984.

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

[26] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 433–444, May 1989.

[27] Michael J. Kearns. *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, Massachusetts, 1990.

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

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

[30] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 256–261, October 1989.

[31] Peter Matthews. Generating a random linear extension of a partial order. Unpublished manuscript, 1989.

[32] D.S. Mitrinović. *Analytic Inequalities*. Springer-Verlag, New York, 1970.

[33] Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *Journal of the Association for Computing Machinery*, 35(4):965–984, 1988.

[34] Leonard Pitt and Manfred Warmuth. Reductions among prediction problems: On the difficulty of predicting automata (extended abstract). In *3rd IEEE Conference on Structure in Complexity Theory*, pages 60–69, June 1988.

[35] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[36] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 411–420, May 1989.

[37] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 1990. Special issue for COLT 89, to appear. A preliminary version is available in *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 28–33,1989.

[38] George Shackelford and Dennis Volper. Learning k-DNF with noise in the attributes. In *First Workshop on Computatinal Learning Theory*, pages 97–103, Cambridge, Mass. August 1988. Morgan Kaufmann.

[39] Alistair Sinclair. *Randomised Algorithms for Counting and Generating Combinatorial Structures*. PhD thesis, University of Edinburgh, Department of Computer Science, November 1988.

[40] Robert H. Sloan. Some notes on Chernoff bounds. (Unpublished), 1987.

[41] Robert H. Sloan. Types of noise in data for concept learning. In *First Workshop on Computational Learning Theory*, pages 91–96. Morgan Kaufmann, 1988.

[42] Leslie Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:198–201, 1979.

[43] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[44] Leslie Valiant. Learning disjunctions of conjunctions. In *Proceedings of Ninth International Joint Conference on Artificial Intelleigence*, 1985.