Recent Results on Boolean Concept Learning

Michael Kearns
Harvard University

Ming Li
Harvard University and Ohio State University

Leonard Pitt
University of Illinois at Urbana-Champaign

Leslie Valiant
Harvard University

Abstract

Recently, a new formal model of learnability was introduced [23]. The model is applicable to practical learning systems because it requires the learning algorithm to be feasibly computable, yet at the same time demands only that the algorithm find an approximation to the unknown rule. We survey recent results in this new area of theoretical induction, giving both positive (learnability) and negative (non-learnability) results, as well as outlining useful techniques for proving learnability. Our main focus is the application of the model to the problem of learning boolean formulae.

1 Introduction

One of the main difficulties in comparing various algorithms which learn from examples is the lack of a formally specified model by which the algorithms may be evaluated. Typically, different learning algorithms and theories are given together with examples of their performance, but without a precise definition of "learnability" it is difficult to characterize the scope of applicability of an algorithm or analyze the success of different approaches and techniques.

Although various formal models of inductive learning have been given in the past, they have not been widely accepted by the machine learning community. One explanation is that the definitions fail to capture those aspects perceived to be essential for practical learning systems. Another explanation is that the domains of investigation (*e.g.* the inductive inference of classes of recursive functions) have been significantly removed from the types of domains typically of interest, and few results can be transferred.

Recently, in [23], a new formal definition of concept learning from examples was introduced, which we will refer to as the "learnability model." In this paper we summarize subsequent work by the authors and others in this new area of induction. The main focus here will be the learnability of propositional formulae from examples. The learnability model itself is not restricted, however, to any one knowledge representation or learning mode. After introducing the learnability model below, we present a simple example of a learning algorithm in section 2. In section 3 we sketch some general tools which are useful for proving that given algorithms satisfy the requirements of the definition, and for determining which types of formulae are learnable. In sections 4 and 5 we summarize known positive (learnability) and negative (non-learnability) results, and conclude with extensions and restrictions of the model in section 6.

There are three important components of the learnability model: **convergence**, **feasibility**, and **approximation**. We motivate these properties by briefly discussing some previous definitions of learning.

A main problem for defining learning from a never-ending stream of examples is that at any point in the learning process, the very next datum may contradict the current hypothesis. Gold [12] (in the context of learning of formal languages) effectively dealt with this issue by introducing the notion of **convergence** in the limit to a correct rule for the data. The learning algorithm is allowed to make many hypotheses, and need not ever *know* when it is correct, as long as past some point in its computation it settles on some correct hypothesis. Subsequent research within Gold's paradigm has not been applicable to practical learning, as typically the complexity (run time) of the learning algorithm has been very high.

There are far fewer results if we insist that the convergence occur within a **feasible** amount of time, *i.e.* that the learning algorithm find a correct rule within time polynomial in the size of the examples seen and of the rule to be learned. The work of [1, 2, 8, 20, 21] stand as notable exceptions to a number of NP-completeness results which point out that this type of "exact" learning is too difficult to achieve in polynomial time when the rule may be any one of a large (exponential) class of possible rules.

To avoid these types of worst case negative results, we consider learning algorithms which are not required to exactly find some unknown rule, but are allowed instead to find a rule which is an **approximation** to the unknown rule. By assuming certain particular input distributions, many techniques of statistical pattern recognition are capable of finding such approximate rules [9]. One problem with this approach, however, is that the input distributions assumed are not necessarily representative of nature. The solution adopted by the learnability model is to assume that there is some fixed probability distribution on the input examples, but that this distribution is *completely arbitrary and need not be known.* Another problem with previous statistical and other approaches is that they often make particular and restrictive assumptions about the knowledge representation. The learnability model avoids this by treating the representation as a parameter.

The Learnability Model

An algorithm is said to learn from examples if it can, in a feasible (polynomial) amount of time, find (with high likelihood) a rule which is reasonably accurate. "Reasonably accurate" means that the rule found by the algorithm will be able to predict future events *drawn from the same distribution* on which it has learned, with controllable error. No assumption is made about the input distribution of examples, except that it is time-invariant.

To formally define learnability, we must say what it is that we wish to learn: Imagine a world with n attributes. For the purposes of this paper, we assume that the attributes are boolean (0-1) valued, although many of the results have extensions in the case of certain types of multi-valued attributes [13]. Each object in our world W consists of a vector $\vec{v}$ of length n, the i^{th} position of the vector being either "1" or "0", indicating that the i^{th} attribute is (respectively is not) present. Thus there are exactly 2^n distinct elements of W. (In its full generality, the model allows vectors to have a "$*$" indicating that the corresponding attribute is unspecified. For simplicity, we exclude this possibility from our present discussion). The model is a hierarchical one, in that there is no requirement that the attributes be primitive inputs or features; they may in fact be boolean concepts which have been previously learned. A *concept* F is a subset of W. Elements of F are *positive* examples of F and elements of $W - F$ are *negative* examples of F. We are not interested in learning concepts, (which are subsets of vectors), but rather representations of concepts. For example, for any F, there are many possible boolean formulae f over the attribute variable space which represent F in the following sense: $f(\vec{v}) = 1$ (*i.e.* f evaluates to TRUE when the values of the vector $\vec{v}$ are substituted for the variables in f) if and only if $\vec{v} \in F$. We are interested in representations for which there are fast (polynomial time) algorithms for testing whether a given example is a member of the concept or not.

The classes of representations we shall consider here will all be classes of boolean formulae of n variables. For example, *monomials* and kDNF are classes of representations of concepts. A monomial is a formula which is the conjunct of any number of the literals representing the available attributes or their negations. A kDNF formula is the disjunct of any number of monomials (also called terms) where each monomial has at most k literals. We shall see that learnability may depend on the representation chosen, hence learnability is defined with respect to a given class of representations. If A is a class of representations, then for each $f \in A$, let $size(f)$ denote the fewest number of symbols needed to write the representation f.

We assume that the learning algorithm has available a black box which is called EXAMPLES(f), with two buttons labeled POS and NEG. If POS (NEG) is pushed, a positive (negative) example is generated according to some fixed but unknown probability distribution D^+ (D^-). We assume nothing about the distributions D^+ and D^-, except that $D^+(\vec{x}) = 0$ for each negative example $\vec{x}$, and $D^-(\vec{x}) = 0$ for each positive example $\vec{x}$. In other words, the probability is zero that a negative example will be generated by pushing POS, or that a positive example will be generated by pushing NEG.

Definition 1 *Let A be a class of representations. Then A is* learnable from examples *iff there exists a polynomial p, and a (possibly randomized) learning algorithm L, such that* $(\forall n)(\forall f \in A)(\forall D^+, D^-)$ $(\forall \epsilon > 0)$,$(\forall \delta > 0)$, *L, given only EXAMPLES(f), halts in time* $p(n, size(f), \frac{1}{\epsilon}, \frac{1}{\delta},)$ *and outputs a representation $g \in A$ that with probability at least* $1 - \delta$, *has the following properties:*

$$\sum_{g(\vec{x})=0} D^+(\vec{x}) < \epsilon$$

and

$$\sum_{g(\vec{x})=1} D^-(\vec{x}) < \epsilon.$$

This definition can be understood as follows. We think of Nature as providing positive and negative examples of the formula to be learned according to some unknown probability distribution for which we can make no assumptions. Since there may be very bizarre examples of the formula which occur with low probability, it is unreasonable to expect the learning algorithm to produce a formula which correctly classifies all examples. Hence a successful formula g is one which approximates the unknown f by agreeing with f on *most* of the distribution. That is, the probability that the formula g will disagree with a positive (negative) example randomly chosen according to D^+ (D^-) is at most ϵ in either case. A second source of error is introduced by the possibility that the particular sequence of examples which were provided by Nature was highly unrepresentative. In this case, it is reasonable that the formula g be highly inaccurate. We require that this occur with probability at most δ. A further requirement of the definition is that the run time of the algorithm (and hence the number of examples drawn) be polynomial in the number of attributes n, the size of the formula to be learned, and in the error parameters $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Thus the formula found by the algorithm cannot be exponentially larger than the unknown formula, and the amount of time required for increased accuracy or confidence should not be exponential. We refer the reader to [6, 23, 24, 25] for further discussion and justification of this model.

If there is an algorithm L as above which never asks for any negative (respectively positive) examples, then we'll say that A is learnable from positive (negative) examples only. Throughout the paper, we shall abbreviate "is learnable from examples" by "is learnable".

A generalization of the above definition allows the formula output by the learning algorithm to be from a different class of representations: the class A is *learnable by the class B* iff Definition 1 holds, except that the output formula g is from B. Thus "A is learnable by A" is equivalent to "A is learnable." We shall see that algorithms which choose their hypotheses from a different (possibly larger) class of formulae than that from which the input formula was taken may be more powerful than algorithms which attempt to find a formula of the same type as the input formula.

In general, the larger the class A, the harder the learning task, since the unknown formula may be any one of a more diverse class of possibilities. Note however that for fixed A, the larger the class B, the *easier* the learning task, since the learning algorithm may now choose from a larger class of representations to express its answer. Thus $A \subseteq B$ does not necessarily imply anything about the learnability of A (by A) as compared to the learnability of B (by B.)

For later reference, we define here the classes of formulae which we shall consider. Let the collection of attributes be $\{x_1, x_2, \ldots, x_n\}$. A *literal* is either the symbol x_i or its negation $\overline{x_i}$ for some i. In the following, let k be any fixed natural number.

monomials: a monomial is a conjunction of literals (*e.g.* $x_1 x_5 \overline{x_6} x_8$.)

DNF: disjunctive normal form formulae: Each formula is a disjunct of terms. A term is a monomial. (*E.g.* $x_1 x_5 \overline{x_6} x_8 \vee x_1 x_3 x_4 \vee \overline{x_5} \vee \overline{x_7} x_3$.)

CNF: conjunctive normal form formulae: Each formula is a conjunct of clauses. Each clause is a disjunct of literals. (*E.g.* $(x_1 \vee x_5 \vee \overline{x_6} \vee x_8) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_5}) \wedge (\overline{x_7} \vee x_3)$.)

internal disjunctive formulae: Here the n attributes are given an arbitrary but fixed partition P. The class of C_P of internal disjunctive formulae consists of all CNF formulae $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in which each C_i contains attributes (or their negations) from exactly one element of the partition P. For example, if $P = \{\{x_1, x_2\}, \{x_3, x_4, x_5\}\}$ then $(x_1 \vee x_2) \wedge (\overline{x_3} \vee x_5)$ is a formula in C_P but $(x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_4)$ is not.

***k*DNF:** DNF formulae where each term consists of at most k literals, with no restriction on the number of terms. (*E.g.* $x_1 x_5 \vee x_1 \overline{x_3} \vee \overline{x_5} \vee \overline{x_7} x_3$ is a 2DNF formula.)

k-term-DNF: DNF formulae of at most k terms, with no restriction on the number of literals within each term. (*E.g.* $x_1x_5\overline{x_3}x_7\overline{x_8} \vee x_2x_4\overline{x_5}x_7$ is a 2-term-DNF formula.)

kCNF: CNF formulae where each clause consists of at most k literals, with no restriction on the number of clauses. (*E.g.* $(x_1 \vee x_5) \wedge (x_2 \vee \overline{x_3}) \wedge (x_7)$ is a 2CNF formula.)

k-clause-CNF: CNF formulae of at most k clauses, with no restriction on the number of literals within each clause. (*E.g.* $(x_1 \vee x_5 \vee \overline{x_3} \vee x_7 \vee \overline{x_8}) \wedge (x_2 \vee x_4 \vee \overline{x_5} \vee \overline{x_7})$ is a 2-clause-CNF formula.)

monotone formulae: A monotone formula is one in which no negations occur; each literal is a positive literal. We may also view monotonicity as an operator, which when applied to classes of formulae yield corresponding classes which contain only the corresponding monotone formulae. Thus monotone DNF is the class of DNF formulae where each literal is positive.

μ-formulae: A μ-formula is a formula in which each variable appears at most once. We also view μ as an operator applied to classes of formulae to get corresponding classes where each variable appears at most once; for example, μDNF is the class of DNF formulae with at most one occurrence of each variable.

The relevance of the class DNF was discussed in [24], and this class has been used in many systems as a natural representation for concepts [16]. A major goal of some of this research has been to determine whether this particular class is learnable. We indicate the implications of a number of results with respect to this tantalizing open problem in the following sections.

Finally, we assume that the reader is acquainted with the complexity-theoretic hypothesis that P $\neq$ NP. Intuitively, P is the set of problems which have solutions computable by polynomial time algorithms, and NP are those which can be solved by *nondeterministic* polynomial time algorithms. If in fact P = NP, then every reasonable class of concepts would be learnable, and in addition, hundreds of problems now thought to be intractable would have efficient solutions. If a problem is NP-hard, then an efficient algorithm for it would provably imply that P = NP. Hence a proof of NP-hardness is conventionally taken as evidence that the problem is in fact intractable. For technical reasons, the results of section 5 rely on the slightly stronger hypothesis that RP $\neq$ NP, where RP is essentially the set of problems for which there are polynomial time *randomized* algorithms which solve the problem with high probability. It would be just as startling if this stronger hypothesis were false. The reader is referred to [10] for further details concerning these classes.

2 An Example

In this section, we illustrate how the learnability model works in the very basic case of monomials. In later sections we discuss larger learnable classes that are better suited to the real world.

Suppose we are interested in a set of boolean attributes describing the animal kingdom. For concreteness, we will give the attributes descriptive names, rather than referring to them with abstract symbols such as x_i. The attribute set for animals might include attributes describing the physical appearance of the animals (such as **is_large, has_claws, has_mane, has_four_legs** and **has_wings**); attributes describing various motor skills (such as **can_fly, walks_on_two_legs** and **can_speak**); attributes describing the animal's habitat (**is_wild, lives_in_circus**); as well as attributes describing more scientific classifications (**is_mammal**), and many others.

We wish to construct a monomial (assuming one exists) to distinguish lions from non-lions. For the attributes mentioned above, an appropriate conjunction may be

$$M = \textbf{is\_mammal} \textit{ and } \textbf{is\_large} \textit{ and } \textbf{has\_claws} \textit{ and } \textbf{has\_four\_legs}$$

In this example, the probability distribution D^+ is interpreted as reflecting the natural world regarding lions. For instance, each of the four attributes appearing in M may be true (i.e., assigned the value 1) with probability 1 in D^+; if so, this simply reflects the fact that, for example, *all* lions are mammals. (Since we are assuming here that lions can be represented usefully by monomials, it follows that some attributes must be true in D^+ with probability 1. The informal intuition that no event has probability 1 corresponds here to the fact that monomials are probably not rich enough to describe the real world). Other attributes are true in D^+ with smaller probabilities. We might expect the attribute **has_mane** to be true with probability approximately $\frac{1}{2}$, if there are equal numbers of male and female lions. Similarly, we expect the attribute **walks_on_two_legs** to be true with relatively low probability, and **has_wings** to be true with probability 0. Notice that there may be dependencies of arbitrary complexity between attributes in the distributions. The attribute **is_wild** may be true with very high probability in D^+ if most lions live in the wild, but the probability that *both* **is_wild** and **lives_in_circus** are true is 0. A slightly more subtle dependency might be that even though few lions can walk on two legs, almost all of those that live in the circus can walk on two legs. In an analogous manner, the negative distribution D^- is intended to reflect the examples of non-lions in the animal world, and again there are many dependencies. Animals with wings may comprise only a small fraction of those animals that are not lions, but the probability that an animal with wings can fly is very high. Note that for simplicity, we have chosen an example that is monotone — no attribute appears negated in the monomial M. A natural example of nonmonotonicity might be a monomial for female lions, where we would need to include the *negation* of the attribute **has_mane**.

Thus, in this domain, a learning algorithm must infer a monomial over the animal attributes that performs well as a classifier of lions and non-lions. Note that the meaning of "performs well" is intimately related to the distributions D^+ and D^-. In the distributions described above, it may be that the monomial M is the only good approximation of the concept, depending on the exact probabilities in the distributions, and the value of the error parameter ϵ. However, if we considered only animals for which the attribute **lives_in_circus** is true, the monomial consisting of the sole attribute **has_claws** might suffice to distinguish lions from the other animals in the circus. Note that these conjunctive formulae are not intended as Platonic descriptions of categories. The only requirement on the monomials is that they distinguish with sufficient accuracy categories in the real world as specified by D^+ and D^-.

We now describe an algorithm L for learning monomials over n attributes with arbitrary distributions D^+ and D^-. Although the monomial output by L has error less than ϵ on both distributions, L needs only examples drawn from D^+ in order to learn. In fact, one can argue that L is the only reasonable algorithm for learning monomials from positive examples alone.

The idea behind the algorithm is the following: suppose that the attribute x_i appears in the monomial M being learned. Then in a randomly drawn positive example, x_i is *always* assigned the value 1. Thus, if some attribute x_j is assigned the value 0 in a positive example, we are certain that x_j does not appear in M. The algorithm L is:

```
$H \leftarrow x_1\overline{x_1}x_2\overline{x_2}\cdots x_n\overline{x_n}$;
**for** $i := 1$ **to** $B$ **do**
**begin**
    $\vec{v} \leftarrow$ POS;
    **for** $j := 1$ **to** $n$ **do**
        **if** $\vec{v}(j) = 0$ **then**
            delete $x_j$ from $H$;
        **else**
            delete $\overline{x_j}$ from $H$;
**end**
output $H$.
```

where $\vec{v}(j)$ denotes the jth bit of $\vec{v}$.

How can algorithm L err? Only by failing to delete some attribute x_j that does not appear in M. An exact bound on the value of the loop counter B such that this happens with probability less than δ can be deduced to be $\frac{2n}{\epsilon}(\ln(2n) + \ln(\frac{1}{\delta}))$ by applying the techniques described in theorem 8. Here we use ln to denote $\log_e$.

In the case of our lions example, the attributes **can_speak**, **can_fly**, and **has_wings** will be deleted from the hypothesis immediately, since no lion can speak or has wings (i.e., every positive example assigns the value 0 to these attributes). With high probability, we would also expect **walks_on_two_legs**, **lives_in_circus**, and **has_mane** to be deleted, because each of these attributes is false with some reasonable probability in the positive examples. Depending on the exact value of ϵ and the precise probabilities in D^+, the attribute **is_wild** may also be deleted. However, the four attributes appearing in M will certainly not be deleted, if they are never assigned the value 0 in the positive examples.

In this example, the two sources of error can be exemplified as follows. First, it is possible that rare midget lions exist but have not occurred in the training set of examples. In other words, **is_large** should have been deleted from the hypothesis monomial, but has not been. This is not serious, since the learned monomial will only misclassify future examples that are infrequent in D^+. Second, it is possible that the training set contained a very unrepresentative set of lions, all of which can walk on two legs. In this case the learned monomial will include this attribute, and hence misclassify many future examples. While there is no ultimate defense against either of these two kinds of error, the learnability model allows the probabilities of their occurrence, ϵ and δ, to be controlled.

As mentioned above, algorithm L outputs a hypothesis that is accurate over both D^+ and D^-, but needs only positive examples to construct this hypothesis. In [15] it is shown that there is no algorithm for monomials requiring only negative examples. Such an impossibility result, however, does not preclude the usefulness of negative examples in learning monomials. In fact, it is shown in [13] that if a short monomial is to be learned, the number of examples required if both positive and negative examples are available is only logarithmic in n, the number of attributes. This is significant, since it shows that the problem of selecting the attributes relevant to a concept (from among a possibly large set of attributes) can be solved within the learnability model, and does not require the postulation of additional focusing mechanisms.

3 Some Tools

Substitution and Augmentation

We describe a substitution technique which allows us to obtain learning algorithms for new classes from existing learning algorithms for related classes. As an example, suppose we had a learning algorithm for monotone DNF formulae. We could use it to learn arbitrary DNF formulae as follows: For each variable x_i we create a new variable z_i. A positive (negative) example gets transformed to a new positive (negative) example over the expanded variable set so that z_i is equivalent to $\overline{x_i}$, and the new example is then fed to the learning algorithm for monotone DNF. Thus the example vector $\langle x_1, x_2, x_3, x_4 \rangle = \langle 1, 0, 0, 1 \rangle$ gets transformed to the new example $\langle x_1, x_2, x_3, x_4, z_1, z_2, z_3, z_4 \rangle = \langle 1, 0, 0, 1, 0, 1, 1, 0 \rangle$. The monotone DNF formula learned over the expanded variable set is easily transformed back into a (not necessarily monotone) DNF formula over the original variables by replacing, for each i, every occurrence of z_i with $\overline{x_i}$. The following two theorems allow much more general types of substitutions than the one just described.

Theorem 2 ([15]) *Let C be learnable, and let $\mathcal{G}$ be a finite collection of boolean formulae over k (constant) variables. Let C' be the class of all formulae that can be obtained by choosing an $f(x_1, \ldots, x_n) \in C$, and replacing one or more of the variables x_i with any formula $g_i(x_{i_1}, \ldots, x_{i_k})$, where $g_i \in \mathcal{G}$, and each x_{i_j} is one of the original n variables (thus, the resulting formula is still over n variables). Then C' is also learnable.*

Theorem 3 ([15]) *Let C be learnable. Let $p(n)$ be a fixed polynomial, and let the description of the $p(n)$-tuple $(B_1^n, \ldots, B_{p(n)}^n)$ be polynomial time computable from n, where each B_i^n is a boolean formula of n inputs. Let f be a formula in C over $n + p(n)$ variables. Then the class C' of formulae of the form*

$$f(x_1, \ldots, x_n, B_1^n(x_1, \ldots, x_n), \ldots, B_{p(n)}^n(x_1, \ldots, x_n))$$

is also learnable.

An important corollary of both theorems 2 and 3 is that the monotone learning problem is always as hard as the general learning problem (for classes that are closed under the required substitutions). For example, as described above,

Corollary 4 ([15]) *If monotone DNF is learnable, then DNF is learnable.*

Corollary 5 ([15]) *Let C be a class of formulae. Let μC be the class of those formulae in C in which each variable occurs at most once. Then if μC is learnable, then C is learnable. In particular, if μDNF is learnable, then DNF is learnable.*

The proof of corollary 5 follows from theorem 3 by creating new variables which act as identical copies of the original variable set; if there are enough copies then there is a μ-formula over the new variable set which is identical to the formula to be learned, except each variable occurs only once. The learned μ-formula is transformed back to a formula over the original variable set by replacing each copy of a variable with the original variable.

Corollary 5 is surprising, in that allowing only one occurrence of each variable is a strong restriction. The corollary states that the general learning problem is no harder than this restricted version. One of our results given later is that if A is the class of arbitrary formulae, then μA is not learnable (theorem 18).

Compression

Many of the restricted forms of the classes we consider have the property that any element of the class can be expressed with a number of bits polynomial in n, the number of variables. For example, for each number k, any kDNF formula of n variables can be written using at most $(2n)^k$ bits. A *polynomial time recognition algorithm* [6] for a class of representations A is an algorithm which, given any collection of examples, can find (in polynomial time) an element of A which is consistent with all examples in the given collection (assuming one exists).

Theorem 6 ([6]) *Let A be a class of representations such that every element of A can be expressed in $p(n)$ bits for some polynomial p. Then if there is a polynomial time recognition algorithm algorithm for A then A is learnable.*

As an example, theorem 6 may be used to show that kDNF is learnable. In general however, theorem 6 may not be applicable because either of the two conditions in the hypothesis may be unattainable: For DNF there is clearly a polynomial time recognition algorithm – simply form a disjunction of all of the positive examples in the given sample. However, not all DNF formulae can be expressed with a polynomial number of bits. On the other hand, every k-term-DNF formula can be expressed with a polynomial number of bits, but there is no polynomial recognition algorithm unless P = NP [17]. In either of these cases, the approach suggested in [6, 7] is that of an "Occam algorithm", which for the boolean domain is easier to describe in terms of data compression:

Theorem 7 ([7]) *Let $\alpha < 1$, and $c \geq 1$. If A is a class of representations, and there is a polynomial time algorithm L such that for any number m and any set of examples of total size m, L finds an element of A consistent with the examples and of size at most $x^c m^\alpha$, where x is the size of the smallest element of A consistent with the examples, then A is learnable.*

The idea is that for a sufficiently large (but still polynomial) set of randomly generated examples, any consistent hypothesis which is significantly smaller than the examples must in fact be consistent with most of the unseen examples as well. Thus for learning DNF, theorem 7 points out that if have an algorithm that can do slightly better than simply forming a disjunct of all positive examples seen, by instead finding *fewer* terms than the than the number of examples seen (where "fewer" means less than linear), then we essentially have a learning algorithm. Section 5 further discusses the problem of minimizing the number of terms in a DNF expression consistent with given data.

Learning in a Shallow Hypothesis Space

The following method allows us to prove that certain types of algorithms learn as required by definition 1 without having to resort to detailed probabilistic arguments for each new case. Theorem 8 follows from a combination of ideas from [5, 14, 18].

It is easy to show that to be $(1-\delta)$ confident that a given hypothesis is $(1-\epsilon)$ accurate, it suffices to test it against $\frac{1}{\epsilon}\ln\frac{1}{\delta}$ randomly generated examples. The problem is that in the course of learning, an algorithm may need to test many different hypotheses, and if each has a δ chance of appearing $(1-\epsilon)$ accurate, when in fact it is not, then by the time $\frac{1}{\delta}$ bad hypotheses are tested, one of them may look good. The way around this problem is to test each next hypothesis more times. Let

$$\mathit{ENOUGH}(i,\epsilon,\delta) = \frac{1}{\epsilon}(i\ln 2 + \ln(\frac{1}{\delta})).$$

If a hypothesis is tested against $\mathit{ENOUGH}(i,\epsilon,\delta)$ randomly generated examples, then it is easy to show that if it does not incorrectly classify any of the test examples, then the probability that it is not $(1-\epsilon)$ accurate is at most $\delta/2^i$.

Now consider modifying any incremental learning algorithm so that it tests its i^{th} hypothesis against $\mathit{ENOUGH}(i,\epsilon,\delta)$ examples, and outputs the hypothesis iff it correctly classifies all of the test examples, otherwise it picks a new hypothesis. One can show that if the algorithm *ever* outputs any hypothesis, then with probability at least $(1-\delta)$, the hypothesis is $(1-\epsilon)$ accurate. This method essentially tells us when it is safe to keep a particular hypothesis while guaranteeing that the conditions of learnability have been met. This technique was first used in [3, 5].

Following [14], define a learning algorithm to be

conservative iff it discards the current hypothesis if and only if the hypothesis incorrectly classifies the next datum.

acyclic iff the algorithm never conjectures any hypothesis more than once.

poly-failure-bounded iff the algorithm never makes more than a polynomial number of incorrect hypotheses.

Then we have the following

Theorem 8 *If L is a conservative, acyclic, poly-failure-bounded learning algorithm which never takes more than some fixed polynomial time between input examples, then L can be transformed into an algorithm which learns as required by the learnability model.*

The proof is simply the observation that by changing L so that it tests its i^{th} hypothesis $ENOUGH(i, \epsilon, \delta)$ times, and noting that L, being conservative, acyclic, and poly-failure-bounded, *must* find within a polynomial number of conjectures an hypothesis which passes all of its tests, in which case it is $(1 - \epsilon)$ accurate by the remarks above. From a different viewpoint, the hypothesis space being searched by L is a shallow (polynomial depth) directed acyclic graph (DAG). L walks through the DAG testing hypotheses as it goes, and either reaches the bottom, in which case it must be correct by the fact that it is conservative, or it got stuck somewhere along the path because an hypothesis tested well, in which case, by our probabilistic analysis, it is acceptable. If the depth of the search space (the polynomial failure bound) is known *a priori* to be some number d, then each hypothesis can be tested exactly $ENOUGH_d(\epsilon, \delta) = \frac{1}{\epsilon}(\ln d + \ln(\frac{1}{\delta}))$ times, and the same results hold. In this latter case the total number of examples needed for testing may be significantly less. As a sample application, note that a monomial learning algorithm such as the one given in section 2, where each next hypothesis is obtained by crossing off some literals from the previous hypothesis, satisfies the conditions of theorem 8, and therefore, augmented with the appropriate counters, it learns the class of monomials.

Boolean Closure

These results provide tools for determining learnability of new classes of formulae. Let A and B denote classes of representations of boolean concepts. (For example, k-term-DNF, monomials, etc.)

Theorem 9 ([15]) *If A is learnable, and if B is learnable from negative examples only, then $A \vee B = \{f_1 \vee f_2 : f_1 \in A, f_2 \in B\}$ is learnable.*

Theorem 10 ([15]) *If A and B are each learnable from positive examples only, then $A \wedge B = \{f_1 \wedge f_2 : f_1 \in A, f_2 \in B\}$ is learnable from positive examples only.*

There are natural duals for both theorems 9 and 10 (switch "$\wedge$" and "$\vee$" wherever they occur, as well as "positive" and "negative"), and together with theorem 16, we have the following two tables for the learnability of $A \wedge B$, and $A \vee B$, given the learnability of A and the learnability of B. In the tables, we label the rows and columns according to whether each of A and B is learnable from positive examples only, negative examples only, or from both positive and negative examples. We then label the corresponding learning problem ($A \vee B$ for the table of Figure 1, $A \wedge B$ for the table of Figure 2) by YES (if we can always learn in polynomial time) or NO (if the learning problem is NP-hard for some pairs (A, B)). Note that theorem 9 is optimal in that we cannot relax the constraints on B to allow B to be learnable from both positive and negative examples.

4 Positive Results

In this section, we survey some positive results — algorithms for learning nontrivial classes in the distribution-free sense required by the learnability model. We discuss theorems stating

$A \vee B$	A learnable from POS	A learnable from NEG	A learnable from POS and NEG
B learnable from POS	NO	YES	NO
B learnable from NEG	YES	YES	YES
B learnable from POS and NEG	NO	YES	NO

Figure 1: Learnability of $A \vee B$.

$A \wedge B$	A learnable from POS	A learnable from NEG	A learnable from POS and NEG
B learnable from POS	YES	YES	YES
B learnable from NEG	YES	NO	NO
B learnable from POS and NEG	YES	NO	NO

Figure 2: Learnability of $A \wedge B$.

the learnability of several classes of boolean formulae, as well as another representation of boolean concepts called Decision Lists that is quite similar to the **cond** predicate in LISP.

Since a monomial is simply a 1-CNF formula, the algorithm L in section 1 shows that 1-CNF is learnable from positive examples only. In fact, algorithm L can be generalized to an algorithm for learning kCNF, again using only positive examples. This generalized algorithm relies on the fact that for fixed k, there are only a polynomial number of disjunctive clauses with k or fewer literals.

Theorem 11 ([23]) *For each $k \geq 1$, kCNF is learnable from positive examples only.*

By duality, we also have that kDNF is learnable from negative examples alone. However, as in the case of monomials, it is shown in [13] that there is an algorithm using both types of examples that requires fewer examples than algorithms using only one type, if the formula being learned is small compared to the number of attributes.

Using the tools described in theorem 9, we have

Theorem 12 ([15]) *kDNF $\vee$ kCNF $= \{f_1 \vee f_2 : f_1 \in k\text{DNF}, f_2 \in k\text{CNF}\}$ is learnable.*

Similarly, by theorem 10, kDNF $\wedge$ kCNF is also learnable. Furthermore, since kDNF (respectively, kCNF) cannot be learned from positive (respectively, negative) examples alone, it follows that any learning algorithm for these classes requires both types of examples.

While a theorem in section 5 shows that learning k-term-DNF (by k-term-DNF) is probably intractable, theorem 13 states that there is an algorithm that learns k-term-DNF by employing a more powerful representation. This point is discussed in more detail in section 5.

Theorem 13 ([17]) *For all positive integers k, k-term-DNF is learnable by kCNF.*

As final examples of learnable classes, we have theorems 14 and 15. A k-Decision List is a list of pairs $C = ((m_1, b_1), \cdots, (m_j, b_j))$ where each m_i is a monomial of length at most k, and each b_i is 0 or 1. The value of C on a boolean vector $\vec{v}$ can be defined algorithmically: let i be the smallest integer such that $\vec{v}$ satisfies m_i. Then $C(\vec{v}) = b_i$ (or 0 if no such i exists). Decision Lists are powerful representations of boolean functions, since any kDNF (or kCNF) formula can be represented by a (polynomially) small k-Decision List [Rivest].

Theorem 14 ([19]) *The class of k-Decision Lists is learnable.*

Theorem 15 ([13]) *For each partition P, the class C_P of internal disjunctive formulae is learnable.*

The proofs of the theorems rely on techniques using theorems 6 and 7. In addition to the positive results for representations of boolean functions presented here, there have also been learning algorithms given for continuous representations, such as linear separators and geometric regions in Euclidean space [6].

5 Negative Results

This section summarizes negative results which indicate that certain classes cannot be learned in the distribution-free sense required by the learnability model. Recall that all negative results here *rely on the complexity-theoretic assumption that* RP $\neq$ NP. It should be noted that the proofs of these theorems indicate nonlearnability by constructing, for each concept class in question, a particular distribution for which no learning algorithm can learn, and thus no algorithm can learn for all distributions as required by the learnability model. Learnability for natural subclasses of distributions is not necessarily excluded.

While 1-term DNF (the class of monomials) is known to be learnable, we have:

Theorem 16 ([17]) *For all integers $k \geq 2$, (monotone) k-term-DNF is not learnable by k-term-DNF.*

Thus, even when the unknown formula is the sum of two monotone terms, it is NP-hard to find a two term (possibly non-monotone) expression for the examples seen. By contrasting this with theorem 13, we conclude that it is the restricted expressibility of k-term-DNF which prohibits learning, *i.e.* although patterns in the data may be observable, the demand that the learned formula be expressed in k-term-DNF is significant enough of a constraint to render the task intractable. A richer domain of representation, k-CNF, allows a greater latitude in expressing the formula learned. Thus the availability to the learner of a variety of knowledge representations is seen to be valuable for learning. Often a change of representation can make a difficult learning task easy.

In light of this result, we are prompted to ask, for each k, what is the minimal value $s(k)$ such that k-term-DNF is learnable by $s(k)$-term-DNF. In other words, how much expansion of the representation space is needed before we can obtain learnability. If we can show that for some polynomial s, k-term-DNF is learnable by $s(k)$-term-DNF, then it immediately follows that the class DNF is learnable. Theorem 7 asserts that we needn't even do *that* well, that the function s may rely also on the number of examples seen. The following corollary of theorem 16 stands as a first step in determining the minimal possible function s.

Corollary 17 ([17]) *For all integers $k \geq 4$, (monotone) k-term-DNF is not learnable by $(2k-5)$-term- DNF.*

Thus finding a good formula from examples is hard even if we allow the formula found to have roughly twice as many disjuncts as the formula to be learned.

Recall that the class of μ formulae is the class of boolean formulae (of any form) where each variable appears at most once. It is surprising that even for this restricted type of formula, we have the following result.

Theorem 18 ([17]) *μ-formulae are not learnable.*

We are also interested in *threshold* operators which are not usually concisely representable as boolean formulae. For example, a common decision rule might be: It's a lion iff it has at least 5 of the following 12 attributes: has_mane, is_wild, has_claws, is_large, ... While we might also allow weights to be given to each feature reflecting its relative importance, it turns out that intractability appears already in the $\{0,1\}$ case. More formally, a boolean threshold function $Th_k(Y)$ of n variables is a subset Y of variables, and a threshold k such that the set of positive examples are exactly those vectors with at least k of the variables of Y set to "1".

Theorem 19 ([17]) *Boolean threshold functions are not learnable.*

In cases where learning a class of representations is thought to be intractable, or when no learning algorithm exists, we may wonder whether we can learn *heuristic* rules for the concept; a rule which accounts for some significant fraction of the positive examples, while avoiding incorrectly classifying most of the negative examples. For example, since there is a learning algorithm for monomials, but not for k-term-DNF, and none known for general DNF, perhaps we can find a single monomial which covers half of the positive examples while avoiding error on all but $1-\epsilon$ of the negative examples whenever such a monomial exists. In [17], the notion of $h(n)$-heuristic learnability was formalized so that a class is $h(n)$-heuristically learnable iff a learning algorithm can produce a hypothesis which correctly classifies at least the fraction $h(n)$ of the positive examples, while correctly classifying $1-\epsilon$ of the negative examples.

Theorem 20 ([17]) *For any c, $0 < c < 1$, DNF is not c-heuristically learnable by monomials.*

Theorem 21 ([17]) *μ-formulae of n variables are not $e^{-n^{1/3}}$-heuristically learnable, if the formula produced must avoid misclassifying all negative examples.*

Theorem 20 shows that the heuristic of covering as large a fraction of the positive examples as possible with a single monomial while learning DNF is not feasible.

Theorem 21 shows that even if there is a μ-formula correctly classifying all of the positive and negative examples, it is NP-hard to find one which correctly classifies an exponentially vanishing fraction of the positive examples, if we require that it avoid misclassifying any negative example.

Finally, we point out the results of [11], indicating that, assuming the existence of certain types of cryptographic functions, there are families of easy to compute functions which must appear totally random in a very strong sense to any polynomial time learning algorithm.

6 Extensions and Restrictions

In addition to the basic learnability model, there are several realistic variations that have been proposed to model other natural learning scenarios. For instance, for those classes for which we (provably) cannot learn from examples only, or for which we cannot find a learning algorithm, a reasonable question to ask is the following: what information might allow these classes to be learned? To study this question, learning algorithms are allowed to access *oracles* (or *teachers*) that answer queries about the concept being learned (in addition to the sources of random examples, POS and NEG). Examples are an oracle that answers membership queries for the concept being learned (*e.g.,* is the boolean vector $\vec{v}$ a positive example of the concept?) and an oracle that answers whether a given boolean formula is logically equivalent to the formula being learned. An extensive study of various oracles for concept learning can be found in [5]; see also [3, 23]. In the context of algorithms that learn in polynomial time, there is the related issue whether the algorithms can be made to converge even faster by a carefully chosen set of examples [26].

The learnability model as presented assumes a perfect source of examples drawn from the unknown distributions, and in some practical situations, this assumption may be too strong. Thus, we are led to consider the case where there is some probability of error in the source. The two types of errors studied thus far are "random" errors and "malicious" errors. In the random error model, there is some probability that the source POS will draw from the distribution D^-, yet still report the example as being positive (similarly for the source NEG). In [4] it is shown that kDNF can be learned in this model even with rather high error rates. In the malicious model, where there is some probability of an error about which we can make no assumptions (e.g., it may be chosen by our adversary), [24] gives an algorithm for kDNF tolerating a lower rate of error.

For classes where learnability is either an unresolved problem or known to be intractable, it is reasonable to try to find learning algorithms that work for specific natural probability distributions on the examples. In [15] an algorithm is given for μDNF and k-term-μDNF when both D^+ and D^- are uniform. This contrasts with the fact that learning k-term-μDNF is NP-hard [15]. It is also shown that monotone DNF is learnable, when both distributions are uniform, in the more general model of group learning, which appears to be of independent interest. In group learning, the learning algorithm needs only to be able to identify large (polynomial) sets of examples as either all positive or all negative, rather than individual examples.

Clearly, an important line of research in the learnability model is the investigation of other types of representations for concepts. As a start in this direction, various representations of continuous concepts have been studied [6, 13], as well as representations of languages using finite automata [3]. Also, it has been observed that relations, in the sense of predicate calculus, can be added to the propositional case in certain very restricted ways while maintaining learnability [24]. The representation issue, more generally, has been addressed also in the context of "bias" [13, 22].

The results cited all serve to identify the limits of polynomial time learnability on universal computers. It is interesting also to ask where the limits lie if we restrict the model of computation, for example, to neural networks. In [24] some results are given about the learning capabilities of networks of threshold elements.

Acknowledgements

This research was supported by grants NSF-DCR-8600379 and ONR-N00014-85-K-0445. The third author was also supported by the Computer Science Department, University of Illinois at Urbana-Champaign, and the fourth author by a Guggenheim Fellowship. Author's addresses: Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138, and Computer Science Department, University of Illinois, Urbana, IL 61801.

References

[1] D. Angluin. Finding Patterns Common to a Set of Strings. *JCSS* 21, 46–62, 1980.

[2] D. Angluin. Inference of Reversible Languages. *JACM* 29(3), 741–765, 1982.

[3] D. Angluin. *Learning Regular Sets From Queries and Counter-examples.* Tech. Rept. TR-464, Yale U. Comp. Sci. Dept., 1986.

[4] D. Angluin and P.D. Laird. *Identifying k-CNF Formulas From Noisy Examples.* Tech. Rept. TR-478, Yale U. Comp. Sci. Dept., 1986.

[5] D. Angluin. *Types of Queries for Concept Learning.* Tech. Rept. TR-479, Yale U. Comp. Sci. Dept., 1986.

[6] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Classifying Learnable Geometric Concepts With the Vapnik-Chervonenkis Dimension. In *Proc. of the* 18th *Annual ACM STOC*, pp 273-282, 1986.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. *Occam's Razor.* Tech. Rept. UCSC-CRL-86-2, UC Santa Cruz, 1986.

[8] S. Crespi-Reghizzi. An Effective Model for Grammar Inference. In *Inform. Process. 71*, B. Gilchrist, Ed. North Holland, NY, 1972.

[9] Duda - Hart

[10] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman, San Francisco, 1979.

[11] O. Goldriech, S. Goldwasser, and S. Micali. How to Construct Random Functions. In *Proc. of the 25th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 1984.

[12] E. M. Gold. Language Identification in the Limit. *Inf. Contr.* 10, 447–474, 1967.

[13] D. Haussler. *Quantifying the Inductive Bias in Concept Learning.* Tech. Rept. UCSC-CRL-86-25, UC Santa Cruz, 1986.

[14] D. Haussler. *Space Efficient Learning Algorithms.* Unpublished manuscript, 1986.

[15] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the Learnability of Boolean Formulae. In *Proc. of the* 19^{th} *Annual ACM STOC*, 1987.

[16] Michalski

[17] L. Pitt and L.G. Valiant. *Computational Limitations on Learning From Examples.* Tech. Rept. TR-05-86, Harvard University, 1986, submitted for publication.

[18] L. Pitt. Unpublished course notes for Formal Models of Machine Inference, taught at University of Illinois.

[19] R. Rivest. Learning Decision-Lists. Unpublished manuscript, December, 1986.

[20] T. Shinohara. Polynomial Time Inference of Extended Pattern Languages. In *Proceedings, Software Sci. and Engineering*, Kyoto, Japan, 1982.

[21] T. Shinohara. Polynomial Time Inference of Pattern Languages and its Application. In *Proc. of the 7th IBM Symposium on Math. Foundations of Comp. Sci.*, 1982

[22] P. E. Utgoff and T. M. Mitchell. Acquisition of Appropriate Bias for Inductive Concept Learning. In *Proc. of AAAI-82*, Pittsburgh, PA, 1982.

[23] L. G. Valiant. A Theory of the Learnable. *Comm. ACM,* 27(11):1134-1142, 1984.

[24] L. G. Valiant. Learning Disjunctions of Conjunctions. In *Proceedings of the* 9^{th} *IJCAI,* vol. 1, pp 560-566, Los Angeles, CA. August, 1985.

[25] L. G. Valiant. Deductive Learning. *Phil. Trans. R. Soc. Lond.* A 312, 441-446, 1984.

[26] P. H. Winston. Learning Structural Descriptions from Examples. PhD thesis, in *The Psychology of Computer Vision*, McGraw-Hill, NY, 1975.