Machine Learning, 5, 165–196 (1990)

Special issue for COLT90

Learning Nested Differences of Intersection-Closed Concept Classes

DAVID HELMBOLD (DPH@SATURN.UCSC.EDU)
Board of Studies in Computer and Information Sciences, University of California at Santa Cruz, Santa Cruz, CA 95064

ROBERT SLOAN (RSLOAN@ENDOR.HARVARD.EDU)
Aiken Computation Lab., Harvard University, Cambridge, MA 02138

MANFRED K. WARMUTH (MANFRED@SATURN.UCSC.EDU)
Board of Studies in Computer and Information Sciences, University of California at Santa Cruz, Santa Cruz, CA 95064

Abstract. This paper introduces a new framework for constructing learning algorithms. Our methods involve master algorithms which use learning algorithms for intersection-closed concept classes as subroutines. For example, we give a master algorithm capable of learning any concept class whose members can be expressed as nested differences (for example, $c_1 - (c_2 - (c_3 - (c_4 - c_5)))$) of concepts from an intersection-closed class. We show that our algorithms are optimal or nearly optimal with respect to several different criteria. These criteria include: the number of examples needed to produce a good hypothesis with high confidence, the worst case total number of mistakes made, and the expected number of mistakes made in the first t trials.

Keywords. concept learning, mistake bounds, exception handling, pac learning, nested difference, intersection-closed.

1. Introduction

We are interested in efficient algorithms for learning concepts from examples. Formally, *concepts* are subsets of some *instance domain* X from which instances are drawn and a *concept class* is a subset of 2^X, the power set of X. The instances are labeled consistently with a fixed *target concept* t which is in the concept class C to be learned; that is, an instance is labeled "+" if it lies in the target concept and "−" otherwise. Labeled instances are called *examples*.

There has been a surge of interest in learning from examples sparked by the introduction of a model of learning by Valiant (1984). This model accounts for both the performance of the learning algorithm as well as the computational resources and the number of examples used. Even though some practical learning algorithms have been found (Valiant, 1984; Rivest, 1987; Shvaytser, 1988; Littlestone, 1988; Blumer, Ehrenfeucht, Haussler & Warmuth, 1989; Haussler, 1989), and learnability of concept classes has been characterized (Blumer et al., 1989) using the Vapnik-Chervonenkis (VC) dimension (Vapnik and Chervonenkis, 1971), no practical algorithms have been found for many natural classes, such as DNFs, DFAs, and general decision trees. Recently strong evidence has been found that classes such as boolean formulae and DFAs are actually not efficiently learnable (Pitt and Warmuth, 1990b; Kearns and Valiant, 1989; Pitt and Warmuth, 1990a).

In this paper we give various schemes for composing known efficient learning algorithms to create provably efficient learning algorithms for more complicated problems. Thus we give *constructive* results for learning new classes of concepts for which efficient learning algorithms were not previously known. The composition technique consists of new master algorithms that use the algorithms for the "simpler" classes as subroutines. The master algorithms learn nontrivially more complicated classes which can be defined in terms of the simpler classes. The master algorithms do not need to know the specific simpler classes, since they only pass information among the various algorithms for the simpler classes.

The simple classes considered here are usually intersection-closed concept classes and the master algorithms learn various compositions of intersection-closed concept classes. (A concept class is *intersection-closed* if for any finite set contained in some concept the intersection of all concepts containing the finite set is also a concept in the class.) There is a canonical algorithm for learning intersection-closed classes which we call here the *Closure* algorithm: The hypothesis of this algorithm is always the smallest concept containing all of the positive examples (POS) seen so far (Natarajan, 1987). We denote this concept as CLOS(POS). This paper presents new algorithms, which use the Closure algorithm as a subroutine, for learning compositions of intersection-closed concept classes.

The simplest composition scheme we consider learns the concept class DIFF(C), which consists of all concepts of the form $c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - c_p) \cdots))$, where all c_i are in C, and p is a positive integer called the *depth* of the concept. It is easy to see that an instance x is in the concept $c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - c_p) \cdots))$ if and only if the lowest indexed c_i that does not contain x has an even index (assume for convenience that $c_{p+1} = \emptyset$).

A more involved scheme efficiently learns the class DIFF($C_1 \cup C_2 \cup \cdots \cup C_s$); that is, each c_i may be in any C_j, for $1 \leq j \leq s$. This scheme assumes that each class C_j is interesection-closed and that their Closure algorithms can be implemented efficiently.

Examples of intersection-closed classes with efficient Closure algorithms include orthogonal rectangles in R^n, monomials (i.e., orthogonal sub-rectangles of the boolean hypercube), vector sub-spaces of R^n (Shvaytser, 1988), and so forth. In Figures 1 and 2 we give examples of DIFF(C), when C is the class of orthogonal rectangles in R^2. In this case DIFF(C) contains staircase type objects and some restricted unions of orthogonal rectangles. If C is the class of initial segments on the real line (orthogonal rectangles in dimension one with the same left endpoint), then DIFF(C) is the class of unions of intervals. For finite domains, it has been shown (Natarajan, 1987) that a concept class is learnable with one-sided error (i.e., the error with respect to the negative distribution is zero)[1] if and only if it is intersection closed and the VC dimension of the class grows polynomially in the relevant parameters.

If a concept class B is not intersection closed one can always embed it into a larger class C that is. However, the VC dimension of C may be much larger than the VC dimension of the original class B. The Closure algorithm learns an intersection-closed class C from positive examples only. Note that DIFF(C) is not necessarily intersection-closed, and our master algorithms for learning DIFF(C) will use both positive and negative examples.

One can generalize the composition scheme for DIFF(C) by allowing the innermost concept, c_p, to be in an *arbitrary* polynomially learnable class B (the learning algorithm for B may use both positive and negative examples). Let DIFF(C, B) be all concepts of the

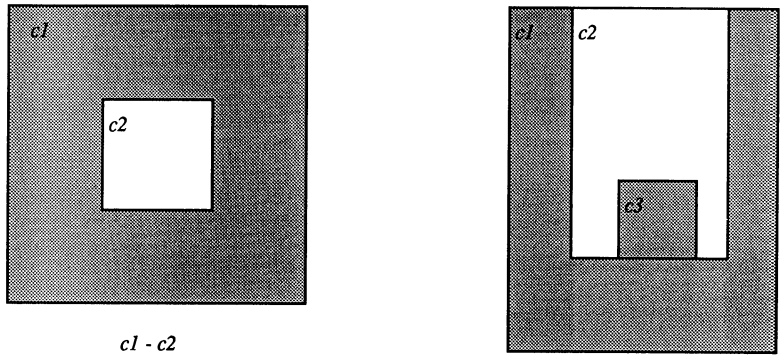

Figure 1. Two concepts in DIFF(Orthogonal Rectangles).

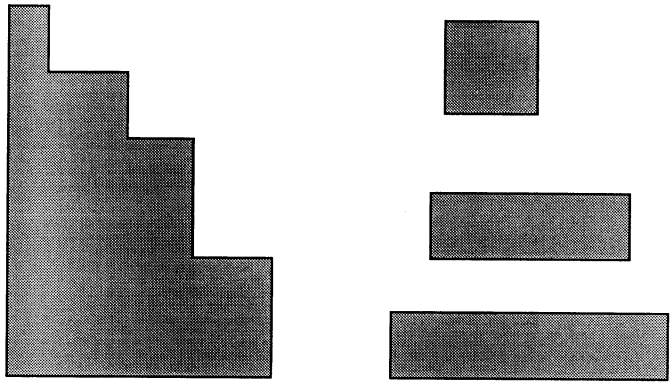

Figure 2. Another two concepts in DIFF(Orthogonal Rectangles).

form $c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - b) \cdots))$ where the c_j are in the intersection-closed class C and b is in B. The c_j are in some sense a filter formed with special concepts, while b is allowed to be more general.

Concepts in DIFF(C) and DIFF(C, B) of depth two were previously shown to be learnable in (Kearns, Li, Pitt and Valiant, 1987). In this paper, we consider the case of arbitrary depth. Observe the following closure properties: for two intersection-closed classes C_1 and C_2, the class $C_1 \wedge C_2 = \{c_1 \cap c_2 : c_1 \in C_1 \text{ and } c_2 \in C_2\}$ is intersection closed as well; the same holds for the class $C_1 \cap C_2 = \{c : c \in C_1 \text{ and } c \in C_2\}$, and dual results hold if intersection is replaced by union. Note also that C is intersection closed if and only if $\bar{C}$, which consists of the complements of the concepts of C, is "union closed."

In (Rivest, 1987) an algorithm is given for learning decision lists, which include nested differences of constant size monomials. In this paper we learn nested differences of intersection-closed classes by exploiting the combinatorial properties of intersection

closedness. Since the class of monomials (of arbitrary size) is intersection closed, this leads to a learning algorithm for the nested differences of monomials. Constant size monomials are not intersection closed; however, this class of monomials is "simple" in the sense that it contains few concepts. The decision list algorithm performs an exhaustive search, thus relying on the smallness of its simple class.

We have developed two types of master algorithms, one that remembers all examples seen (the Total Recall algorithm), and one that remembers a number of examples bounded by the VC dimension (the Space Efficient algorithm).

We recently discovered that Steven Salzberg (1988) has independently developed a space efficient algorithm similar to ours for DIFF(C) (where C consists of orthogonal rectangles in R^n) as a subroutine in his algorithm for predicting, among other things, breast cancer data. In some cases, his algorithms outperform the best previously known prediction algorithms. However, those results are only empirical. The crux of the type of research presented here is that using the methodology of computational learning theory (Valiant, 1984; Haussler, Littlestone & Warmuth, 1988) (which is rooted in the earlier works of Vapnik and others in the area of pattern recognition (Vapnik and Chervonenkis, 1971; Vapnik, 1982)), we can give efficient algorithms and *prove* their optimality.

In a companion paper (Helmbold, Sloan & Warmuth, 1989a) we present an interesting application of our methods. We give a time and space efficient implementation of the Closure algorithm for a nontrivial intersection-closed class: the class C of subsets of $\mathbf{Z}^k$ that are closed under addition and subtraction. In algebraic terms C consists of all submodules[2] of the free $\mathbf{Z}$-module of rank k. The space efficient Closure algorithm for submodules can be used as a subroutine in the space efficient master algorithm, leading to learning algorithms for nested differences of submodules.

2. The inclusion-exclusion algorithms

In this section we present the Total Recall and Space Efficient algorithms for learning DIFF(C), nested differences of an intersection-closed class C. The first algorithm assumes *total recall*; that is, sufficient space is available to store all of the examples. Then we show how the algorithm can be modified for space efficiency (so that only a few examples are memorized).

A *batch learning algorithm* takes as input a set of labeled training examples and produces, as output, a hypothesis. The examples are labeled according to some unknown *target concept* from the concept class to be learned. Intuitively, the hypothesis is supposed to approximate the hidden target concept. (Note that the hypothesis is not required to be in the concept class to be learned.) An *on-line learning algorithm* interactively participates in a series of *trials*. On each trial, the algorithm gets an unlabeled instance and predicts what its label is. After predicting, the on-line algorithm is informed of the instance's true label. A *mistake* is a trial where the on-line algorithm makes an incorrect prediction. The distinction between batch and on-line algorithms is blurred by the fact that a batch algorithm can be used in an on-line setting and vice versa. To use a batch algorithm in an on-line setting, give it all the examples previously seen and predict with the resulting hypothesis on the next trial. An on-line algorithm can be run in batch mode by feeding it all of the

training examples (ignoring its predictions) and using as its hypothesis the set of instances where, if seen on the next trial, the algorithm would predict "+".

We present the Total Recall algorithm as a batch algorithm and the Space Efficient algorithm as an on-line algorithm. From the above discussion it is easy to convert either algorithm to the other setting.

Before describing our two basic algorithms we give formal definitions of *closure* and *intersection closed.*

DEFINITION. For any concept class C and any subset S of the domain the *closure of S with respect to C*, denoted by CLOS(S), is the set $\cap \{c : c \in C \text{ and } S \subseteq c\}$. A concept class C is *intersection closed*[3] if C contains at least two concepts and whenever S is a finite subset of some concept in C then CLOS(S) is a concept of C.

Note that if C is intersection closed, then CLOS(∅) (i.e., the intersection of all concepts in C) is a member of C.

Description of the Total Recall algorithm: The algorithm first computes the closure of the positive examples. In general, this closure may contain some negative examples. These exceptions must be subtracted out of the hypothesis. The algorithm now focuses on those examples contained in the closure. By flipping their labels and computing the closure of the resulting positive examples (which were the original exceptions), the algorithm finds a suitable concept to subtract off. This concept may contain further exceptions—examples that were originally positive. However, by iterating this procedure, the Total Recall algorithm creates a nested difference consistent with the examples.

A detailed description of the Total Recall algorithm is given in Figure 3. For any intersection-closed class C it learns DIFF(C) assuming an efficient implementation of the Closure algorithm. The function POS takes a set of examples and returns those which are labeled positive. The function FLIP takes a set of examples and returns a new set of examples containing the same instances but with the labels flipped (i.e., "+" examples become "−" examples and vice versa).

The concept $h = h_1 - (h_2 - \cdots - (h_{p-1} - h_p) \cdots)$ is the hypothesis of the algorithm. If C contains the empty concept, then one possible hypothesis of depth one is $h = h_1 = \emptyset$. For syntactic purposes, we set h_{p+1} to ∅ (even if C does not contain the empty concept), ensuring that for any instance there is always some h_i which does not contain the instance.

When given a new instance x, the algorithm predicts its label according to h as follows (see Figure 4). Let l be the least index such that h_l does not contain x. If l is even then $h(x) = +$, and if l is odd then $h(x) = -$. In the on-line setting, the example is added to EX_1 even if the prediction was correct, and the Total Recall algorithm is executed to generate a new hypothesis.[4] This ensures that hypotheses produced by the Total Recall algorithm are consistent with all the examples that have been seen.

The Space Efficient algorithm differs from the Total Recall algorithm in that it keeps only a minimal number of instances for each h_i.

Algorithm Total Recall
Inputs: Examples of the target concept.
/* Computes hypothesis h and depth p. */
EX_1 := **all examples;** $i := 0$;
repeat
    increment i;
    $h_i := \mathrm{CLOS}(\mathrm{POS}(\mathrm{EX}_i))$;
    $\mathrm{EX}_{i+1} := \mathrm{FLIP}(h_i \cap \mathrm{EX}_i)$;
    /* "FLIP" flips the labels of the examples */
until $\mathrm{POS}(\mathrm{EX}_{i+1}) = \emptyset$;
$p := i$; $h_{p+1} := \emptyset$;
$h := h_1 - (h_2 - \cdots - (h_{p-1} - h_p)$;

Figure 3. Total Recall algorithm.

Algorithm Predict
Inputs: Hypothesis h and instance x.
/* Computes whether x is in set represented by h. */
$l := 0$;
repeat $l := l + 1$ **until** $x \notin h_l$;
If l **is even then output** $+$;
    else output $-$;

Figure 4. Prediction algorithm.

DEFINITION. Let S be a set of instances contained in some concept of C. A *spanning set* of S (with respect to some intersection-closed concept class C) is any $S' \subseteq S$ for which $\mathrm{CLOS}(S') = \mathrm{CLOS}(S)$.

Description of the Space Efficient algorithm: This algorithm (for a detailed description see Figure 5) represents each h_i by a minimal spanning set, S_i, and the hypothesis $h = h_1 - (h_2 - (h_{p-1} \cdots - h_p))$ by a sequence of minimal spanning sets, $S_1, \ldots, S_p$. Predicting is done as in the Total Recall algorithm. However, if there was a mistake made on an instance x and l is the least index such that $x \notin h_l$, then S_l is updated to a minimal spanning set of $S_l \cup \{x\}$ and thus h is modified by changing h_l to $\mathrm{CLOS}(S_l \cup \{x\})$. An illustration of the Space Efficient algorithm with domain R^2 and concept class DIFF(orthogonal rectangles) is given in Figures 6 and 7.

One would expect the Total Recall algorithm to perform well, since it produces a hypothesis in DIFF(C) of minimal depth that is consistent with the training examples. Because the

Algorithm Space Efficient
Inputs: Current hypothesis h, current depth p, and instance x.
/* On-line algorithm */
If first call then
initialize $p := 1$; $S_1 := \emptyset$; $h_1 := \text{CLOS}(\emptyset)$; $h_2 := \emptyset$;
Call Predict(h, x);
If mistake made then
$l := 0$;
repeat $l := l + 1$ until $x \notin h_l$;
$S_l :=$ minimal spanning set$(S_l \cup \{x\})$;
$h_l := \text{CLOS}(S_l)$;
if $l > p$ then $p := l$; $h_{p+1} := \emptyset$;

Figure 5. Space Efficient algorithm.

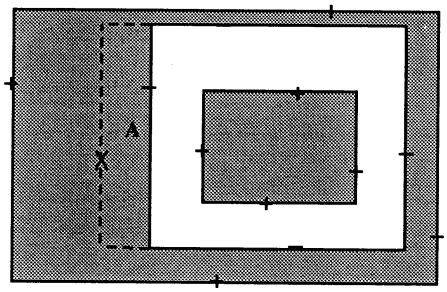

Figure 6. The minimal spanning sets stored by the Space Efficient algorithm and the corresponding hypothesis. The prediction for instance X is positive.

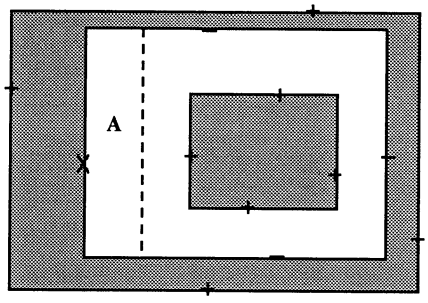

Figure 7. If a mistake was made on instance X, then the Space Efficient algorithm's new hypothesis is shown above. Note that the algorithm may have previously seen "+" points in region A. If it sees one of these points again, it will predict incorrectly.

hypothesis of the Space Efficient algorithm may not be consistent with the training examples (see Figures 6 and 7), the goodness of its performance is less clear. The remainder of the paper is devoted to proving performance bounds for these and related algorithms.

3. Performance criteria

We begin by enumerating several criteria by which we can judge how well a learning algorithm performs. In this enumeration each criteria is described informally. The definitions of the criteria are then formalized in the following subsections.

1. Bounds on the total number of mistakes made in any sequence of t trials, where an adversary chooses the examples and their order (Littlestone, 1988).
2. Bounds on the probability of making a mistake at trial t in a sequence of t trials, where the t instances are chosen by an adversary, but their order is picked at random from among the $t!$ possible permutations (Haussler et al. 1988).
3. Bounds on the probability of making a mistake at trial t in a sequence of t trials, where an adversary chooses a probability distribution over the instance domain, and the t instances are randomly drawn according to that distribution (Haussler et al. 1988).
4. Bounds on the expected total number of mistakes made in any sequence of t trials, where the examples are chosen by an adversary, but their order is picked at random from among the $t!$ possible permutations.
5. Bounds on the expected total number of mistakes made in a sequence of t trials where an adversary chooses a probability distribution over the instance domain, and the t instances are randomly drawn according to that distribution.
6. Bounds on the number of randomly drawn examples required for a batch algorithm to almost certainly produce a good hypothesis (Valiant, 1984).

The first five criteria, and indeed all mistake-based criteria, only make sense for on-line learning algorithms, while the last criteria is applicable only to batch algorithms.

We show in the next section that the Total Recall algorithm is optimal for criteria 2 through 6. Furthermore, if the Closure algorithm is optimal, then both the Total Recall and the Space Efficient algorithms are optimal under the first criteria. All of this paper's results are summarized in the conclusions.

3.1. Worst case mistake bounds

Our first performance criterion deals with the maximum number of mistakes made by the learning algorithm on any sequence of trials and was introduced by Littlestone (Littlestone, 1988).

For algorithm $\mathfrak{A}$ and target concept c, define $\mathbf{M}_{\mathfrak{A}}(c)$ to be the maximum number of mistakes made by algorithm $\mathfrak{A}$ on any possible sequence of trials where the instances are labeled according to c. For any concept class C, define the *worst case mistake bound of* $\mathfrak{A}$, $\mathbf{M}_{\mathfrak{A}}(C) = \sup_{c \in C} \mathbf{M}_{\mathfrak{A}}(c)$. If the context makes it clear which algorithm's performance is being bounded, a "." may be substituted for the subscript "$\mathfrak{A}$".