Machine Learning 2: 319–342, 1988

Queries and Concept Learning

DANA ANGLUIN (ANGLUIN@YALE.EDU)
Department of Computer Science, Yale University, P.O. Box 2158, New Haven, CT 06520, U.S.A.

(Received: July 20, 1987)
(Revised: January 26, 1988)

Keywords: Concept learning, supervised learning, queries

Abstract. We consider the problem of using queries to learn an unknown concept. Several types of queries are described and studied: membership, equivalence, subset, superset, disjointness, and exhaustiveness queries. Examples are given of efficient learning methods using various subsets of these queries for formal domains, including the regular languages, restricted classes of context-free languages, the pattern languages, and restricted types of propositional formulas. Some general lower bound techniques are given. Equivalence queries are compared with Valiant's criterion of probably approximately correct identification under random sampling.

1. Introduction

A successful learning component in an expert system will probably rely heavily on queries to its instructors. For example, Sammut and Banerji's (1986) system uses queries about specific examples as part of its strategy for efficiently learning a target concept. Shapiro's (1981, 1982, 1983) Algorithmic Debugging System uses a variety of types of queries to the user to pinpoint errors in Prolog programs. In this paper we use a formal framework to study the power of several types of queries for concept-learning tasks.

We consider the problem of identifying an unknown set L_* from some finite or countable hypothesis space $L_1, L_2, \ldots$ of subsets of a universal set U. The usual assumption is that one is given an arbitrarily or stochastically generated sequence of elements of U, each classified as to whether it is in L_*.[1] Instead, we will assume that the learning algorithm has access to a fixed set of *oracles* that will answer specific kinds of queries about the unknown concept L_*. The types of queries we consider are the following:

[1]We direct the reader to Angluin and Smith (1983) for a survey of inductive inference.

- *Membership.* The input is an element $x \in U$ and the output is *yes* if $x \in L_*$ and *no* if $x \notin L_*$.
- *Equivalence.* The input is a set L and the output is *yes* if $L = L_*$ and *no* if $L \neq L_*$. If the answer is *no*, an element $x \in L \oplus L_*$ is returned. ($L \oplus L_*$ is the symmetric difference of L and L_*, that is, the set of elements in one but not both of the sets.)
- *Subset.* The input is a set L and the output is *yes* if $L \subseteq L_*$ and *no* otherwise. If the answer is *no*, an element $x \in L - L_*$ is returned.
- *Superset.* The input is a set L and the output is *yes* if $L \supseteq L_*$ and *no* otherwise. If the answer is *no*, an element $x \in L_* - L$ is returned.
- *Disjointness.* The input is a set L and the output is *yes* if $L \cap L_*$ is empty and *no* otherwise. If the answer is *no*, an element $x \in L \cap L_*$ is returned.
- *Exhaustiveness.* The input is a set L and the output is *yes* if $L \cup L_*$ is U and *no* otherwise. If the answer is *no*, an element $x \notin L \cup L_*$ is returned.

For the queries other than membership, the returned element x is called a *counterexample.* The particular selection of a counterexample is assumed to be arbitrary, that is, a successful identification method must work no matter what counterexample is returned. We shall also consider *restricted* versions of each of these queries, for which the responses are just *yes* and *no*, with no counterexample provided.

Except in Section 2.2, we assume that if the input to a query is a set L, then it must be one of the elements L_i of the original hypothesis space. In particular, our lower bounds are strongly dependent on this requirement. The issue of how hypotheses are represented is very important, too important to be given any detailed treatment in this paper. We assume the hypotheses are simply represented as indices: the index i represents the hypothesis L_i.

1.1 An example: Poker hands

To illustrate the definitions above, we consider a concrete example from the game of poker. Let the target concept L_* be the set of all ordered pairs of poker hands such that the first hand beats the second hand. We choose a representation of a card that gives its value and suit, for example, QH or $2C$. A hand is an unordered set of five cards. A pair of hands is an ordered pair of hands with no cards in common. Then the universal set U is simply all pairs of hands, and the target concept is the subset consisting of those pairs in which the first hand beats the second. Thus,

$$\langle\{2S, 10H, QH, 2D, 5C\}, \{JS, 3H, 4H, JH, 2C\}\rangle$$

is an element of U that is not an element of L_*, since a pair of twos does not beat a pair of jacks.

We must also choose a description language to express subsets of U. This choice is crucial to the success of a learning method, but it is not the focus of this paper. We assume that the description language contains at least one expression for the target concept; the goal of learning is to find one such expression. We now give some examples of queries for this domain.

A membership query gives a specific pair of hands, for example,

$$\langle\{2S, 10H, QH, 2D, 5C\}, \{JS, 3H, 4H, JH, 2C\}\rangle.$$

The reply in this case is *no*, because the first hand does not beat the second.

An equivalence query specifies an expression in the description language, for example, "all pairs of hands in which the first hand contains two or three cards with equal value." This is not a correct description of the target concept, so the reply is *no* combined with a counterexample, i.e., a specific pair of hands that is a positive example of the target concept and a negative example of the queried expression, or vice versa. Thus the counterexample could be the pair of hands

$$\langle\{2S, 10H, QH, 2D, 5C\}, \{JS, 3H, 4H, JH, 2C\}\rangle.$$

This pair satisfies the description, but is not in the target concept L_*. Or the counterexample might be

$$\langle\{2S, 10H, QH, 3D, 5C\}, \{7S, 3H, 4H, JH, 2C\}\rangle,$$

which does not satisfy the description, but is nonetheless in the target concept.

1.2 Exact and probabilistic identification

The abstract examples we consider include sets of strings generated by grammars or recognized by automata and sets of truth-value assignments that satisfy certain classes of propositional formulas. We consider two criteria of successful identification, exact and probabilistic.

An identification method *exactly identifies* a set L_* with access to certain types of queries if it always halts and outputs an index i such that $L_i = L_*$. Note that this is not a limiting criterion of identification – the identification method is allowed one guess, and that guess must be exactly right.

In contrast, Valiant (1984) has introduced a criterion of probabilistic identification, and here we give a definition suitable for this paper. There is some probability distribution D on the universal set U. The probability of element x with respect to D is denoted $Pr(x)$. There is a *sampling oracle* $EX(\ )$, which has no input. Whenever $EX(\ )$ is called, it draws an element $x \in U$ according to the distribution D and returns the element x, together with an indication of whether or not x is in the target set L_*.

Successful identification is parameterized with respect to two positive quantities, the accuracy parameter ϵ and the confidence parameter δ. We define a notion of the difference between two sets L and L' with respect to the probability distribution as

$$d(L, L') = \sum_{x \in L \oplus L'} Pr(x).$$

The probability of getting an element in one but not the other of the two sets L and L' in one call to $EX(\ )$ is precisely $d(L, L')$.

An identification method is said to *probably approximately correctly identify* L_* if it always halts and outputs an index i such that

$$Pr(d(L_i, L_*) \leq \epsilon) \geq 1 - \delta.$$

That is, with high probability (quantified by δ) there is not too much difference (quantified by ϵ) between the conjectured set and the target set.

Probably approximately correct identification (abbreviated *pac*-identification) is the criterion of success if the sampling oracle $EX(\ )$ is available, otherwise we use exact identification. For more on *pac*-identification, see Blumer, Ehrenfeucht, Haussler, and Warmuth (1986, 1987) or Angluin and Laird (1987).

2. Equivalence queries

We now consider the different types of queries in more detail. This section concerns equivalence queries.

2.1 Exhaustive search

Given only equivalence queries, then one strategy is exhaustive search: enumerate the indices $i = 1, 2, \ldots$, querying each L_i until one gets an answer of *yes*, at which point one halts and outputs index i. This achieves exact identification.

If the queries are restricted to the hypothesis space, $L_1, L_2, \ldots$, then exhaustive search is sometimes the best that can be done. Suppose the hypothesis space is the set of singleton subsets of the set of all 2^n binary strings of length n. Then it is clear that a strategy using equivalence queries can only eliminate one hypothesis with each query, and so may use as many as $2^n - 1$ queries.

We can strengthen this result: the following adversary forces any method of exact identification using equivalence, membership, subset, and disjointness queries to make $2^n - 1$ queries in the worst case. (However, the counterexample returned by a single superset query will disclose the unknown set.)

The adversary maintains a set S of all the uneliminated binary strings of length n. Initially S contains all 2^n binary strings of length n. As long as S contains at least two distinct strings, the adversary proceeds as follows. If the next query is a membership query with the string x, then the adversary answers with *no*. If the next query is an equivalence or subset query with the singleton set $\{x\}$, then the adversary answers *no* along with the counterexample x. If the next query is a disjointness query with the singleton set $\{x\}$, then the adversary answers with *yes*. In each case, if x is a member of S, the adversary removes it from S.

It is not difficult to see that the responses of the adversary are compatible with the unknown hypothesis being $\{x\}$ for any element x still in S, and at most one element is removed from S with each query. Thus to be correct the algorithm must make at least $2^n - 1$ queries.

2.2 A logarithmic strategy: Majority vote

If the input L to an equivalence query is *not* required to be a member of the hypothesis space, a strategy based on "majority vote" can achieve exponentially fewer queries than exhaustive search. In this case we assume that the inputs to equivalence queries are arbitrary subsets of the universal set U, and the hypothesis output can also be an arbitrary subset of U. Suppose the hypothesis space is finite: $L_1, \ldots, L_N$. Then instead of making $N - 1$ queries, we may make $\lfloor \log N \rfloor$ queries, as follows. (Note: we use *log* to denote the base 2 logarithm, and *ln* to denote the natural logarithm.)

A set S is maintained of all the indices of hypotheses compatible with all the counterexamples seen so far. S initially contains all N indices. Iterate the following process until it halts.

If S contains just one element, i, then halt and output L_i. Otherwise define M_S to be the set of all elements x such that for more than $|S|/2$ elements i of S, $x \in L_i$. Make an equivalence query with M_S as input. If the answer is *yes*, halt and output M_S. Otherwise, there is a counterexample x. If $x \in M_S$, remove from S all i such that $x \in L_i$; otherwise, remove from S all i such that $x \notin L_i$.

A query is made only if $|S| > 1$, and every query answered *no* must reduce the cardinality of C by at least one half. Thus, a total of $\lfloor \log N \rfloor$ queries suffices. This result is a special case of the results of Barzdin and Freivald (1972).

A direct implementation of the majority-vote strategy, keeping an explicit list of all uneliminated hypotheses, is infeasible for most domains. In Section 2.5 we show that a simple algorithm indirectly implements the majority-vote strategy for k-CNF formulas. Approximating the majority vote leads to questions of how to sample "fairly" from the space of uneliminated hypotheses. Littlestone (1987) shows that the majority-vote method may not be query-optimal in some domains.

2.3 Some general lower bound techniques

Since in this paper we restrict the inputs to the queries to be elements of the hypothesis space, we now develop some simple lower bound techniques that generalize the lower bound proof in Section 2.1.

Lemma 1 *Suppose the hypothesis space contains a class of distinct sets $L_1, \ldots, L_N$, and a set $L_\cap$ such that for any pair of distinct indices i and j,*

$$L_i \cap L_j = L_\cap.$$

Then any algorithm that exactly identifies each of the hypotheses L_i using restricted equivalence, membership, and subset queries must make at least $N-1$ queries in the worst case.

PROOF: We describe an adversary. Initially S contains $1, 2, \ldots, N$. If $|S| > 1$ the adversary proceeds as follows. For a restricted equivalence query with the hypothesis L, the reply is *no*, and the (at most one) i such that $L_i = L$ is removed from S. For a membership query with the element x, if $x \in L_\cap$ then the reply is *yes*. Otherwise, the reply is *no*, and the (at most one) element i of S such that $x \in L_i$ is removed from S. For a subset query with the hypothesis L, if L is a subset of $L_\cap$ then the reply is *yes*. Otherwise, the reply is *no* and any element $x \in L - L_\cap$ is selected as the counterexample. The (at most one) element i of S such that $x \in L_i$ is removed from S.

At any point, for each $i \in S$, L_i is compatible with the replies to the queries so far. A correct exact identification algorithm must reduce the cardinality of S to at most one. Each query removes at most one element from the set S, so $N-1$ queries are required in the worst case, which proves Lemma 1. ∎

If the "intersection set" $L_\cap$ is not itself an element of the hypothesis space, then Lemma 1 can be strengthened as follows.

Lemma 2 *Suppose the hypothesis space contains a class of distinct sets $L_1, \ldots, L_N$, and there exists a set $L_\cap$ which is* not *a hypothesis, such that for any pair of distinct indices i and j,*

$$L_i \cap L_j = L_\cap.$$

Then any algorithm that exactly identifies each of the hypotheses L_i using equivalence, membership, and subset queries must make at least $N - 1$ queries in the worst case.

PROOF: The proof of Lemma 1 may be modified as follows. The replies to queries are the same, except that a counterexample must be provided when an equivalence query is answered *no.* Let L be the input to an equivalence query. Since $L_\cap$ is not in the hypothesis space, $L \neq L_\cap$. The counterexample is any element x of $L \oplus L_\cap$. If x is from $L_\cap$, there is nothing further to do, but if x is from $L - L_\cap$, the (at most one) element i in S such that $x \in L_i$ is removed from S. The rest of the proof is unchanged, proving Lemma 2. ■

There are dual results for equivalence, membership, and superset queries, which we state without proof.

Lemma 3 *Suppose the hypothesis space contains a class of distinct sets $L_1, \ldots, L_N$, and a set $L_\cup$ such that for any pair of distinct indices i and j,*

$$L_i \cup L_j = L_\cup.$$

Then any algorithm that exactly identifies each of the hypotheses L_i using restricted equivalence, membership, and superset queries must make at least $N - 1$ queries in the worst case.

Lemma 4 *Suppose the hypothesis space contains a class of distinct sets $L_1, \ldots, L_N$, and there exists a set $L_\cup$ which is* not *a hypothesis, such that for any pair of distinct indices i and j,*

$$L_i \cup L_j = L_\cup.$$

Then any algorithm that exactly identifies each of the hypotheses L_i using equivalence, membership, and superset queries must make at least $N - 1$ queries in the worst case.

2.4 Equivalence queries and stochastic equivalence

If the source of information about the unknown concept L_* is a domain expert, then it may be unreasonable to expect correct answers to equivalence queries, but the stochastic equivalence testing described below may be a practical substitute. Stochastic testing is a feature of Knobe and Knobe's (1976) method for identifying context-free grammars.

An identification method that uses equivalence queries and achieves exact identification may be modified to achieve *pac*-identification using calls to $EX(\ )$ instead of equivalence queries. The idea is instead of asking an equivalence query about L, the identification method calls $EX(\ )$ a number of times and checks to see that L is compatible with the element x returned and classified by $EX(\ )$. If not, then the identification method proceeds as though the equivalence query had returned the answer *no* with x as a counterexample. If L is compatible with all the samples drawn, then the identification method proceeds as though the equivalence query had returned the answer *yes*.

Suppose that when an equivalence query returns *yes*, the identification method simply halts and outputs the index j such that $L_j = L_*$. If the identification method makes

$$q_i = \lceil \frac{1}{\epsilon}(\ln \frac{1}{\delta} + i \ln 2) \rceil$$

calls to the $EX(\ )$ oracle in place of the i^{th} equivalence query, then the probability that the identification method will output an index j such that $d(L_j, L_*) \geq \epsilon$ is at most $(1-\epsilon)^{q_i}$. Thus, the probability that at any stage the identification method will output a hypothesis that is not an ϵ-approximation of L_* is at most

$$\begin{aligned} \sum_{i=1}^{\infty}(1-\epsilon)^{q_i} &\leq \sum_{i=1}^{\infty} e^{-\epsilon q_i} \\ &\leq \sum_{i=1}^{\infty} \frac{\delta}{2^i} \\ &\leq \delta. \end{aligned}$$

Thus the modified method achieves *pac*-identification of L_*.

What about the converse? Can the $EX(\ )$ oracle and *pac*-identification be replaced by equivalence queries and exact identification? Not if efficiency must be preserved. Consider the hypothesis space of all singleton subsets of the set of binary strings of length n. An identification algorithm using only equivalence queries must make $2^n - 1$ queries in the worst case.

However, for *pac*-identification in this domain, it suffices to make

$$q = \lceil \frac{2}{\epsilon}(\ln \frac{1}{\delta} + n \ln 2) \rceil$$

calls to $EX(\ )$. (This quantity grows linearly in n for fixed ϵ and δ.) If at least one of these calls returns a positive example x, output the set $\{x\}$. If all the calls return negative examples, output any set $\{y\}$ such that y has not appeared among the negative examples.

Let $D(z)$ denote the probability of string z with respect to the unknown distribution D. The probability that after q calls to $EX(\ )$ there exists a string z with $D(z) > \epsilon/2$ that has not been drawn is easily shown to be at most δ.

If all the strings z such that $D(z) > \epsilon/2$ have been drawn, then either we output the correct set $\{x\}$, or x was not drawn and and we output some set $\{y\}$ such that y was not drawn, in which case,

$$d(\{y\}, \{x\}) \leq D(y) + D(x) \leq \epsilon/2 + \epsilon/2 \leq \epsilon.$$

Hence this method achieves *pac*-identification.

Littlestone (1987) shows that in certain circumstances equivalence queries and errors of prediction are very closely related. His proof gives methods of converting identification algorithms that use equivalence queries into prediction algorithms and vice versa. Combined with the transformation above, Littlestone's result gives a method of converting prediction algorithms into algorithms for *pac*-identification.

2.5 Equivalence queries: k-CNF and k-DNF formulas

We now show that k-CNF and k-DNF formulas can be efficiently identified using only equivalence queries.

Let k-CNF denote the set of propositional formulas in conjunctive normal form over the n variables $x_1, \ldots, x_n$ with at most k literals per clause. If X is a formula and a is an assignment of truth-values to the n variables, $a(X)$ denotes the truth-value assigned to X.

We assume that there is an unknown k-CNF formula ϕ_* and that the $EX(\ )$ oracle returns pairs $\langle a, s \rangle$ where a is a truth-value assignment and $s = +$ if $a(\phi_*)$ is true and $s = -$ otherwise.

Upper bounds.

Valiant (1984) gives a method that runs in time polynomial in n^k, $1/\epsilon$, and $\ln(1/\delta)$ that achieves *pac*-identification of k-CNF formulas. A simple modification of this algorithm uses equivalence queries, achieves exact identification of the k-CNF formulas, and runs in time polynomial in n^k.

Initially let ϕ be the conjunction of all clauses C over n variables with at most k literals per clause. (There are no more than $(2n+1)^k$ such clauses.) At every point the set of assignments satisfying ϕ will be a subset of the set of assignments satisfying ϕ_*. Iterate the following process until the equivalence query returns *yes*, at which point halt and output ϕ.

Test ϕ for equivalence with ϕ_*. If it is not equivalent, let a be the counterexample. Then $a(\phi_*)$ is true and $a(\phi)$ is false. Remove from ϕ all clauses C such that $a(C)$ is false.

At least one clause is removed from ϕ for each negative answer to an equivalence query. By the time one removes all clauses from ϕ that are not implied by ϕ_*, ϕ is equivalent to ϕ_*, so the claim follows. The number of equivalence queries is bounded by $(2n+1)^k$.

The algorithm given above implements the majority vote method described in Section 2.2 for the class of k-CNF formulas. The set S of hypotheses consistent with the examples seen so far consists of every formula that is a conjunction of some subset of the uneliminated clauses. If an assignment assigns true to all the uneliminated clauses, then it assigns true to every hypothesis in S, so the majority vote is true. If an assignment assigns false to some uneliminated clause c, then for every hypothesis in S that is assigned true, there is another hypothesis in S (obtained by conjoining c) that is assigned false, so the majority vote is false. Hence the conjunction of all the uneliminated clauses gives the majority vote value for all assignments.

There is a logically dual method for k-DNF formulas, that is, formulas in disjunctive normal form over n variables with at most k literals per term. Haussler (1986, in press) and Littlestone (1987) have described other methods of identifying k-CNF and k-DNF formulas that may use many fewer queries.

Lower bounds.

Lemma 1 may be applied to the class of 1-CNF formulas. Consider the class of all formulas of the form

$$P_1 \cdot P_2 \cdots P_n,$$

where each P_i is either x_i or $\neg x_i$. There are 2^n such formulas, each one a 1-CNF formula satisfied by exactly one assignment, and no two formulas are satisfied by the same assignment. Thus, the hypothesis space of 1-CNF formulas satisfies the conditions of Lemma 1, which implies that any algorithm that exactly identifies every 1-CNF formula over n variables using restricted equivalence, membership, and subset queries must make $2^n - 1$ queries in the worst case.

Dually, we may consider the class of 1-DNF formulas of the form

$$P_1 + P_2 + \ldots + P_n,$$

where each P_i is either x_i or $\neg x_i$, to see that the hypothesis space of 1-DNF formulas satisfies the conditions of Lemma 3. Thus any algorithm that exactly identifies every 1-DNF formula using restricted equivalence, membership, and superset queries must make $2^n - 1$ queries in the worst case.

3. Membership queries

A membership query returns one bit of information: whether or not the queried element is a member of the unknown set L_*. The learning systems of both Shapiro (1981, 1982, 1983) and Sammut and Banerji (1986) use membership queries.

If the source of information is a domain expert, it seems reasonable to ask the expert to classify cases generated by the learning systems. However, in a practical case, say X-rays of potential tumors, it may be difficult for the system to generate fully instantiated cases (simulated X-rays) that embody the particular features the system has decided are relevant. In such a case, subset, superset, or disjointness queries using a higher-level description language may actually prove more reasonable.

In this section, we give some examples of learning algorithms using membership and equivalence queries, a combination termed a *minimally adequate teacher* by Angluin (1987d).

3.1 Monotone DNF formulas

We consider the class of monotone DNF formulas, that is, disjunctive normal form formulas over n variables that contain no negative literals. The main result to be proved in this section is the following.

Theorem 1 *There is an algorithm that exactly identifies every monotone DNF formula ϕ_* over n variables that uses equivalence and membership queries and runs in time polynomial in n and the number of terms of ϕ_*.*

PROOF: This theorem is proved by modifying an algorithm given by Valiant (1984) that *pac*-identifies any DNF formula ϕ_* over n variables using sampling and restricted subset queries and runs in time polynomial in n and the number of prime implicants of ϕ_*.

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

ϕ. The number of prime implicants of a general DNF formula may be exponentially larger than the number of terms of the formula. However, for a monotone DNF formula, the number of prime implicants is bounded by the number of terms of the formula. Thus to prove Theorem 1 it suffices to replace sampling by equivalence queries and restricted subset queries by membership queries in Valiant's algorithm.

The algorithm keeps a current hypothesis ϕ, initially the empty formula (equivalent to false). The hypothesis ϕ always consists of a sum of prime implicants of ϕ_* and therefore implies ϕ_*.

The algorithm tests ϕ using an equivalence query. If the reply is *yes*, the algorithm outputs ϕ and halts. Otherwise, the counterexample is an assignment a that satisfies ϕ_* but not ϕ.

From a the algorithm searches for a new prime implicant of ϕ_*. Let t be the product of all those variables x_i such that $a(x_i)$ is true. Clearly $a(t)$ is true. The following procedure is used to reduce t to a prime implicant.

For each t' obtained by deleting one literal from t, determine whether t' implies ϕ_* as follows. Let a' be the assignment that assigns true to those variables in t' and false to the others. Make a membership query with a'. It is not difficult to see that $a'(\phi_*)$ is true if and only if t' implies ϕ_*.

If t' implies ϕ_*, then t is replaced by t' and the reduction process is continued. Eventually the algorithm arrives at a term t such that t implies ϕ_*, but no term obtained from t by deleting one literal implies ϕ_*. In other words, t is a prime implicant of ϕ_*.

Note that the counterexample a still satisfies t, so t is not already in ϕ. The algorithm now replaces the hypothesis ϕ by $\phi + t$ and iterates from the equivalence test.

Let m denote the number of terms in the unknown formula ϕ_*. Each prime implicant added to the formula requires one equivalence query and at most n membership queries, so the running time of the algorithm is clearly bounded by a polynomial in m and n. The total number of equivalence queries is bounded by $m + 1$ and the total number of membership queries is bounded by mn. This concludes the proof of Theorem 1. ■

The counterexamples provided by the equivalence queries are essential to the efficiency of the above algorithm. (Recall that a restricted equivalence query returns only *yes* or *no* with no counterexample.)

Theorem 2 *For each positive integer n 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, membership, and subset queries must make at least $2^n - 1$ queries in the worst case.*

PROOF: Let $n > 0$ be given. Let

$$\phi_\cap = x_1y_1 + x_2y_2 + \ldots + x_ny_n.$$

The class $\mathcal{D}$ consists of the 2^n formulas of the form

$$T + \phi_\cap,$$

where T is any product

$$T = P_1 \cdot P_2 \cdots P_n,$$

where each P_i is either x_i or y_i.

Note that for any formula $\phi = T + \phi_\cap$ from $\mathcal{D}$ there is just one assignment that satisfies ϕ and does not satisfy $\phi_\cap$, namely the assignment that assigns true to exactly one member of each pair $\{x_i, y_i\}$ and makes all the variables in the term T true. Thus, for any pair of distinct formulas ϕ_1 and ϕ_2 from $\mathcal{D}$, the assignments that satisfy both formulas are exactly those that satisfy $\phi_\cap$.

Thus the class $\mathcal{D}$ satisfies the hypotheses of Lemma 1, so any algorithm that exactly identifies every formula in $\mathcal{D}$ using restricted equivalence, membership, and subset queries must make at least $2^n - 1$ queries in the worst case, proving Theorem 2. ■

Raymond Board (personal communication) has noted that Theorem 2 can be strengthened to include disjointness and exhaustiveness queries. Any two monotone DNF formulas are both satisfied by the "all true" assignment, and both falsified by the "all false" assignment. The adversary can answer every disjointness query with the "all true" assignment and every exhaustiveness query with the "all false" assignment without revealing any information about ϕ_*.

3.2 Other methods using membership and equivalence queries

We briefly describe some other domains in which exact identification can be done efficiently using membership and equivalence queries.

The regular sets. Angluin (1987d) has described an algorithm for identifying regular sets using equivalence and membership queries. If L_* is an unknown regular set whose canonical minimum deterministic acceptor (dfa) has n states, then the algorithm runs in time polynomial in n and m, where m is the maximum length of any counterexample returned by an equivalence query.

There are nice generalizations of this result for the classes recognized by deterministic one counter automata (Berman & Roos, 1987), deterministic bottom-up tree automata (Sakakibara, 1987a), and deterministic skeletal automata (Sakakibara, 1987b).

The set consisting of one string of length n is recognized by a dfa with $n + 2$ states. The hypothesis space of dfa's with $n + 2$ states satisfies the conditions of Lemma 1, so there is a lower bound of $2^n - 1$ on the number of queries needed by any exact identification algorithm for this domain using restricted equivalence, membership, and subset queries.

Context-free grammars with full non-terminal information. The domain described by Angluin (1987a) is a special case of identifying context-free languages modeled on Shapiro's (1981, 1982, 1983) approach to diagnosing errors in Prolog programs.

Let k be a positive integer and G_* be a context-free grammar with terminal alphabet T, nonterminal alphabet V, start symbol S, and productions P. Each production in P has at most k nonterminal symbols (and any number of terminal symbols) on the right hand side. Such a grammar is called k-bounded.

Only P is unknown, that is, k, T, V, and S are all known to the identification algorithm. Equivalence queries take as input a grammar G and return either *yes* or *no* combined with a counterexample x, i.e., a string in $L(G) \oplus L(G_*)$. *Nonterminal membership queries* take as input a string x and a nonterminal symbol A and return *yes* or *no* according to whether the string x can be generated from the nonterminal A using the grammar G_*.

Angluin (1987a) gives an algorithm that uses equivalence and nonterminal membership queries, runs in time polynomial in the size of G_* and the length of the longest counterexample, and exactly identifies any k-bounded context-free language.

k-term DNF and k-clause CNF formulas. The set of k-term DNF formulas is the set of formulas in disjunctive normal form over n variables with at most k terms. Dually, the set of k-clause CNF formulas is the set of formulas in conjunctive normal form over n variables with at most k clauses.

Angluin (1987b) shows that there is an algorithm that uses equivalence and membership queries, runs in time polynomial in n^k, and exactly identifies any k-term DNF formula. A dual result holds for k-clause CNF formulas.

A simple modification of a proof due to Pitt and Valiant[2] (1986) shows that for each k greater than 1, the class of k-term DNF or k-clause CNF formulas cannot be exactly identified by any algorithm that uses just equiv-

[2]See also Kearns, Li, Pitt, and Valiant (1987).

alence queries and runs in time polynomial in n^k unless P = NP.[3] Thus, membership queries seem to be essential to the efficient exact identification of k-term DNF formulas.

These negative results depend on the restriction that the hypothesis come from the original hypothesis space, that is, be expressed in DNF with at most k terms. If that restriction is dropped, k-term DNF formulas can be learned by using the larger hypothesis space of k-CNF formulas, since every k-term DNF formula is equivalent to a k-CNF formula.

Every 1-term DNF formula is also a 1-CNF formula, and vice versa. Thus the lower bound for 1-CNF formulas proved in Section 2.5 applies to the 1-term DNF formulas. The same holds for 1-clause CNF formulas and 1-DNF formulas.

4. Subset and superset queries

Membership queries are reducible to restricted subset queries if the space of hypotheses includes all the singleton subsets of the universal set U.

Valiant (1984) postulates a *necessity-oracle* for the problem of identifying DNF formulas. Given a term t, the necessity-oracle tests whether t implies the unknown formula ϕ_*, i.e., whether the set of assignments satisfying t is a subset of the set of assignments satisfying ϕ_*. The question of whether a DNF formula implies ϕ_* can be reduced to whether each of its terms implies ϕ_*, so in the terminology of this paper, the necessity-oracle is polynomially equivalent to restricted subset queries in the domain of DNF formulas.

For the problem of identifying CNF formulas, the dual question, whether or not the unknown formula ϕ_* implies a given clause c, is polynomially equivalent to a restricted superset query.

4.1 The pattern languages

Angluin (1980) introduced the class of pattern languages. Let A be a finite alphabet of constant symbols and let X be a countably infinite alphabet of variable symbols. We assume that A contains at least two distinct symbols. A *pattern* is a nonempty finite string of symbols from $A \cup X$. The *language of a pattern* p, denoted $L(p)$, is the set of all strings over the alphabet A obtained by substituting non-empty strings of constant symbols for the variable symbols of p. For example, if $p = 122x5yyx3$, then the language of p includes the strings 12205111103 and 122001512120013, but not the strings 12253 or 1221560601113.

[3]P is the class of sets recognizable in deterministic polynomial time and NP is the class of sets recognizable in nondeterministic polynomial time. P is contained in NP, but it is unknown whether the containment is proper.

The following result shows that superset queries alone suffice for efficient identification of the pattern languages; equivalence queries are not required for correct termination.

Theorem 3 *There is an algorithm that exactly identifies the class of languages defined by patterns of length n that uses restricted superset queries and runs in time polynomial in n.*

PROOF: We assume that there is an unknown pattern p_*. The goal is to find a pattern equivalent to p_* by asking queries of the form "$L(p) \supseteq L(p_*)$?" for any pattern p. The replies will be either *yes* or *no*, with no counterexample supplied.

Note that if p is a pattern of length n, then $L(p)$ contains at least one string of length n and contains only strings of length n or greater. Also, $L(x_1 x_2 \cdots x_n)$ is precisely all those strings of symbols from A of length n or greater. Thus we can determine the length of the unknown pattern p_* by using superset queries on the patterns x_1, $x_1 x_2$, $x_1 x_2 x_3$, and so on, to find the least $k+1$ such that $L(x_1 x_2 \cdots x_{k+1})$ is not a superset of $L(p_*)$. Then the length of p_* is k.

Having determined that the length of p_* is k, we can determine the positions and values of its constant symbols as follows. For each $a \in A$ and $i = 1, 2, \ldots, k$, query whether

$$L(x_1 \cdots x_{i-1} a x_{i+1} \cdots x_k) \supseteq L(p_*).$$

If so, then the i^{th} symbol of p_* is the constant symbol a. If for no a is this query answered *yes*, then the i^{th} symbol of $L(p_*)$ is a variable symbol.

Knowing the length of p_* and the positions and values of its constant symbols, it remains only to determine for each pair of positions containing variables whether the variables are the same or not. For each pair $i < j$ of positions of variable symbols in p_*, we query whether $L(p_{i,j})$ is a superset of $L(p_*)$, where $p_{i,j}$ is obtained from $x_1 x_2 \cdots x_k$ by substituting the new variable x for both x_i and x_j. If the answer is *yes*, then positions i and j of p_* contain the same variable; otherwise, they contain different variables.

Once all these tests have been completed, a pattern p equivalent to p_* can be constructed and output. The computation time for this method is clearly bounded by a polynomial in k, where k is the length of p_*. The number of queries used is bounded by $(k+1) + k|A| + k(k-1)/2$. This concludes the proof of Theorem 3. ■

Theorem 4 *Any algorithm that exactly identifies all the patterns of length n using equivalence, membership, and subset queries must make at least* $2^n - 1$ *queries in the worst case.*

PROOF: If p is a pattern that contains no variables, then $L(p) = \{p\}$. Thus, the class of singleton sets of binary strings is a subclass of the class of languages of patterns of length n. Moreover, the intersection language (the empty set) is not a pattern language, so we may apply Lemma 2 to conclude that any algorithm that exactly identifies all the patterns of length n using unrestricted equivalence, membership, and subset queries must make at least $2^n - 1$ queries, which proves Theorem 4. ■

5. Disjointness queries

Valiant (1984) describes a *possibility-oracle*, which takes as input a term t and determines whether or not t has any satisfying instances in common with the unknown formula ϕ_*. The possibility-oracle answers restricted disjointness queries.

Shapiro (1981, 1982, 1983) has also made use of disjointness queries in his work on automatic debugging of Prolog programs. A brief description will give some idea of the uses of queries in his system; we direct the reader to the original papers for full details.

5.1 Queries in Shapiro's debugging system

In Shapiro's system the user is assumed to have in mind a model of the correct behavior of his or her program, consisting of a collection of named procedures and, for each procedure, the set of tuples of ground terms on which it is true. For example, the user might be writing a procedure $member(X, Y)$ that should be true whenever X is a member of the list Y, or a procedure $reverse(X, Y)$ that should be true whenever the list Y is the reverse of the list X.

In addition, there is a current program, represented as a set of axioms in Prolog, which may or may not be correct for the intended model. It is assumed that if the program is not correct, this will eventually be discovered and a counterexample provided. This is in essence an equivalence query.

The counterexample may be an atom $P(t_1, \ldots, t_k)$ that is provable from the program and not true in the correct model. In this case, the system takes any computation that derives the incorrect atom from the program and, using membership queries, locates an incorrect axiom in the program. The membership queries take the form of asking whether ground atoms are true or false in the intended model. For example, the user might be queried whether $member(3, [1, 2])$ is true or not.

If the counterexample is an atom $P(t_1, \ldots, t_k)$ that is true in the intended model but is such that the program terminates without proving it, then a different diagnosis algorithm is applied to locate an "incomplete procedure," that is, a procedure that requires further axioms. This diagnosis algorithm uses what Shapiro calls an *existential query*, which gives the user an atom containing variables and asks if there is any instantiation of the variables that makes the atom true in the intended model. If the user answers *yes*, he or she is then asked to supply instantiations that make the atom true in the intended model.

An example given by Shapiro (1983) in the course of debugging an incorrect insertion sort is the query: $isort([2,1], Z)$? This can be restated as "is there any value of Z for which the insertion sort procedure $isort([2,1], Z)$ is true?" The user answers *yes* and is queried for a value of Z. The user then supplies $Z = [1,2]$, and the diagnosis algorithm continues.

In our terminology, this is a disjointness query, testing whether the set of ground instances of the atom $isort([2,1], Z)$ has any elements in common with the ground atoms making up the correct behavior of *isort*, and if so, to supply one. (In Shapiro's system the user must be prepared to enumerate all of the common instances.)

As thus described, Shapiro's system makes use of equivalence, membership, and disjointness queries. The system also uses an additional type of query in cases where the program fails to terminate, but this is beyond the scope of our discussion.

5.2 k-CNF formulas using disjointness queries

The following polynomial time algorithm, due to Raymond Board (personal communication, 1987), exactly identifies k-CNF formulas using restricted disjointness queries.[4]

Let T_k denote the set of terms consisting of a conjunction of k or fewer literals. The complements of elements of T_k is the set of all clauses that contain k or fewer literals. For each term t in T_k, the algorithm makes a disjointness query. (Note that each term is a 1-CNF formula, so this is allowed.) The final output ϕ is the conjunction of the negations of all the terms t found to be disjoint from the unknown formula ϕ_*. Note that the number of queries made by the algorithm is bounded by $(2n+1)^k$.

To see that ϕ is equivalent to ϕ_*, first consider any assignment a that satisfies ϕ_*. Then a must falsify every term t found to be disjoint from ϕ_*, so it satisfies all the clauses making up ϕ, and hence satisfies ϕ. Conversely, consider any clause c from ϕ_*. The negation of c is a term in T_k that is disjoint from ϕ_*, so c must be included in ϕ. Hence any assignment that satisfies ϕ must satisfy all the clauses of ϕ_*, that is, must satisfy ϕ_*.

[4]The dual algorithm for k-DNF formulas uses restricted exhaustiveness queries.

5.3 A very small class of context-free languages

In this section we give an artificial example to illustrate the use of disjointness queries. Let A be a fixed finite alphabet and let A^+ denote the set of nonempty finite strings over A. Let f be any mapping of A into the integers. We extend f additively to any string $w = a_1 a_2 \cdots a_n$ in A^+ by $f(w) = f(a_1) + f(a_2) + \ldots + f(a_n)$.

Now we define a set of strings, $L(f) \subseteq A^+$, as follows. The string $w = a_1 a_2 \cdots a_n$ is in $L(f)$ if and only $f(w) = 0$, and for each $i = 1, 2, \ldots, n-1$, $f(a_1 a_2 \cdots a_i) \geq 0$. Let C denote the class of all languages $L(f)$ as f ranges over all functions from A to the integers.

Some simple examples follow. If $A = \{a, b\}$ and $f(a) = f(b) = 1$, then $L(f)$ is the empty set. If $A = \{a, b, c\}$ and $f(a) = 0$, $f(b) = 0$, and $f(c) = 1$, then $L(f) = \{a, b\}^+$. If A consists of a left and a right parenthesis and f assigns 1 to the left parenthesis and -1 to the right parenthesis, then $L(f)$ is the language of nonempty strings of balanced parentheses.

C is a subclass of the deterministic one counter languages, which Berman and Roos (1987) show can be learned efficiently using equivalence and membership queries. C does not include any finite subset of A^+ except the empty set.

Theorem 5 *There is an algorithm that exactly identifies every language in the class C that uses only disjointness queries and runs in time polynomial in $|A|$ and the length of the longest counterexample.*

PROOF: We assume A is known. Let f_* be an unknown function mapping A to the integers. The input to a disjointness query is an arbitrary function f mapping A to the integers. If $L(f) \cap L(f_*)$ is empty, the reply is *yes*, otherwise the reply is *no* with a counterexample from $L(f) \cap L(f_*)$.

The algorithm begins by determining for each $a \in A$ whether $f_*(a) = 0$. This is true if and only if a disjointness query with $\{a\}^+$ is answered *no*. Let Z denote the set of symbols $a \in A$ such that $f_*(a) = 0$.

For every pair a_1 and a_2 of distinct symbols from $A - Z$, the algorithm makes a disjointness query with $\{a_1, a_2\}^+$. If the reply is *yes* then $f_*(a_1)$ and $f_*(a_2)$ are both positive or both negative.

Otherwise, the counterexample is a nonempty string w in $L(f_*)$ that contains r occurrences of a_1 and s occurrences of a_2. Without loss of generality, assume that the first symbol in w is a_1. Then we know that $f_*(a_1) > 0$ and $f_*(a_2) < 0$, and moreover,

$$r f_*(a_1) = -s f_*(a_2),$$

and neither r nor s is 0.

If we find any function f such that $f(a) = 0$ for all $a \in Z$ and f satisfies all the inequalities and equations above for all pairs a_1 and a_2 from $A - Z$, then $L(f) = L(f_*)$. Angluin (1986) sketches one way of finding such an f.

The number of queries needed to determine f is bounded by the square of $|A|$, and the computations involved are bounded by a polynomial in $|A|$ and the lengths of the counterexamples provided. This concludes the proof of Theorem 5. ■

6. The double sunflower: A lower bound for all six queries

The following construction, dubbed the "double sunflower," was devised by participants in the learning seminar at the University of California, Santa Cruz, in the fall of 1987, and communicated to the present author by Michael Kearns. It demonstrates a domain with N concepts in which $N - 1$ queries are necessary, even given the full set of query types, settling an open problem posed by Angluin (1986).

Let $n > 0$ be given. Let $N = 2^n$. Let $X = \{x_1, x_2, \ldots, x_N\}$ and $Y = \{y_1, y_2, \ldots, y_N\}$. Let z_1 and z_2 be two special points not contained in X or Y. Define the universe as $U = X \cup Y \cup \{z_1, z_2\}$. Thus, U contains $2^{n+1} + 2$ points. For each $j = 1, \ldots, N$, let

$$C_j = \{z_1, x_j\} \cup (Y - y_j).$$

The hypothesis space consists of the N sets C_j. Note that C_j always contains z_1, never contains z_2, contains only x_j among the x's, and contains all but y_j among the y's.

To see that $N - 1$ queries are necessary for any algorithm that exactly identifies this class of concepts, even given the full complement of equivalence, membership, subset, superset, disjointness, and exhaustiveness queries, we exhibit the following adversary. Let S denote the set of indices of hypotheses C_j that are compatible with all the queries answered so far. Initially S is all N indices. As long as S contains at least two elements, the adversary answers queries as follows.

- An equivalence query with C_j is answered *no*, and the counterexample x_j is given. The at most one element i (namely $i = j$) of S such that $x_j \in C_i$ is removed from S.
- A membership query with x_j is answered *no*. The at most one element i of S such that $x_j \in C_i$ is removed from S. A membership query with y_j is answered *yes*. The at most one element i of S such that $y_j \notin C_i$ is removed from S. A membership query with z_1 is answered *yes*, and one with z_2 is answered *no*. In either case, no elements are removed from S.

- A subset query with C_j is answered *no*, and the counterexample x_j is given. The at most one element i of S such that $x_j \in C_i$ is removed from S.
- A superset query with C_j is answered *no*, and the counterexample y_j is given. The at most one element i of S such that $y_j \notin C_i$ is removed from S.
- A disjointness query with C_j is answered *no*, and the counterexample z_1 is returned. No elements are removed from S.
- An exhaustiveness query with C_j is answered *no*, and the counterexample z_2 is returned. No elements are removed from S.

A correct exact identification algorithm for this domain must reduce the set S to one element, and the adversary guarantees that at most one element will be removed from S by each query. Hence the lower bound of $N - 1$ queries is proved.

7. Summary, remarks, and open questions

We have described efficient algorithms and lower bounds for the use of queries in several specific domains. These results are summarized in Table 1. The first column lists the specific domains we have discussed, and the second column gives a reference to the description of the domain. The third column, labeled *sufficient*, states a minimal set of query-types that has been shown to suffice for efficient exact identification in the specified domain. The fourth column, labeled *insufficient*, specifies a maximal set of query-types for which an exponential lower bound on exact identification has been shown for the specified domain.[5] The types of queries are identified by numbers according to the following scheme: (1) equivalence, (2) membership, (3) subset, (4) superset, (5) disjointness, (6) exhaustiveness.

A number of open problems are implicit in Table 1. For example, can membership queries be shown to be essential to efficient identification of the regular sets or monotone DNF formulas? Are disjointness queries of any help in identifying the pattern languages?

In any practical setting, the answers to queries of all types are likely to be contaminated with errors. The errors may reflect some consistent ignorance or bias of the informant, or may be generated by some random process. Work by Valiant (1985), Kearns and Li (1987), Angluin and Laird (1987), and Laird (1987) on malicious and random errors has begun to clarify the effect of errors on identification and learning.

[5]A number superscripted with a minus sign denotes the restricted version of the corresponding query, that is, with no counterexamples returned. For example, 1^- denotes restricted equivalence queries. The number 2^+ denotes nonterminal membership queries.

Table 1. Summary of results for specific domains.

domain	*reference*	*sufficient*	*insufficient*
singleton languages	2.1	4	1, 2, 3, 5
k-CNF formulas	2.5, 5.2	1 or 5^-	1^-, 2, 3
k-DNF formulas	2.5, 5.2	1 or 6^-	1^-, 2, 4
monotone DNF formulas	3.1	1, 2	1^-, 2, 3, 5, 6
regular languages	3.2	1, 2	1^-, 2, 3
k-bounded CFLs	3.2	1, 2^+	
k-term DNF formulas	3.2	1, 2	1^-, 2, 3
k-clause CNF formulas	3.2	1, 2	1^-, 2, 4
pattern languages	4.1	4^-	1, 2, 3
very restricted CFLs	5.3	5 or 1, 2	
double sunflower	6	none	1, 2, 3, 4, 5, 6

Valiant (1984) makes use of two additional types of queries specific to DNF formulas: *relevant possibility* and *relevant accompaniment*. Shapiro (1981, 1982, 1983) makes use of one additional type of query to help diagnose nonterminating Prolog programs. Still other sources of information will prove to be relevant for other specific domains.

Angluin (1987c) introduces a new type of query, called a "request for a hint," for the domain of propositional Horn sentences. The answer to such a query is intended to model a partial explanation of how a conclusion follows from a set of premises. Formal models of the "explanations" or "reasons" that may be supplied by a human expert are an important area of research.

Acknowledgements

I have enjoyed conversations about this material with Manuel Blum, Bill Gasarch, David Haussler, Mike Kearns, Phil Laird, Lenny Pitt, Udi Shapiro, and Carl Smith. I thank the National Science Foundation for their support through grant number IRI-8404226. This paper is a revision of an earlier report (Angluin, 1986), but is not identical to it.

References

Angluin, D. (1980). Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, *21*, 46–62.

Angluin, D. (1986). *Types of queries for concept learning* (Technical Report YALEU/DCS/RR-479). New Haven, CT: Yale University, Department of Computer Science.

Angluin, D. (1987a). *Learning k-bounded context-free grammars* (Technical Report YALEU/DCS/RR-557). New Haven, CT: Yale University, Department of Computer Science.

Angluin, D. (1987b). *Learning k-term DNF formulas using queries and counterexamples* (Technical Report YALEU/DCS/RR-559). New Haven, CT: Yale University, Department of Computer Science.

Angluin, D. (1987c). *Learning propositional Horn sentences with hints* (Technical Report YALEU/DCS/RR-590). New Haven, CT: Yale University, Department of Computer Science.

Angluin, D. (1987d). Learning regular sets from queries and counterexamples. *Information and Computation*, *75*, 87–106.

Angluin, D., & Laird, P. (1987). Learning from noisy examples. *Machine Learning*, *2*, 343–370.

Angluin, D., & Smith, C. (1983). Inductive inference: Theory and methods. *Computing Surveys*, *15*, 237–269.

Barzdin, J. M., & Freivald, R. V. (1972). On the prediction of general recursive functions. *Soviet Mathematics Doklady*, *13*, 1224–1228.

Berman, P., & Roos, R. (1987). Learning one-counter languages in polynomial time. *Proceedings of the Twenty-Eighth IEEE Symposium on Foundations of Computer Science* (pp. 61–67). New York: The Institute of Electrical and Electronics Engineers.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1986). Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing* (pp. 273–282). Berkeley, CA: The Association for Computing Machinery.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1987). *Learnability and the Vapnik-Chervonenkis dimension* (Technical Report UCSC-CRL-87-20). Santa Cruz: University of California, Computer Research Laboratory.

Haussler, D. (1986). Quantifying the inductive bias in concept learning. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 485–489). Philadelphia, PA: Morgan Kaufmann.

Haussler, D. (in press). Quantifying the inductive bias: AI learning algorithms and Valiant's framework. *Artificial Intelligence.*

Kearns, M., & Li, M. (1987). *Learning in the presence of malicious errors* (Technical Report TR-03-87). Cambridge, MA: Harvard University, Center for Research in Computing Technology.

Kearns, M., Li, M., Pitt, L., & Valiant, L. (1987). On the learnability of Boolean formulae. *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (pp. 285–295). New York: The Association for Computing Machinery.

Knobe, B., & Knobe, K. (1976). A method for inferring context-free grammars. *Information and Control*, *31*, 129–146.

Laird, P. (1987). *Learning from good data and bad.* Doctoral dissertation, Department of Computer Science, Yale University, New Haven, CT.

Levy, L., & Joshi, A. (1978). Skeletal structural descriptions. *Information and Control*, *39*, 192–211.

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*, 285–318.

Pitt, L., & Valiant, L. (1986). *Computational limitations on learning from examples* (Technical Report TR-05-86). Cambridge, MA: Harvard University, Center for Research in Computing Technology.

Sakakibara, Y. (1987a). *Inductive inference of logic programs based on algebraic semantics* (Technical Report No. 79). Numazu, Japan: Fujitsu Limited, International Institute for Advanced Study of Social Information Science.

Sakakibara, Y. (1987b). *Inferring parsers of context-free languages from structural examples* (Technical Report No. 81). Numazu, Japan: Fujitsu Limited, International Institute for Advanced Study of Social Information Science.

Sammut, C., & Banerji, R. (1986). Learning concepts by asking questions. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Shapiro, E. (1981). A general incremental algorithm that infers theories from facts. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 446–451). Vancouver, B.C., Canada: Morgan Kaufman.

Shapiro, E. (1982). Algorithmic program diagnosis. *Proceedings of the Ninth ACM Symposium on Principles of Programming Languages* (pp. 299–308). Albuquerque, NM: The Association for Computing Machinery.

Shapiro, E. (1983). *Algorithmic program debugging.* Cambridge, MA: MIT Press.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*, 1134–1142.

Valiant, L. G. (1985). Learning disjunctions of conjunctions. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 560–566). Los Angeles, CA: Morgan Kaufmann.