Maximal Machine Learnable Classes*

John Case

Department of CIS, University of Delaware, Newark, Delaware 19716

and

Mark A. Fulk

Geneseo Valley Software, 92 Monteray Road, Brighton, New York 14618

Received April 21, 1987; revised June 18, 1997

A class of computable functions is maximal iff it can be incrementally learned by some inductive inference machine (IIM), but no infinitely larger class of computable functions can be so learned. Rolf Wiehagen posed the question whether there exist such maximal classes. This question and many interesting variants are answered herein in the affirmative. Each IIM can be infinitely improved upon! Also discussed are the problems of algorithmically finding the improvements proved to exist.

1. INTRODUCTION

Let \( \mathcal{PR} \) denote the class of primitive recursive functions. There is a machine \( M_{\mathcal{PR}} \) which, when fed successively more values of any \( f \in \mathcal{PR} \), after finitely many trials and error output programs, eventually settles on an output program which computes \( f \) [16,3]. We say \( M_{\mathcal{PR}} \) incrementally learns (or Ex-identifies) \( \mathcal{PR} \) [10]. \( \mathcal{PR} \) is a large and inclusive class of computable functions [11]. It is interesting, then, to ask whether there are classes of computable functions strictly larger than \( \mathcal{PR} \) which are also incrementally learnable (i.e., Ex-identifiable).

Let \( \mathcal{Ex} \) denote the class of all classes of computable functions each Ex-identifiable by some machine. Then, for example, \( \mathcal{PR} \in \mathcal{Ex} \). It is well known and easy to show that, if \( \mathcal{F} \in \mathcal{Ex} \) and \( \mathcal{F} \subseteq \mathcal{F}' \), where \( (\mathcal{F}' - \mathcal{F}) \) is finite, then \( \mathcal{F}' \in \mathcal{Ex} \) too. Hence, if we add any finite collection of computable functions to \( \mathcal{PR} \) we still get a class in \( \mathcal{Ex} \). This discussion suggests the following.

**Definition 1.** A class of computable functions \( \mathcal{F} \) is Ex-maximal if both \( \mathcal{F} \in \mathcal{Ex} \) and there is no \( \mathcal{F}' \in \mathcal{Ex} \) such that \( \mathcal{F} \subseteq \mathcal{F}' \) and \( (\mathcal{F}' - \mathcal{F}) \) is infinite.

A variant of the question of the first paragraph in this section is whether \( \mathcal{PR} \) is Ex-maximal. It trivially is not. For example, using the enumeration technique [3,15,16] one can show that each level of the Peter-hierarchy [22,27] is in \( \mathcal{Ex} \).

Wiehagen [30] asked whether there is an Ex-maximal class. In Section 3 below we answer the question and interesting variants each negatively. A positive, informal restatement is Infinite improvement is always possible!

In Section 4 below we discuss the problems of algorithmically finding the improvements proved to exist.

2. PRELIMINARIES

The definitions, notations, and facts given here all from [10,16,26].

\( N \) is the set of natural numbers \( \{0, 1, 2, \ldots\} \); \( i, n, m, x, y, z \), as well as subscripted and superscripted versions of these variables.

1 Roughly, \( \mathcal{PR} \) is characterized as the class of functions computable by assembly language programs whose only loop structure is equivalent for the for-loop [18,29].
these letters, range over \( N \). The asterisk \( * \) is used as a symbol in various locations that might otherwise be occupied by a number, \( \sigma \) ranges over \( N \cup \{ * \} \).

\( \varphi \) is an arbitrary computable numbering of the partial computable functions \( [17, 23, 24, 25, 28] \). Hence, \( \varphi_p \) is the partial computable function: \( N \to N \) computed by \( \varphi \)-program \( p \). Lower case Greek letters other than \( \varphi \) range over partial computable functions. \( \mathcal{R} \) is the set of computable functions; \( f \) and \( g \) range over \( \mathcal{R} \).

The domain of a partial computable function \( \psi \) is denoted by \( \text{dom} \psi \). We write \( \psi(x) \downarrow \) if \( x \in \text{dom} \psi \), that is, if \( \psi(x) \) is defined. We write \( \psi_1(x) = \psi_2(x) \downarrow \) if either both \( \psi_1 \) and \( \psi_2 \) are undefined on \( x \) or both are defined and equal on \( x \). We write \( \psi_1 \equiv \psi_2 \downarrow \{ x \in N : \psi_1(x) \neq \psi_2(x) \} \) has at most \( n \) elements; in this case, we call \( \psi_1 \) an \( n \)-variant of \( \psi_2 \). We write \( \psi_1 \equiv \psi_2 \downarrow \{ x \in N : \psi_1(x) \neq \psi_2(x) \} \) is finite; in that case, we call \( \psi_1 \) a finite variant of \( \psi_2 \). If \( P \) is a predicate on \( N \), we write \( (\forall x) P(x) \downarrow \) if the set of numbers that do not satisfy \( P \) is finite.

The variables \( \sigma \), \( \sigma_0 \), and \( \sigma_1 \) range over finite sequences of numbers; consistently with the use of lower-case Greek letters given above, finite sequences are considered to be functions from initial segments of \( N \) to \( N \). We write \( \sigma_0 \leq \sigma_1 \) and \( \sigma_0 < \sigma_1 \) if \( \sigma_0 < \sigma_1 \) respectively. \( \sigma_0 \) is an initial subsequence of \( \sigma_1 \) and \( \sigma_0 \) is an initial subsequence of \( f(0) \).

An inductive inference (or learning) machine (IIM) is an algorithmic device which takes, as input, a sequence of values \( f(0), f(1), ... \) from some computable function \( f \), and which, from time to time, as it is receiving its input, outputs a computer program \( [10, 26] \).

The variables \( M \), \( M_0 \), and \( M' \) range over inductive inference machines. Following \([10] \), we assume that the output of an IIM is well defined after any sequence of inputs (including the empty sequence); \([3, 10] \) demonstrate that our assumptions about IIMs gnat all notion of generality.

We write \( M/f \downarrow \) if \( \exists x (\forall \sigma < f) (M/\sigma = x) \). That is, \( M/f \downarrow \) if, as \( M \) is receiving its input, it eventually outputs some program \( x \) and thereafter never outputs any other program. If \( M/f \downarrow \), we write \( M(f) \) for that final program.

We say that \( M \text{ Ex}^a \)-identifies functions \( f, [10, 16] \) if both \( M(f) \downarrow \) and \( \varphi_{M(f)} \equiv f \). That is, \( M \) is as it is receiving successive values of \( f \) as input, eventually outputs a program for an \( a \)-variant of \( f \) (finite variant if \( a = * \) ) and thereafter never outputs a different program.

We say that \( M \text{ Be}^a \)-identifies functions \( f, [2, 10] \) if \( (\forall \sigma \in f) [\varphi_{M(f)} \equiv \sigma \equiv f] \). That is, \( M \) is fed the values of \( f \), then, after \( M \) has received some suitable amount of input, thereafter each of the outputs of \( M \) is a program for an \( a \)-variant of \( f \).

For a given IIM \( M \), \( \text{Ex}^a(M) \) is the class of functions that \( M \text{ Ex}^a \)-identifies. Similarly, \( \text{Be}^a(M) \) is the class of functions that \( M \text{ Be}^a \)-identifies. \( \text{Ex}^a = \{ (S \subseteq \mathbb{R} : (3 \mathbb{M}[f \in \text{Ex}^a(M)]) \} \). \( \text{Be}^a = \{ (S \subseteq \mathbb{R} : (3 \mathbb{M}[f \in \text{Be}^a(M)]) \} \). Note that \( \text{Ex}^a \) is a class of classes of computable functions, as is \( \text{Be}^a \).

### 3. RESULTS

The following definition generalizes \( \text{Ex}^a \)-maximality from Definition 1 in Section 1 above.

**Definition 2.** Suppose \( I \) is a class of classes of computable functions (e.g., \( I \) could be \( \text{Ex}^a \) or \( \text{Be}^a \)). A class of computable functions \( \mathcal{S} \) is \( I \)-maximal if both \( \mathcal{S} \subseteq I \) and \( (\mathcal{S} - \mathcal{S}') \) is infinite.

**Theorem 1.** For each \( a \in N \cup \{ * \} \), there is no \( \text{Ex}^a \)-maximal class.

**Proof.** Assume we are given a set \( \mathcal{S} \subseteq \text{Ex}^a \). Let \( M \) be an IIM such that \( \mathcal{S} \subseteq \text{Ex}^a(M) \). We are done if we can construct an IIM \( M' \) such that \( \text{Ex}^a(M) \subseteq \text{Ex}^a(M') \) and \( (\text{Ex}^a(M') - \text{Ex}^a(M)) \) is infinite. The crucial trick is to consider the following two cases:

**Case (1).** For each finite sequence \( \sigma \), there is an \( f \in \text{Ex}^a(M) \) that extends \( \sigma \) (i.e., which is such that \( \sigma \subset f \)).

By Kleene's S-m-n theorem \([26] \), there is a computable function \( g \) such that, for each \( i \), \( \text{program} \ g(i) \) runs the following computation in stages and \( \varphi_{g(i)} \) is the union of \( \{ \tau_0, \tau_1, \ldots \} \) from the stages.

**Stage 0.** \( \tau_0 = \tau_0 = (0, i), y_0 = 0 \).

**Stage \( n+1 \).** \( g(i) \) searches for \( \tau_{n+1} \) and \( y_{n+1} \) such that \( \tau_n \leq \tau_{n+1} \), \( y_{n+1} \neq \delta_{\tau_{n+1}} \), and \( \varphi_{M_{\tau_{n+1}}}(y_{n+1}) \downarrow \). \( g(i) \) then sets \( \tau_{n+1} \) to be such that \( \tau_{n+1} \leq \tau_{n+1} \) and \( \tau_{n+1}(y_{n+1}) = 1 + \varphi_{M_{\tau_{n+1}}}(y_{n+1}) \), and \( \tau_{n+1}(x) = 0 \) for all \( x < y_{n+1} \) such that \( x \neq \delta_{\tau_{n+1}} \).

\( \{ \varphi_{g(i)} : i \in N \} \) is easily seen to be an infinite set of partial computable functions from the fact that, for each \( i \), \( \varphi_{g(i)}(0) \downarrow = i \).

To finish Case (1), we proceed with a series of claims.

**Claim 1.** For each \( i \), \( \varphi_{g(i)} \) is total.

**Proof of Claim 1.** We begin by proving that every stage in the construction of \( \varphi_{g(i)} \) halts. That Stage 0 halts is obvious.

Consider Stage \( n+1 \), for an arbitrary \( n \). Since we are in Case (1), there is some \( f \) such that \( \tau_n \leq f \) and \( f \in \text{Ex}^a(M) \). From the definition of \( \text{Ex}^a \), for all sufficiently long \( \sigma \equiv f \), \( \varphi_{M_{\tau_n}} \equiv f \). Choose such a \( \sigma \) such that \( \tau_n \equiv \sigma \). All sufficiently large \( z \) satisfy \( \varphi_{M_{\tau_n}}(z) \downarrow = f(z) \); choose such a \( z \) large enough that \( z \neq \delta_{\tau_n} \). The chosen \( \sigma \) and \( z \) satisfy the requirements on \( \tau_{n+1} \) and \( y_{n+1} \) in Stage \( n+1 \); therefore the search in Stage \( n+1 \) halts, and we have shown that every stage halts.

It is easily seen that, for each \( n, (\tau_{n+1} - \tau_n) \) is not empty. Immediately we have Claim 1.

\( ^5 \) We will have more to say about the dichotomy provided by these cases in Section 4 below.
CLAIM 2. For each $i$, $\varphi_{g(i)} \notin \text{Ex}^a(M)$.

Proof of Claim 2. Suppose by way of contradiction otherwise. Then $M(\varphi_{g(i)}) \uparrow$. Note that $y_0 < y_1 < \ldots$. For all $m$, $\varphi_{g(i)}(y_m) = 1 + \varphi_{M_{\varphi_{g(i)}}}(y_m) \neq \varphi_{M_{\varphi_{g(i)}}}(y_m)$. For all sufficiently large $m$, $M(\tau_m) = M(\varphi_{g(i)})$. Therefore, for all sufficiently large $m$, $\varphi_{M_{\varphi_{g(i)}}}(y_m) \neq \varphi_{g(i)}(y_m)$. □

CLAIM 3. $(\text{Ex}^a(M) \cup \{\varphi_{g(i)} : i \in N\}) \notin \text{Ex}^a$.

Proof of Claim 3. Let $M'$ be an IIM such that

$$M'(\sigma) = \begin{cases} g(\sigma(0)), & \text{if } \sigma \subseteq \varphi_{g(i)}, \\ (M(\sigma)), & \text{otherwise}. \end{cases}$$

Suppose $f \in \text{Ex}^a(M)$. Then $f \neq \varphi_{g(i)}(0)$, and for all sufficiently large $a \subseteq f$, $\sigma \not\subseteq \varphi_{g(i)}$ and $M'(\sigma) = M(\sigma)$. Then $M'(f) \downarrow = M(f)$ and $f \in \text{Ex}^a(M')$.

Suppose $i$ given. For all $\sigma \subseteq \varphi_{g(i)}$, $M'(\sigma) = g(i)$, so $M'(\varphi_{g(i)}) \downarrow = g(i)$ and therefore $\varphi_{g(i)} \in \text{Ex}^a(M') \subseteq \text{Ex}^a(M)$. □

Claims 1, 2, and 3 prove Theorem 1 in Case (1).

Case (2). Not Case (1); that is, there is no $\sigma_0$ such that $(\forall y) [\sigma_0 \subseteq f] \Rightarrow [f \notin \text{Ex}^a(M)]$. Let $S_0$ be some easily identifiable infinite set of total extensions of $\sigma_0$, for example, the set of functions of finite support $[3]$ containing $\sigma_0$. Let $M_0$ be an IIM that $\text{Ex}^a$-identifies $S_0$.

Let $M'$ be such that

$$M'(\sigma) = \begin{cases} M_0(\sigma), & \text{if } \sigma_0 \subseteq \sigma, \\ (M(\sigma)), & \text{otherwise}. \end{cases}$$

We will prove that $(\text{Ex}^a(M) \cup S_0) \subseteq \text{Ex}^a(M')$.

Suppose $f \in \text{Ex}^a(M)$. Then, for all $\sigma_0 \subseteq f$, $\sigma_0 \not\subseteq \sigma$, and $M'(\sigma) = M(\sigma)$. Therefore, $M'(f) \downarrow = M(f)$, and $f \in \text{Ex}^a(M')$.

Suppose $f \in S_0$. Then, for all sufficiently large $\sigma_0 \subseteq f$, $\sigma_0 \subseteq \sigma$, and $M'(\sigma) = M_0(\sigma)$. Since $f \in \text{Ex}^a(M_0)$, it follows that $M'(f) \downarrow = M_0(f)$ and $f \in \text{Ex}^a(M')$. □

THEOREM 2. For each $m \in N$, there is no $\text{Be}^m$-maximal class.

Proof. This proof is identical to the proof of Theorem 1 above, except for the following modifications in Case (1). In Stage $n + 1$, $g(i)$ searches for $\tau_{n+1}$ and $y_{n+1}, y_{n+1}^{m}, \ldots$ such that, for each $i \leq m$, $\varphi_{M_{\varphi_{g(i)}}}(y_{n+1}^{m}) i$. For each $i \leq m$, set $\tau_{n+1}(y_{n+1}^{m}) = 1 + \varphi_{M_{\varphi_{g(i)}}}(y_{n+1}^{m})$. Thus, infinitely many of the output programs of $M$ are made wrong on $m + 1$ inputs, when $f$ is fed $g_{g(i)}$. □

By Harrington’s surprising result in [10], $\not\exists \in \text{Be}^*$. Hence, since $\not\exists$ cannot even be finitely improved upon, $\not\exists$ is a $\text{Be}^m$-maximal class.

It is easy to modify the proofs of Theorems 1 and 2 to apply to identifiable classes of 0, 1-valued functions. The first change is to have the sequence of values of $\varphi_{g(i)}$ begin with 0’s followed by a 1. The second change is to substitute proper subtraction for addition in the definition of $\tau_{n+1}$ in Stage $n + 1$ of the constructions in Case (1) of each proof. We thus see that, if we restrict $\text{Ex}^a$ and $\text{Be}^a$ to 0, 1-valued computable functions, there are still no $\text{Ex}^a$-maximal or $\text{Be}^a$-maximal classes.

References [3, 8, 10, 14, 19] contain definitions of restricted classes of inductive inference machines called Popperian, postdictively complete, postdictively consistent, and reliable (or strong). It is easy to see that the machines $M'$ constructed in the proof of Theorem 1 have any of these properties that apply to the input machines $M$.

4. RELATED WORK AND FURTHER PROBLEMS

Another model of machine learning, called formal language learning, involves sequential presentation of the elements of a recursively enumerable language $L$ (positive data or text for $L$) to an IIM which must, after finitely many trial and error rounds, eventually output a grammar (or grammars) for $L$. See [6, 9, 16, 20, 21] for definitions and basic results. The language learning analogs of $\text{Ex}^a$ and $\text{Be}^a$ are called $\text{TxtEx}^a$ and $\text{TxtBe}^a$, respectively. Surprisingly, while the class $\not\exists$ of all finite languages is in $\text{TxtEx}^a$, if $L$ is any finite r.e. language, $(\not\exists \cup \{L\})$ is not in $\text{TxtBe}^a$ [6, 16, 20]. For related results see [1, 20].

From Theorems 1 and 2 in Section 3 above, we have that any IIM $M$ for $\text{Ex}^a$-identification or $\text{Be}^a$-identification can be infinitely improved. More specifically, there is an IIM $M'$ which identifies infinitely many more computable functions than $M$ does. It is interesting to ask whether such an $M'$ can be algorithmically found from $M$. Unfortunately, it cannot and not even if $M'$ is supposed to identify only one more computable function than does $M$ [10]. It is further shown in [5] that this latter kind of $M'$ also cannot be found from $M$ by an incremental algorithm which is allowed to change its mind finitely many times about its outputs before settling on a correct output.

The proofs of Theorems 1 and 2 above, usefuly considered a dichotomy of IIMs $M$; those $M$ satisfying Case (1) and those satisfying Case (2). Reference [7] contained the slightly overzealous claim to the effect that we had found two algorithms for finding $M'$ (satisfying Theorem 1): one for when $M$ satisfies Case (1) and another for when $M$ satisfied Case (2). In fact, we had not found an algorithm for the side of this dichotomy corresponding to Case (2). The second author has conjectured that there is none, but this remains open. It is also open whether there is some dichotomy with algorithms for (infinitely) improving IIMs on each side.

REFERENCES

30. R. Wiehagen, personal communication to J. Case.