Mechanical Turkeys

Gordon Belot

Abstract

Some learning strategies that work well when computational considerations are abstracted away from become severely limiting when such considerations are taken into account. We illustrate this phenomenon for agents who attempt to extrapolate patterns in binary data streams chosen from among a countable family of possibilities. If computational constraints are ignored, then two strategies that will always work are learning by enumeration (enumerate the possibilities—in order of simplicity, say—then search for the one earliest in the ordering that agrees with your data and use it to predict the next data point) and Bayesian learning. But there are many types of computable data streams that, although they can be successfully extrapolated by computable agents, cannot be handled by any computable learner by enumeration. And while there is a sense in which Bayesian learning is a fully general strategy for computable learners, the ability to mimic powerful learners comes at a price for Bayesians: they cannot, in general, become highly confident of their predictions in the limit of large data sets and they cannot, in general, use priors that incorporate all relevant background knowledge.

1. Introduction

Consider a situation in which an infinite data stream is being revealed to us one data point at a time and in which we are certain that this data stream has been selected from a particular countable set of possible data streams (*hypotheses*). We may be interested in either of two related inductive problems: to determine which of the competing hypotheses is true (the *problem of identification*); or to become good at predicting the next data point (the *problem of extrapolation*).

This is a context in which long-run learning is, in a sense, easy. A strategy that will always work is *learning by enumeration*: enumerate the salient hypotheses (in order, say, of strictly decreasing simplicity); at each stage of inquiry, conjecture the hypothesis consistent with the data that occurs earliest in your order and use it to arrive at your prediction for the next data point. No matter which hypothesis is true, you will from some point onwards always conjecture the true hypothesis and correctly predict the next data point (each false hypothesis will eventually be ruled out by the data—and only finitely many of them can come earlier in your order than does the true one).

Another strategy that will always work is *Bayesian learning*. Divide your prior credence among the salient hypotheses and update by conditionalization on the data seen (killing off any credence assigned to hypotheses inconsistent with the data and normalizing). So long as your prior assigns positive credence to each hypothesis in play, in the limit of large data sets you will become arbitrarily confident in the true hypothesis and will therefore also become arbitrarily confident in its prediction of the next data point.

Note that for this sort of problem (setting aside fiddly details about hypotheses of equal *a priori* plausibility) the Bayesian approach just described can be thought of as quantitative version of learning by enumeration: a Bayesian prior of this sort induces a rank-ordering of hypotheses by *a priori* plausibility—but also allows us to make quantitative comparisons of relative plausibility. This extra quantitative information has epistemological import: there is an important difference between an agent who always makes the right prediction, but never with more than minimal confidence, and one who not only always makes correct predictions but who also becomes as confident as can be in these predictions in the limit of large data sets.

It has long been known that aspects of the above picture must

Forthcoming in *Journal of Philosophical Logic*

Earlier versions of this were presented at UC Irvine, CMU, ewip, and PSA 2024. Thanks to two anonymous referees, Jeffrey Barrett, Eissa Haydar, Simon Huttegger, Kevin Kelly, Chris Mierzewski, Laura Ruetsche, Tom Sterkenburg, and Francesca Zaffora Blando for discussion. Thanks to Chris Porter for helpful correspondence. 'Thanks' to Ted Sider for getting me to work on this.

be revised in interesting ways if we restrict attention to agents who follow computable strategies for learning (as, presumably, we and the machines that we build must).[1] Gold (1967) introduced a version of the problem of identification in which a computable agent conjectures natural numbers, aiming to eventually settle on the code number of a Turing machine that computes the data stream. For this problem computable learners by enumeration are in fact quite limited in their power: there are learning tasks that can easily be accomplished by some computable learners but which cannot be accomplished by any computable learner by enumeration.[2] And, as was noted in Juhl (1993), in this context Bayesian learning is essentially just a species of learning by enumeration and so suffers from the same shortcoming.[3]

Here we will focus on the problem of extrapolation for computable learners.[4] This subject was initiated by Putnam (1963).[5] His approach was developed and generalized in the Brezhnev era—see Bārzdiņš (1972), Bārzdiņš and Freivalds (1972), and Podnieks (1974, 1975).[6] A sense of the limitations of learning by enumeration as a strategy for computable extrapolators emerged in those classic papers: here, as in the problem of identification, there are learning tasks easily achievable by computable agents that cannot be achieved by any computable learner by enumeration. As these results appear not to be well-known among philosophers, after some preliminaries our first task below will be to survey the limitations of learning by enumeration as a strategy for computable extrapolators, extending the approach of Fortnow *et al.* (1998) in order to put the point in an especially sharp way.

We will then turn to computable Bayesian extrapolators. Here there is good news and bad news. The good news for Bayesians is that *any* class of patterns that can be successfully extrapolated by a computable learner can be successfully extrapolated by a computable Bayesian learner. So there is a sense in which Bayesianism is not restrictive for computable extrapolators. The bad news is that some aspects of the picture that emerge will be unsettling to many Bayesians: in our setting, powerful Bayesian learners will not, in general, become highly confident in their predictions in the limit of large data sets—and they cannot, in general, use priors that incorporate all relevant background knowledge.

[1]For fundamental investigations of the extent to which requiring computability can impair the power of learning strategies, see Kelly (1996) and Jain *et al.* (1999).

[2]This observation goes back to Bārzdiņš (1971) and to Blum and Blum (1975). More generally, as observed in Wiehagen (1976), in this setting there are feasible learning goals that cannot be attained by any learner whose conjectures are always consistent with the data. As noted in Kugel (1977), additional restrictions apply to agents who, further, never abandon a conjecture once made unless it has been ruled out by data. For further discussion and references see Chapter 5 of Jain *et al.* (1999).

[3]For a related result in a more elaborate setting, see Osherson, Stob, and Weinstein (1988). For cases in which Bayesian updating cannot be implemented computably, see Ackerman, Freer, and Roy (2011).

[4]On the relation between the problem of identification and the problem of extrapolation in the computable setting, see, e.g., Angluin and Smith (1983) and Case and Smith (1983). Below, we will work in the simple context in which data streams are binary sequences—although this renders invisible some phenomena that arise for learners facing data streams consisting of natural numbers—see, e.g., Corollary 13 in Zeugmann and Zilles (2008).

[5]For discussions and elaborations of Putnam's approach, see Earman (1992) and Sterkenburg (2019).

[6]See also Blum and Blum (1975).

2. Framework

The sections below aim to present an elementary and self-contained introduction to issues of computation in epistemology via an examination of the problem of extrapolation for computable agents facing computable binary data streams. This preliminary section sets the stage, reviewing conventions, notions, and results required for our subsequent discussion.

2.1 *Bits and Pieces*

We call $\{0,1\}$ the set of *bits*. For any $k \geq 0$, a *k-bit string* is a k-tuple $(b_1, b_2, \ldots, b_k)$ of bits (special case: the empty string $\varnothing$ of zero bits). A *binary sequence* is a function from the positive natural numbers to bits—informally, we picture these as ω-tuples $(b_1, b_2, b_3, \ldots)$ of bits. We denote by 2^ω the set of all binary sequences. If S is a binary sequence and n is a natural number then we write $S(n)$ for the nth bit of S and write $S\restriction n$ for the binary string consisting of the first n bits of S. When a string and a sequence are related in this way, we say that the string is an *initial segment* of the sequence. We write 1^k for the string consisting of k 1's, write 01^ω for the sequence whose initial bit is 0 and whose other bits are all 1, and so on. We say that a set of binary sequences is *dense* if for any binary string, there is a member of the set with that string as an initial segment. For any binary string τ, we write $[\![\tau]\!]$ for the set of binary sequences whose initial bits are given by τ. We call subsets of 2^ω of this type *basic sets*. Special case: $[\![\varnothing]\!] = 2^\omega$. Henceforth, *sequence* always means *one-way infinite binary sequence* and *string* always means *finite binary string*.

2.2 *Computation*

We review some elementary features of the model of computation introduced in Turing (1936).[7] For definiteness we restrict attention to computing machines consisting of: a two-way infinite tape divided into squares, each of which always contains exactly one of three symbols, $*$ (=blank), 0, or 1; and a processing unit, capable of reading and writing these symbols and of moving along the tape one square at a time.

We can think of the specification of a Turing machine as consisting of two steps. First, one specifies the *hardware* for the machine, choosing the number $n \geq 2$ of internal states $q_1, q_2, \ldots\ q_n$ that its processor can be in (q_1 is the distinguished *initial state* and q_n is the distinguished *halting state*). Then one specifies the *software*, choosing a consistent set of commands of the form "if you read *this* symbol while in *such and such* state, replace it by *that symbol*, move *thus* and enter *that* state."[8]

A machine M of this kind determines a (possibly merely partially defined) function φ from binary strings to binary strings. For each string τ, $\varphi(\tau)$ is determined as follows: put the machine into state q_1 with its processing unit immediately to the left of an inscription of τ on the tape (all other squares containing blanks); let the machine run by executing the commands in its software *seriatim*; if the machine eventually enters the halting state, then $\varphi(\tau)$ is the longest binary string on the tape immediately to the left of the processing head; if the machine never enters the halting state, then $\varphi(\tau)$ is undefined.[9] We say that M gives *output* $\varphi(\tau)$ *on input* τ if $\varphi(\tau)$ is defined (otherwise we say that M *gives no output* on this input). And we say that M *computes* the partial computable function φ. We call φ *total* if $\varphi(\tau)$ is defined for each string τ (and call it *merely partial* otherwise). Fixing suitably mechanical ways of using binary strings to code other countable sets, we can also think of the function computed by a Turing machine as (say) a function from strings to natural numbers or as a function from pairs of natural numbers to rational numbers, and so on.

Let M be a Turing machine. If σ is a k-bit string, then we say that M *computes* σ if on input of $1 \leq j \leq k$, M gives as output the jth bit of σ. And if S is a sequence, then we say that M *computes* S if on input of any $k \geq 1$ M gives as output $S(k)$. There are uncountably many binary sequences but only countably many of these can be computable (since there are only countably many Turing machines). Roughly speaking, the computable sequences are those sequences fully characterizable by finitary means. We will denote by $\mathcal{C}$ the set of computable binary sequences. Following computer science usage, we will often call subsets of $\mathcal{C}$ *classes*.

We say that the sequences $S_1, S_2, \ldots$ are *uniformly computable* if there is a Turing machine that on input of positive natural numbers k and ℓ gives as output the ℓth bit of S_k (and we call a class of computable

[7] For full treatments, see, e.g., Odifreddi (1989) or Soare (2016).

[8] So a *command* is a quintuple of the form (q, s, t, d, r) where q and r are internal states ($q \neq q_n$), s and t are tape symbols, and $d \in \{-1, 0, 1\}$. Intuitively, the command (q, s, t, d, r) tells the machine that if its processing unit is in state q and positioned over a square of tape containing tape symbol s, then it should erase this symbol and write t, move one square to the left ($d = -1$)/stay put ($d = 0$)/move one square to the right ($d = 1$), and enter state r. Because q_n is the distinguished halting state, no command gives the machine an instruction for what to do when in that state. Distinct commands (q, s, t, d, r) and (q', s', t', d', r') are *inconsistent* if $q = q'$ and $s = s'$.

[9] We allow the cases where τ or $\varphi(\tau)$ is the empty string.

sequences *uniformly computable* if it admits such a uniformly computable enumeration). Roughly speaking: to say that some sequences are uniformly computable is to say that there is a single Turing machine that knows the structure of each them.

We can assign each Turing machine M a Gödel number $G(M)$.[10] It is a mechanical task to determine whether or not a given natural number is a Gödel number of a Turing machine. So we can build a machine $\mathcal{M}$ that on input of a natural number k searches through the natural numbers in order, looking for the kth smallest number that is a Gödel number of a Turing machine, and then outputs that number—we denote the machine with this Gödel number M_k and say that it has *index* k. So we can think of $\mathcal{M}$ as giving us a means to computably enumerate the Turing machines (i.e., using $\mathcal{M}$ we can build a machine that prints a list of numbers that includes all and only the code numbers of Turing machines).

Given the index of a Turing machine, it is a mechanical task to determine how many internal states it has and what commands are in its software. As Turing showed, with a little more work one can establish the existence of a *Universal Turing machine* $\mathcal{T}$ that on input of a natural number k and a string σ simulates the operation of M_k on input σ: if M_k halts on input σ and gives τ as output, then $\mathcal{T}$ halts on input (k, σ) and gives τ as output; if M_k does not halt on input σ, then $\mathcal{T}$ does not halt on input (k, σ).

As we noted above, it is possible to computably enumerate the set of all Turing machines. This set includes many uninteresting specimens—such as machines that do not halt on any input.[11] The set of machines that compute total functions from strings to strings is much more appealing. But the set of such machines cannot be computably enumerated. For suppose that you have a machine M that when turned on print out a list of indices of machines that compute total functions. Using this machine for one subroutine and a universal Turing machine for another, you could construct a machine $M^\dagger$ that on input of any natural number k gives you as output the number one greater than that given on the same input by the machine whose index was the kth entry in the given list. $M^\dagger$ computes a total function—but its index cannot appear in the list of indices printed by M. Moral: even though (indeed, *because*) it includes some unattractive characters, the set of all Turing machines is computationally well-behaved in ways that the set of machines computing total functions is not.

We will need a magic trick below: the Recursion Theorem. For our purposes, this tells us that if ψ is a partial computable function from pairs of natural numbers to natural numbers, then there is a natural number e such that the partial computable function $\psi(e, \cdot) : \mathbb{N} \to \mathbb{N}$ is computed by the Turing machine with index e.[12] In particular, if the sequences $S_1, S_2, \ldots$ are uniformly computable, then there is a natural number e such that the sequence S_e is computed by the Turing machine with index e.

2.3 Ideals

Recall that an *ideal* on a set X is a non-empty family of subsets of X that is closed under taking subsets and unions—and that an ideal that does not include X itself is said to be *proper*. If $\mathcal{I}$ is a proper ideal on X, then the complement in X of any set in $\mathcal{I}$ is not in $\mathcal{I}$. When, as in each of the cases we will discuss, a proper ideal on X includes each finite subset of X, it is often natural to think of it as collecting the sets that are small or special in some sense: thus, the family of countable sets of sequences is a proper ideal on 2^ω, as is the family of meagre

[10] Choose some suitably mechanical method of associating a code number $G(c)$ to each command c. Then we can assign the Turing machine M with n states and software consisting of commands $c_1, c_2, \ldots, c_k$ (listed in order of increasing code numbers) the Gödel number $2^n p_1^{G(c_1)} p_2^{G(c_2)} \ldots\ p_k^{G(c_k)}$, where p_j is the jth odd prime.

[11] According to the scheme of fn. 10 above, for any $n \geq 2$ the Turing machine with Gödel number 2^n is such a machine (it has n internal states and software consisting of the empty set of commands).

[12] This is a special case of Kleene's Second Recursion Theorem. See, e.g., Odifreddi (1989, Theorem II.2.13).

subsets of 2^ω (relative to the topology generated by the basic sets).[13] In what follows, we will be interested in two proper ideals on the class $\mathcal{C}$ of computable sequences, corresponding to two ways that a class of computable sequences can be computationally special.

A natural computational analog of the condition of being a countable subset of 2^ω is being a uniformly computable subclass of $\mathcal{C}$. But the family of uniformly computable classes is not an ideal on $\mathcal{C}$: while the union of two uniformly computable classes is uniformly computable, a subclass of a uniformly computable class of sequences need not be uniformly computable.[14] But this means that the family of subclasses of uniformly computable classes of sequences forms an ideal on $\mathcal{C}$. We will call any subclass of a uniformly computable class *effectively countable*.[15] A diagonalization argument shows that full class $\mathcal{C}$ of computable sequences is not itself effectively countable: if the sequences S_1, S_2, ... are uniformly computable, then the sequence whose kth bit is the opposite of the kth bit of S_k is a computable sequence that differs from each S_k. It follows that the complement in $\mathcal{C}$ of an effectively countable set is never effectively countable. According to a classic result of Blum and Blum (1975), a class of sequences is effectively countable if and only if there is a sense in which each of its members can be computed rapidly.[16]

A second intuitive sense in which a class of sequences can be computationally special is that it is very easy to diagonalize out of—and this notion can be precisified to yield an effective analogue of meagreness. Let $\mathcal{A}$ be a class of computable sequences. The Banach-Mazur game for $\mathcal{A}$ is a game between two players that is played in a countably infinite succession of rounds R_1, R_2, ... In round R_k, Player 1 plays a non-empty string σ_k and then Player 2 plays a non-empty string τ_k. Player 1 wins if the sequence $S := \sigma_1\tau_1\sigma_2\tau_2\dots$ is in $\mathcal{A}$, otherwise Player 2 wins. We say that $\mathcal{A}$ is *effectively meagre* if Player 2 has a computable winning strategy (i.e., there is a computable function for selecting each τ_k given $\sigma_1,\tau_1,\dots,\sigma_k$ that results in victory no matter how Player 1 chooses the σ_j).[17] The family of effectively meagre classes is a proper ideal on $\mathcal{C}$.[18] Each effectively countable class is also in effectively meagre: if some sequences S_1, S_2, ... are uniformly computable, then a computable winning strategy for Player 2 when the Banach-Mazur game is played for the set of these sequences is to play on round k a single bit that ensures that the sequence constructed is not S_k. But there are effectively meagre classes that are not effectively countable.[19] We can think of the effectively meagre classes as being so easy to diagonalize out of, that one can accomplish this even if forced to work with a malicious adversary.

Remark 1 (Effective Lebesgue Nullsets). Alongside the families of countable and meagre sets of sequences, the family of Lebesgue nullsets constitutes a third natural proper ideal on 2^ω.[20] For present purposes, it can be characterized as follows. A *martingale* is a map m from strings to non-negative real numbers such that: (i) m assigns the empty string the value 1; (ii) for any string σ, $m(\sigma) = 1/2(m(\sigma0) + m(\sigma1))$. We say that a martingale m *succeeds* on a sequence S if $\lim_{n\to\infty} m(S\restriction n) = \infty$.[21]

[13]Recall that a meagre subset of a topological space is one that can be written as a countable union of nowhere dense sets.

[14]Any infinite uniformly computable class of sequences will have uncountably many subclasses, only countably many of which can be uniformly computable.

[15]Warning: this terminology is non-standard.

[16]That is: $\mathcal{A}$ is effectively countable if and only if $\mathcal{A}$ is a subclass of an abstract complexity class in the sense of Blum (1967).

[17]If we drop the requirements that the sequences in $\mathcal{A}$ be computable and that Player 2's strategy be computable, then Player 2 has a winning strategy if and only if $\mathcal{A}$ is a meagre subset of 2^ω (relative to the topology generated by the basic sets)—see Oxtoby (1957).

[18]The notion of effective meagreness was introduced in Mehlhorn (1973) and its Banach-Mazur characterization in Lisagor (1981). For further discussion and references, see Ambos-Spies and Reimann (1998).

[19]This standard fact follows from Proposition 3 and Example 2 below.

[20]On the relation between these three families, see Oxtoby (1980).

[21]Imagine that a gambler begins with unit capital and bets on the identity of each bit in the data stream right before it is revealed. A martingale is a scheme for choosing such bets in a manner acceptable to a disinterested bookmaker who thinks that the data stream is being generated by flipping a fair coin. A martingale succeeds against a sequence just in case the gambler becomes arbitrarily wealthy in the limit of large data sets.

We call a set $\mathcal{A}$ of sequences a *Lebsegue nullset* if there is a martingale that succeeds on every sequence in the set.[22]

This notion, too, can be effectivized. We say that a class of computable sequences is an *effective Lebesgue nullset* if there is an exactly computable martingale m that succeeds against every sequence in the class (to say that m is exactly computable is to say that it is computed by a machine that takes strings as input and gives rational numbers as output).[23]Like the family of effectively meagre classes, the family of effective Lebesgue nullsets is a proper ideal of subclasses of $\mathcal{C}$ that includes effectively countable class of sequences. But there are also classes of sequences that are effective Lebesgue nullsets but not effectively meagre—and vice versa.

The use of the notions of effecitive meagreness and Lebesgue nullsethood in the analysis of computable learners was pioneered in Fortnow *et al.* (1998), where a version of Proposition 5 below was proved with effective Lebesgue nullsethood in place of effective meagreness. It is not difficult to do the same for Proposition 4 below (and hence also for 10). It appears to be an open question whether the same is true for Propositions 2, 7, and 13.

3. The Problem of Extrapolation for Binary Sequences

We are interested in learners who face binary data streams and who are challenged to predict each data point before it is revealed. If our learners are anything like us or the machines that we can build, the learning methods available to them must be implementable by Turing machines. We focus on the case in which each possible data stream is computable: these correspond to finitarily specifiable patterns. Our learners are aiming to grasp the pattern underlying their data streams: to eventually be able to correctly predict each new bit—or at least a suitably large proportion of them.

Definition 1. An *extrapolator* is a Turing machine that computes a (possibly merely partial) function from strings to bits.[24]

So if such a machine halts on input of a given string, it always gives a single bit as output. We call an extrapolator *total* if it computes a totally defined function from strings to bits (i.e., it eventually outputs a bit on input of any string)—otherwise we call it *merely partial*. We allow successful learners to fail to give an output when fed some data sets—but we require that this can happen only finitely often in any case of successful learning. (Recall that if $a_1, a_2, \ldots$ is a bounded sequence of real numbers, then $\limsup_{n\to\infty} a_n$—the *limit superior* of the sequence—is the maximum of the limits attained by its convergent subsequences. The limit superior of a bounded sequence is always defined—and for a convergent sequence, the limit and the limit superior coincide.)

Definition 2. Let S be a sequence and let $\mathcal{M}$ be an extrapolator that with at most finitely many exceptions gives an output on input of any initial segment of S. For $n \geq 1$ let $F(n)$ be the number of $k < n$ such that on input of the first k bits of S, $\mathcal{M}$ does not output the $(k+1)$st bit of S. Then:

a) We say that $\mathcal{M}$ *learns* S if $\lim_{n\to\infty} F(n) < \infty$.[25]
b) We say that $\mathcal{M}$ *weakly learns* S if $\lim_{n\to\infty} F(n)/n = 0$.[26]

[22]This characterization of Lebesgue nullsets is due to Ville (1939, chapter 4). For a modern treatment, see, e.g., Shen, Uspensky, and Vereshchagin (2017, §9.5).

[23]Effective Lebesgue nullset were first investigated in Schnorr (1971a,b), using a more general class of computable martingales and a *prima facie* more general notion of success, which, however, jointly determine the same set of effective nullsets—see, e.g., Downey and Hirschfeldt (2010, Propositions 6.3.8 and 7.1.2). Warning: in the literature on algorithmic randomness one finds a plethora of inequivalent effectivizations of the notion of a Lebesgue nullset (ours arises in the martingale characterization of computable randomness).

[24]Terminology: machines/functions of this type are also known as *prognostication automata* (Bārzdiņš 1972), *extrapolating machines* (Blum and Blum 1975), *prediction methods* (Angluin and Smith 1983), and *next-value functions* (Odifreddi 1999).

[25]This notion was introduced in Bārzdiņš (1972) and Podnieks (1974). Terminology: our *learning* is standardly called called *NV″-learning*; our learning by a total extrapolator is standardly called *NV-learning*.

[26]This is a special case of a notion due to Podnieks and Kinber: for any $0 < \varepsilon \leq 1$, an extrapolator $\mathcal{M}$ that eventually always gives an output on input of initial segments of a computable S NV$''(\varepsilon)$-learns S if $\limsup_{n\to\infty} F(n)/n \leq (1-\varepsilon)$

c) We say that $\mathcal{M}$ *better-than-random learns S* if $\limsup_{n\to\infty} F(n)/n < 1/2$.[27]

Picture that an extrapolator is shown longer and longer initial segments of a sequence and that from a certain point onwards, it always manages to output a prediction of what the next bit will be. To say that the extrapolator learns the sequence is to say that it eventually stops making false next-bit predictions. To say that it weakly learns the sequence is to say that although it may never stop making false predictions, its error rate is zero in the limit of large data sets. To say that it better-than-random learns the sequence is to say, roughly, that in this limit its error rate is less than one-half.[28] Note that if $\mathcal{M}$ learns S (*simpliciter*) then it also weakly learns S. But the converse is not true: the extrapolator that always expects the next bit to be 0 weakly learns the sequence that is all 0's except in the first slot, the tenth slot, the one-hundredth slot, etc.—but it does not learn this sequence. Similarly, if an extrapolator weakly learns a sequence, then it better-than-random learns that sequence (but the converse is not true).

Definition 3. For X=any of the above notions of learning, we say that an extrapolator is a *universal X-learning machine* just in case it X-learns each computable sequence.

Proposition 1 (Podnieks 1975). There exists a universal weak learning machine.

Proof. Consider the extrapolator $\mathcal{M}$ that proceeds as follows: on input of an n-bit string σ it dovetails computations of the Turing machines M_0, $M_1, \ldots M_\ell$ with indices no greater than $\log_2 n$, searching for a machine that outputs the bits of σ on input of $1, 2, \ldots n$, and which also outputs a bit b on input of $n+1$. If such a machine is found, then $\mathcal{M}$ predicts the next bit will be b.[29] Otherwise, it continues computation and $\mathcal{M}(\sigma)$ is undefined. Now suppose that S is a sequence computed by the machine M_N with index N. Then it is clear that on input of initial segments of S of 2^N or more bits, $\mathcal{M}$ will always give an output. Further, for any n, $\mathcal{M}$ can have made at most $\log_2 n$ incorrect predictions in processing the first n bits of S (since at most $\log_2 n$ machines have been considered and any machine that has been used to make a false prediction will never be used again). So $2^N + \log_2 n$ is an upper bound on the number of times that $\mathcal{M}$ fails to make a correct prediction as it is shown the first n bits of S—so the proportion of failures to predict correctly goes to 0 as $n \to \infty$. $\square$

Proposition 2. Let $\mathcal{A}$ be a dense class of computable sequences and let $\mathcal{M}$ be an extrapolator that learns each sequence in $\mathcal{A}$. Then $\mathcal{A}$ is effectively meagre.

Proof. The following is a computable winning strategy for Player 2 when the Banach-Mazur game is played for $\mathcal{A}$. Fix an enumeration σ_1, $\sigma_2, \ldots$ of the strings. On any round, find the string σ that results from concatenating the strings played so far, and dovetail computations of $\mathcal{M}$ on input of $\sigma\sigma_1, \sigma\sigma_2, \ldots$ searching for a σ_k such that $\mathcal{M}$ outputs a bit b on input $\sigma\sigma_k$ (such a σ_k must exist, since by assumption $\mathcal{A}$ includes sequences that extend σ and $\mathcal{M}$ learns each sequence in $\mathcal{A}$). Then play σ_k followed by the opposite of b and hand play back to Player 1. In this way, Player 2 guarantees that the sequence constructed cannot be in $\mathcal{A}$ (since $\mathcal{M}$ errs in its predictions infinitely often when fed this sequence as a data stream). $\square$

Corollary 1 (Podnieks 1974). No extrapolator is a universal learning machine.

(NV$''(\varepsilon)$ learning by a total extrapolator is also called $NV(\varepsilon)$ learning). See Podnieks (1974, 80 f.).

[27] So a (total) extrapolator better-than-random learns S just in case there is some $\varepsilon > 1/2$ such that it NV$''(\varepsilon)$-learns (NV(ε)-learns) S.

[28] This is only roughly accurate because we can have $\limsup_{n\to\infty} F(n)/n < 1/2$ even in cases where the proportion of false predictions bounces around without settling down to a limit.

[29] Here and below: to dovetail a countable set of computations is to perform the first step of the first computation; then to perform the second step of the first computation and the first step of the second computation; then to perform the third step of the first computation, the second step of the second computation, and the first step of the third computation; and so on.

Proof. This follows immediately from the preceding proposition (since $\mathcal{C}$ is dense but is not effectively meagre). □

4. Learning by Total Machines

Total extrapolators are much more limited in their learning power than merely partially defined extrapolators.

Proposition 3 (Bārzdiņš 1972). Let $\mathcal{A}$ be a class of computable sequences. There exists a total extrapolator that learns $\mathcal{A}$ if and only if $\mathcal{A}$ is effectively countable.

Proof. Fix a computable enumeration of the binary strings, $\sigma_1, \sigma_2, \ldots$.

Let $\mathcal{M}$ be a total extrapolator. For each k, Let S_k be the sequence that begins with σ_k, with subsequent bits being chosen to agree with what $\mathcal{M}$ predicts when shown the bits generated so far. The map $(k, \ell) \mapsto S_k(\ell)$ is computable. And if S is learned by $\mathcal{M}$, then S is one of the S_k (to see this, let σ be the shortest initial segment of S such that $\mathcal{M}$ makes correct next-bit predictions on each longer initial segment of S and note that $\sigma = \sigma_k$ for some k). So if every member of a class of sequences is learned by $\mathcal{M}$, then that class must be a subclass of the uniformly computable class of the S_k.

Suppose that $S_1, S_2, \ldots$ are uniformly computable. And for each $k \geq 1$, let $\hat{S}_k$ be the sequence that begins with σ_k, followed by all 0's. The $\hat{S}_1, \hat{S}_2, \ldots$ are uniformly computable. So the $S_1^*, S_2^*, \ldots$ that result from interleaving the S_k and the $\hat{S}_k$ are uniformly computed by some machine. Let $\mathcal{M}$ be the extrapolator that on input of a n-bit string τ searches for the least k such that S_k^* has τ as an initial segment and then outputs $S_k^*(n+1)$ (there must be such a k because the $\hat{S}_j$ are dense). This is a total extrapolator that learns each of the S_j. □

In Fortnow *et al.* (1998) it was shown that if there is a total extrapolator that learns each member of a given class, then that class must be effectively meagre. That result can be strengthened.

Proposition 4 (Belot 2020). Let $\mathcal{A}$ be a class of computable sequences. If there is a total extrapolator $\mathcal{M}$ that better-than-random learns each sequence in $\mathcal{A}$, then $\mathcal{A}$ is effectively meagre.

Proof. The following is a computable winning strategy for Player 2 if the Banach-Mazur game is played for $\mathcal{A}$. On each round, concatenate the strings played so far to give a string σ. Then choose a string τ such that when $\mathcal{M}$ is fed the initial segments of $\sigma\tau$ as its data stream, its rate of successful predictions falls below $1/3$ (choose the first bit of τ to be the opposite of what $\mathcal{M}$ expects to see on input σ; in general, each bit of τ is chosen to be the opposite of what $\mathcal{M}$ expects to see given the bits of $\sigma\tau$ determined so far). □

Example 1 (Bārzdiņš 1971, Blum and Blum 1975). We call a sequence S *self-describing* if its first 1 is preceded by k 0's and it is computed by the Turing machine M_k with index k. It is easy to construct an extrapolator $\mathcal{M}_{SD}$ that learns each self-describing sequence: on input of an n-bit string σ, $\mathcal{M}_{SD}$ outputs 0 if σ does not contain any 1's; otherwise, $\mathcal{M}_{SD}$ finds the number k of 0's preceding the first 1 in σ and then attempts to use M_k to predict the next bit (i.e., $\mathcal{M}_{SD}$ runs M_k on input $n+1$ and if M_k outputs 0 or 1, $\mathcal{M}_{SD}$ gives the same output, otherwise $\mathcal{M}_{SD}$ gives no output). If fed a self-describing sequence as a data stream, $\mathcal{M}_{SD}$ will make a prediction on each input, making only one mistake (failng to predict the first 1 in the sequence). So $\mathcal{M}_{SD}$ learns each self-describing sequence. Note that $\mathcal{M}_{SD}$ is not a total machine: if k is the index of a Turing machine that does not halt on any input, then $\mathcal{M}_{SD}$ will not succeed in making any predictions after it sees the first 1 in a sequence that begins with k 0's followed by a 1.

Proposition 5 (Fortnow *et al.* 1998). The class of self-describing sequences is not effectively meagre.

Proof. Consider any computable strategy β that Player 2 might use for the Banach-Mazur game: such a strategy is a computable map that takes as input all the strings played so far and which gives as output the string to be played. We will show that β is not a winning strategy when the game is played for the set of self-describing sequences. For each k, consider the strategy α_k that Player 1 might adopt: in each round, play

k 0's followed by a 1. Let S_k be the sequence that results when Player 1 plays strategy α_k and Player 2 plays strategy β. The map $(k, \ell) \mapsto S_k(\ell)$ is computable so the Recursion Theorem guarantees that one of the S_k is self-describing. So there is no winning strategy for Player 2 for this version of the Banach-Mazur game. □

There is a large gap in learning power between total extrapolators and merely partial extrapolators. In particular, while some merely partial computable extrapolators weakly learn each computable sequence, no total extrapolator can even better-than-random learn each member of a class of sequences, unless that class is effectively meagre.

Remark 2. As we have seen, there is also a gap between total extrapolators and merely partial extrapolators when it comes to learning *simpliciter*: if a total extrapolator learns every sequence in a given class, that class must be effectively meagre—but Example 1 and Proposition 5 show that merely partial extrapolators are not restricted in this way. But there are of course many acceptable ways to assign Gödel numbers to Turing machines—and the extrapolator $\mathcal{M}_{SD}$ of Example 1 can be expected to perform well only if it sees sequences that wear on their sleeves that indices of machines computing them relative to the system of Gödel numbering that $\mathcal{M}_{SD}$ is built to exploit. Since many results concerning in this field exploit variants on this sort of phenomenon (see, e.g, the surveys Odifreddi (1999) and Zeugmann and Zilles (2008)), it is natural to ask to what extent the extra learning power of partial extrapolators depends on their ability to exploit coding tricks:.[30] For present purposes, it suffices to note that since there exist universal weak learning machines that successfully extrapolate each computable sequence, it is certainly not the case that all of the learning gaps between total and merely partial extrapolators involve the exploitation of coding tricks.

[30]This question has been extensively investigated for the problem of identification—see, e.g., Fulk (1990), Jain and Stephan (2003), and Case *et al.* (2004).

5. Learning by Enumeration

Let $\mathcal{U}$ be a uniformly computable family of sequences, $S_1, S_2, \ldots$ We denote by $\mathcal{E}_U$ the extrapolator that on input of any string τ, searches for the least k such that S_k begins with the bits of τ, and if it finds such a k, gives the next bit in S_k as output. We call an extrapolator of this form a *learner by enumeration.*

According to the following folklore result, learning by enumeration is a fully general strategy within the class of total extrapolators.

Proposition 6. Let $\mathcal{M}$ be a total extrapolator. Then there is a learner by enumeration $\mathcal{E}$ that gives the same output on every input as $\mathcal{M}$.

Proof. We slightly modify the construction of the first half of the proof of Proposition 3 above. We enumerate the strings $\sigma_1, \sigma_2, \ldots$ in order of increasing length; first comes the empty string; then the 1-bit string that $\mathcal{M}$ expects to see on empty input followed by the other 1-bit string; in general, at the $(k+1)$st stage, we consider the k-bit strings in the order already determined, in each case adding first the string that extends the given string by the bit $\mathcal{M}$ expects to see on that input, then the other 1-bit extension. For each $k > 0$, construct the sequence S_k by beginning with the bits of σ_k, then iteratively extending by repeatedly choosing the bit that $\mathcal{M}$ expects to see, given what it has seen so far. Then the $S_1, S_2, \ldots$ are uniformly computable and the learner by enumeration $\mathcal{E}$ based on the S_k gives the same output as $\mathcal{M}$ on every input. □

The learner by enumeration $\mathcal{E}$ based on the uniformly computable sequences $S_1, S_2, \ldots$ is total if and only if the S_k form a dense subclass of $\mathcal{C}$. A total learner by enumeration, like any totally defined extrapolator, can learn each sequence in a class only if that class is effectively countable and can weakly learn—or even better-than-random learn—each member in a class only if that class is effectively meagre. As we have seen, partially defined extrapolators can transcend these limits. So it is natural to ask to what extent learners by enumeration can reap the benefits of being merely partial.

If $\mathcal{E}$ is a (not necessarily total) learner by enumeration based on the uniformly computable sequences $S_1, S_2, \dots$, then $\mathcal{E}$ of course learns each of the S_k. And $\mathcal{E}$ does not learn any other sequences: if S is not one of the S_k, then either there is a longest initial segment of S that overlaps with one of the S_k (in which case $\mathcal{E}$ will give no output when fed yet longer initial segments of S) or there is no such longest initial segment (in which case $\mathcal{E}$ will make infinitely many false predictions when fed S as a data stream). It follows that any class of sequences each member of which can be learned by enumeration must be effectively countable. For learners by enumeration, being merely partially defined brings no dividend when it comes to learning *simpliciter*.

Proposition 7. If a learner by enumeration better-than-random learns each member of a dense class of computable sequences, then that class is effectively meagre.

Proof. Let $\mathcal{E}$ be a learner by enumeration. If $\mathcal{E}$ fails to give an output on input of some string σ, then it likewise fails to give an output input of on any extension of σ. So if $\mathcal{E}$ better-than-random learns each member of a dense class of computable sequences, then $\mathcal{E}$ must be total. $\square$

Corollary 2. No learner by enumeration is a universal better-than-random learning machine.

So learning by enumeration is not a fully general strategy for extrapolation: indeed, in light of the most recent corollary and the fact that there exist universal weak learning machines, it can be seen to be a highly limiting strategy. Further, the following example shows that there exists a meagre class of sequences such that no learner by enumeration can better-than-random learn each member of the class.[31]

Example 2. We call a string *legit* if it does't contain 100 as a substring. We call a computable sequence *legit* if all of its initial segments are legit. Let $\mathcal{A}$ be the class of computable sequences that are *either* self-describing and legit *or* consist of a legit initial segment, followed by all 0's. The following algorithm learns every sequence in $\mathcal{A}$: output 0 on every input—unless the input string begins with k 0's followed by a 1, with no consecutive 0's occurring after that first 1, in which case use the machine with index k to predict the next bit.

$\mathcal{A}$ is meagre (a winning strategy for Player 2 is to play 1001 on the first round) but not dense (no sequence in $\mathcal{A}$ begins with 1001). But no learner by enumeration better-than-random learns each sequence in $\mathcal{A}$. To see this, let $\mathcal{E}$ be a learner by enumeration and consider the partial computable function $\psi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined as follows: for any (k, ℓ), $\psi(k, \ell)$ (if defined) is the ℓth bit (if defined) arrived at by first listing k 0's followed by a 1, with subsequent bits determined by the rule that if the preceding bit is a 0, the next bit is 1 while if the preceding bit is 1 then the next bit is $(1 - b)$ where b is the output of $\mathcal{E}$ on input of the string constructed so far. The recursion theorem tells us that there is an e such that the machine M_e with index e computes the (possibly merely partial) function $\psi(e, \cdot)$. If M_e computes a sequence (as it will if $\mathcal{E}$ is total but may not otherwise) then that sequence is legit and self-describing but $\mathcal{E}$ cannot better-than-random learns it (.5 is an upper bound for $\mathcal{E}$'s limiting rate of correct predictions against this sequence). If M_e does not compute a sequence that is because although it outputs a bit for each number in some initial segment of the natural numbers, there is eventually a (least) natural number j on input of which it gives no output. Let σ be composed of the bits that M_e outputs on input of the natural numbers less that j. Then σ is legit and the result of extending σ by all 0's is a sequence in $\mathcal{A}$ that $\mathcal{E}$ does not learn (since $\mathcal{E}$ fails to give any output on σ or any extension of it).

6. Bayesian Learning

6.1 Motivation

As noted in Section 1 above, if we abstract away from computational considerations, it is easy to design learners by enumeration that are guaranteed to succeed at our extrapolation problem: a learner who lists the hypotheses in order of decreasing *a priori* plausibility (or in any

[31]For another example, see the construction in the proof of Lemma 2 in Podnieks (1975).

other order), then at each stage selects the first hypothesis consistent with the data and uses it it predict the next data point will eventually end up correctly predicting each new data point. And, in this setting, Bayesian learners who put positive prior probability on each hypothesis are likewise guaranteed of success.

But, as we have just seen, the situation for learners by enumeration is quite different when computational considerations are taken into account: there are classes of hypotheses each of which can be learned by some computable agent, but which cannot all be learned by any computable agent who learns by enumeration. We will see that the story is interestingly different for computable Bayesians. There is a sense in which computable Bayesian learners can successfully handle any extrapolation problem that can be handled by any computable learner. But in order to handle problems that cannot be handled by learning by enumeration, Bayesians must employ priors that, in a certain sense, fail radically to exhibit the sort of confidence in their predictions that one normally expects from Bayesian learners—and which will in some situations ignore known background information.

6.2 *Framework*

Recall that for any binary string τ, we write $[\![\tau]\!]$ for the basic set of binary sequences whose initial bits are given by τ.

Definition 4. A *probability measure* on 2^ω is a function μ from basic sets to the closed unit interval such that $\mu([\![\varnothing]\!]) = 1$ and for any string τ we have $\mu([\![\tau]\!]) = \mu([\![\tau 0]\!]) + \mu([\![\tau 1]\!])$.[32]

Henceforth, for any probability measure μ, string τ, and bit b we will just write $\mu(\tau)$ instead of $\mu([\![\tau]\!])$, $\mu(b \mid \tau)$ instead of $\mu([\![\tau b]\!] \mid [\![\tau]\!])$, and so on.

Example 3. For any $r \in (0,1)$, the *Bernoulli measure with parameter* r is the probability measure ν_r such that $\nu_r(\tau) = r^k(1-r)^\ell$ where k and ℓ are the number of 0's and the number of 1's in τ. We call $\nu_{.5}$ the *Lebesgue measure*.

Example 4. Let S be a sequence. The *delta-function concentrated on* S is the probability measure δ_S that assigns a basic set $[\![\tau]\!]$ probability one if τ is an initial segment of S and which otherwise assigns $[\![\tau]\!]$ probability zero.

Definition 5. A real number z in the unit interval is *computable* if there is a Turing machine that on input of any natural number k outputs a rational number q that differs from z by no more than 2^{-k}.

Definition 6. A probability measure μ on 2^ω is *computable* if there is a Turing machine that on input of any string τ and natural number k outputs a rational number q that differs from $\mu(\tau)$ by no more than 2^{-k}.

A delta-function δ_S is computable if and only if the sequence S that it is concentrated on is. A Bernoulli measure ν_r is computable if and only if its parameter r is.

Definition 7. We call a probability measure μ on 2^ω *discrete* if there exist sequences $S_1, S_2, \ldots$ and positive numbers $c_1, c_2, \ldots$ summing to one such that $\mu = \sum c_k \cdot \delta_{S_k}$.

Example 5. For any sequences $S_1, S_2, \ldots, \mu := \sum_{k=1}^{\infty} \frac{9}{10^k} \cdot \delta_{S_k}$ will be a discrete probability measure (and if the S_k are uniformly computable, then μ will be computable).[33] No Bernoulli measure is discrete.

We are interested in Bayesian agents who know that their binary data streams are computable but have no other relevant background knowledge about them. It is natural to assume that any agent of this

[32]Equivalently, we could define a Borel probability measure on 2^ω to be a countably additive set function $\tilde{\mu}$ from the σ-algebra generated by the basic sets to the closed unit interval such that $\tilde{\mu}(2^\omega) = 1$. The restriction of any such map to the basic sets yields a probability measure as defined in the main text. And the Carathéodory Extension Theorem guarantees that every μ satisfying the conditions in the main text can be uniquely extended to such a Borel probability measure $\tilde{\mu}$ on 2^ω—see, e.g., Klenke (2020, Theorem 1.53).

[33]If a discrete probability measure $\mu = \sum c_k \cdot \delta_{S_k}$ is computable, then each S_k is computable. But it is possible that the S_k may not be uniformly computable—thanks to Chris Porter for this observation.

kind will assign positive probability to each basic set.

Definition 8. A *prior* is a probability measure on 2^ω that assigns positive probability to each basic set.

If μ is a prior and S is a sequence, then for each n, $\mu(0 \mid S\restriction n)$ is well-defined (it is the probability that an agent whose initial credal state is represented by μ will assign to the next bit being a 0, given that the first n bits are given by the initial bits of S).

6.3 *Confident Bayesian Learning*

Discussions of the strengths of the Bayesian account of rationality tend to focus on success stories like the problem of determining the bias of a coin from knowledge of the outcomes of a sequence of tosses.[34] And with good reason: as Savage (1972, p. 50) notes in summarizing his treatment of problems of this kind,

> it has now been shown that, with the observation of an abundance of relevant data, the person is almost certain to become highly convinced of the truth, and it has also been shown that he himself knows this to be the case.

With examples like this in mind, it may well seem natural to take as the criterion of success for Bayesian learning the asymptotic approach to certainty in the truth in the limit of large data sets.[35]

Definition 9. Let μ be a prior and S a sequence. We say that μ *confidently learns* S if $\lim_{k\to\infty} \mu(S(k+1) \mid S\restriction k) = 1$.

Example 6. As in Example 5, consider countably many sequences S_1, $S_2, \ldots$ and let $\mu := \sum(9/10^k) \cdot \delta_{S_k}$. Then μ confidently learns each S_j: for suppose that μ is fed S_j as a data stream; after a finite number of data points, for each $k < j$, S_k will be ruled out by observation; from then on, the prior will always assign probability between .9 and 1 to its predictions (which will always be correct); and you can force these probabilities to be as close to 1 as you like by choosing data sets large enough to rule out a sufficient number of S_{j+1}, S_{j+2}, and so on.[36]

Of course, things are not so easy when computational considerations are taken into account. In order to see why, it helps to introduce a further technical notion and a lemma.

Definition 10. Let μ be a computable prior and let κ be a computable real number in the unit interval. We say that μ *avoids* κ if there is no string τ such that $\mu(0 \mid \tau) = \kappa$.

Proposition 8 (Belot 2024). Let μ be a computable prior and let I be a non-empty open subinterval of $[0, 1]$. Then there is a computable real number $\kappa \in I$ such that μ avoids κ.

Proof. Fix a computable enumeration $\sigma_1, \sigma_2, \ldots$ of the strings. Then we can use a machine computing μ to build a machine that on input of natural numbers k and ℓ gives us a rational number that approximates $\mu(0 \mid \sigma_\ell)$ to within $1/2^k$. We can then take κ to be a real number with a decimal expansion of the following type: chose the initial digits to ensure that $\kappa \in I$ no matter how the expansion continues; choose the next digit by looking at the corresponding digit in the $\mu(0 \mid \sigma_1)$ and adding two (mod 10); choose the following digit by looking at the corresponding digit in $\mu(0 \mid \sigma_2)$ and adding two (mod 10); and so on. Then κ is a computable number in I that is never used by μ as the conditional probability that the next bit will be 0. $\square$

Definition 11. Let μ be a prior and let κ be a computable number in the

[34] See, e.g., Savage (1972, Chapter 3), Earman (1992, §6.3), and Diaconis and Skyrms (2018, Chapter 10). In philosophy of science more generally, the Bayesian treatment of this problem is often taken to provide a good model of the strengths and limits of a Bayesian account of scientific reasoning—see, e.g., Strevens (2020, p. 305).

[35] Earman (1992, §9.3) could be read as taking this approach.

[36] As noted in fn. 32 above, any prior μ induces a Borel probability measure $\tilde{\mu}$ on 2^ω—so we can ask what probability $\tilde{\mu}$ assigns to the set $\mathcal{S}_\mu$ of sequences that μ confidently learns. For the μ of this example, we find that $\tilde{\mu}(\mathcal{S}_\mu) = 1$. On the other hand, for any $r \in (0, 1)$ the Bernoulli measure ν_r fails to confidently learn any sequences, so we have $\tilde{\nu}_r(\mathcal{S}_{\nu_r}) = 0$—this is *not* one of those settings where each Bayesian agent is subjectively certain of successful learning.

unit interval. We denote by $\mathcal{M}_\mu^\kappa$ the extrapolator that on input σ: (i) outputs 0 if $\mu(0 \mid \sigma) > \kappa$, (ii) outputs 1 if $\mu(0 \mid \sigma) < \kappa$, and (iii) gives no output when $\mu(0 \mid \sigma) = \kappa$.

Note that if μ avoids κ, then $\mathcal{M}_\mu^\kappa$ is a total extrapolator.

Proposition 9. Let $\mathcal{A}$ be a class of computable sequences. There is a computable prior μ that confidently learns each sequence in $\mathcal{A}$ if and only if there is a total extrapolator that learns each sequence in $\mathcal{A}$.

Proof. Suppose, on the one hand, that there is a total extrapolator that learns each sequence in $\mathcal{A}$. Then the proof of Proposition 3 guarantees that there is some uniformly computable sequences $S_1, S_2, \ldots$ that form a dense set that includes every sequence in $\mathcal{A}$. Then as noted above in Example 5, the prior $\mu := \sum(9/10^k) \cdot \delta_{S_k}$ is computable (because the S_k are uniformly computable). And as noted in Example 6, μ confidently learns each of the S_k and hence confidently learns each sequence in $\mathcal{A}$. Suppose, on the other hand, that μ is a computable prior that confidently learns each sequence in $\mathcal{A}$. Then as in Proposition 8, we can choose a computable κ in $(.5, 1)$ that μ avoids. So $\mathcal{M}_\mu^\kappa$ is a total extrapolator that learns each sequence confidently learned by μ: if for any small ε, there is an N such that for each $n > N$, $\mu(S(n+1) \mid S{\upharpoonright}n) > 1 - \varepsilon$, then, in particular, there is an N_κ such that for all $n > N_\kappa$, $\mathcal{M}_\mu^\kappa$ predicts the next bit correctly when fed S as a data stream from stage N_κ onwards. □

Corollary 3. If a class of computable sequences is confidently learned by some prior, then it is effectively countable.

Definition 12. We say that *μ confidently learns S with redaction rate less than .5* if there is a set R of natural numbers of density $r < .5$ such that $\lim_{k \to \infty} \mu(S(k+1) \mid S{\upharpoonright}k) = 1$ where the limit is taken over k in the complement of R.

Arguing as in the proof of the preceding proposition, we can show that if each member of a class of computable sequences is confidently learned with redaction rate less than .5 by some computable prior μ, then there is some total extrapolator that better-than-random learns each of these sequences. By appeal to Propositions 4 we then have:

Proposition 10. If $\mathcal{A}$ is a class of computable sequences and μ is a computable prior that confidently learns each sequence in $\mathcal{A}$ with redaction rate less than .5, then $\mathcal{A}$ is effectively meagre.

6.4 *Diffident Bayesian Learning*

Definition 13. Let μ be a prior and let S be a sequence. We say that μ *diffidently (weakly) learns S* if the extrapolator $\mathcal{M}_\mu^{.5}$ (weakly) learns S.

There is a sense in which every extrapolator is Bayesian in behaviour.

Proposition 11. Let $\mathcal{M}$ be any extrapolator. Then there is a computable prior μ such $\mathcal{M}$ and $\mathcal{M}_\mu^{.5}$ are functionally equivalent (i.e., on any input, either neither gives an output or they both give the same output).

Proof. We show how to construct the desired prior μ given an extrapolator $\mathcal{M}$. We need to define a procedure that allows us to calculate $\mu(\sigma)$ to any degree of accuracy, for each string σ. Since the set of computable numbers is closed under addition, subtraction, multiplication, and division, it suffices to show how to calculate $\mu(0 \mid \sigma)$ to any desired degree of accuracy, for each string σ: setting σ equal to the empty string allows us to calculate $\mu(0)$ and thence $\mu(1)$; with these in hand, if we know how to calculate $\mu(0 \mid 1)$ and $\mu(0 \mid 0)$, we will then be able to calculate $\mu(00)$, $\mu(01)$, $\mu(10)$, and $\mu(11)$—and so on.[37]

Fix a string σ. Set the first approximation of $\mu(0 \mid \sigma)$ equal to $1/2$. The second approximation takes the same value—unless the computation of $\mathcal{M}$ on input σ halts and gives an output after one stage of computation, in which case we set the second approximation equal to $1/2 + 1/100$ or equal to $1/2 - 1/100$, depending on whether the output of the computation was a 0 or a 1. In general: the $(k+1)$st approximation to $\mu(0 \mid \sigma)$ is the same as the kth, unless the computation of $\mathcal{M}$ on input σ first halts and gives an output after k stages of computation, in which case we set the $(k+1)$st approximation equal to $1/2 + 1/10^{k+1}$ or $1/2 - 1/10^{k+1}$, depending on whether the output of the computation was a 0 or a 1.

[37]We will have $\mu(\sigma) > 0$ for each σ, so next-bit conditional probabilities will always be defined.

This ensures that $\mu(0 \mid \sigma) > .5$ if $\mathcal{M}(\sigma) = 0$, that $\mu(0 \mid \sigma) < .5$ if $\mathcal{M}(\sigma) = 1$, and that $\mu(0 \mid \sigma) = .5$ if $\mathcal{M}(\sigma)$ is undefined. So $\mathcal{M}$ and $\mathcal{M}_\mu^{.5}$ give the same output on any input for which either gives an output. □

Even among merely partial computable extrapolators, then, Bayesian learning is a fully general strategy: if our standard for Bayesian learning is is diffident learning (rather than confident learning) then for computable learners facing our extrapolation problem, Bayesianism (unlike learning by enumeration) does not restrict learning power. Indeed, there are priors that diffidently learn each self-describing sequence and priors that diffidently weakly learn *every* computable sequence (although of course the extrapolator $\mathcal{M}_\mu^{.5}$ corresponding to such a prior cannot be total).

It is hard not to feel, though, that the construction employed in the proof just given yielded a prior that was a little ... peculiar. For one thing, conditional on *any* finite data set, this prior has credence very near .5 that the next bit will be 0. Even if the agent eventually always chooses correctly, we will not have the sort of satisfying Bayesian learning that we are used to. The following example suggests, however, that we cannot hope to do too much better.

Example 7. Let μ be a prior that for each string, says that the probability of the next bit being 0, conditional on that string, is .9, .5, or .1. This prior will induce an (in general) merely partially defined extrapolator $\mathcal{M}_\mu^{.5}$. But this extrapolator will not be able to learn any sequences that are not learned flat-out by the totally defined extrapolator $\mathcal{M}_\mu^{.6}$. So any class of computable sequences diffidently learned by μ must be effectively countable.

More generally, let μ be any computable prior. Let κ a computable number that μ avoids (as in Proposition 8), so that $\mathcal{M}_\mu^{\kappa}$ is a total extrapolator. It follows that if $\mathcal{M}_\mu^{.5}$ and $\mathcal{M}_\mu^{\kappa}$ both learn every sequence in some class $\mathcal{A}$ of computable sequences, $\mathcal{A}$ must be effectively countable. Let S be a computable sequence learned by $\mathcal{M}_\mu^{.5}$ but not by $\mathcal{M}_\mu^{\kappa}$. Then there must be infinitely many initial segments of S that lead μ to give a next-bit probability of 0 in the interval $(.5, \kappa)$ (no matter small this interval). And of course a similar picture obtains if we take $\kappa < .5$. So we have the following.

Proposition 12. Let μ be a computable prior, let $\varepsilon > 0$, and let $\mathcal{A}$ be a class of computable sequences. Suppose that for any S in $\mathcal{A}$, μ diffidently learns S and there are only finitely many n such that $\mu(0 \mid S{\restriction}n) \in (.5 - \varepsilon, .5 + \varepsilon)$. Then $\mathcal{A}$ is effectively countable.

Recall from Example 5 above that we call a prior discrete if it can be written as a convex combination of a countable family of delta-functions. More generally, we call a prior μ *atomic* if it can be written in the form $\mu = (1 - c) \cdot \nu + c \cdot \delta_S$ for some probability measure ν, delta-function δ_S, and $c \in (0, 1]$ (we then call S an *atom* of μ). Note that if S is an atom of μ, then $\lim_{n \to \infty} \mu(S(n+1) \mid S{\restriction}n) = 1$.[38] The prior that appeared in the proof of Proposition 11 is clearly not atomic, then. And again we cannot hope to do too much better.

Proposition 13. If a discrete prior diffidently learns each sequence in a class $\mathcal{A}$ of computable sequences, then $\mathcal{A}$ is effectively meagre.

Proof. Suppose that μ is a discrete prior with atoms $S_1, S_2, \ldots$. Then the S_k must be dense in 2^ω (otherwise, μ would not assign each basic set positive probability). And, for the reason just noted, $\mathcal{M}_\mu^{.5}$ learns each S_k. So by Proposition 2, the class of sequences learned by $\mathcal{M}_\mu^{.5}$ (= the class of sequences diffidently learned by μ) must be effectively meagre. □

Corollary 4. A computable prior that diffidently learns each self-describing sequence cannot put probability one on the set of computable sequences.

Bayesian learners who know that they will face computable data streams face a dilemma. They can work with priors that incorporate this knowledge, putting probability one on the set of computable sequences. But each such prior is incapable of achieving many learning tasks

[38] This follows from the fact that Borel probability measures are continuous from above—see, e.g., Klenke (2020, Theorem 1.36).

achievable by computable learners. Or they can use priors capable of more impressive feats of learning. But such priors cannot put probability one on any countable set of sequences—so, in particular, each such prior fails to incorporate the agents' knowledge that they face a computable data stream.

7. The Bottom Line

	Learning	Weak Learning	Better-than-Random
Extrapolators	Universal: No Supra-Meagre: Yes	Universal: Yes Supra-Meagre: Yes	Universal: Yes Supra-Meagre: Yes
Learners by Enumeration	Universal: No Supra-Meagre: No	Universal: No Supra-Meagre: ?	Universal: No Supra-Meagre: ?
Total Extrapolators	Universal: No Supra-Meagre: No	Universal: No Supra-Meagre: No	Universal: No Supra-Meagre: No

Table 1: The learning capacities and limitations of extrapolators.

	Confident Bayesian Learning	Diffident Bayesian Learning	Diffident Weak Bayesian Learning
Computable Priors	Universal: No Supra-Meagre: No	Universal: No Supra-Meagre: Yes	Universal: Yes Supra-Meagre: Yes
Discrete Comp. Priors	Universal: No Supra-Meagre: No	Universal: No Supra-Meagre: No	Universal: ? Supra-Meagre: ?

Table 2: The learning capacities and limitations of computable Bayesians.

The tables above summarize the learning powers and limits of various types of extrapolators and computable priors, focusing for simplicity on the questions under which conditions there exist universal learners that extrapolate every computable sequence and under which conditions there exist supra-meagre learners, capable of successfully extrapolating every sequence in at least one class that is not effectively meagre. The question marks in these tables indicate open questions—it is natural to conjecture each of these should be given a negative answer.

We see, in particular that totally defined extrapolators, which give an output on input of any binary string, are of modest learning power in comparison with arbitrary extrapolators (which may be merely partially defined). We also see that at least some of the limitations of totally defined extrapolators extend to learners by enumeration (a class that includes those learners who use the simplest hypothesis consistent with the data to make their predictions).

The picture is interestingly different for computable Bayesian learners. If we require successful Bayesian learners to eventually become arbitrarily confident in their predictions about future data points, then Bayesian learners are saddled with the same limitations of learning power as total extrapolators are. But if we require of successful Bayesian learners only that they be more confident than not in their predictions, then Bayesianism is a fully general strategy for extrapolation: if there is an extrapolator that (weakly) learns each member of some class of computable sequences, then there is a computable Bayesian who does so. But in order to transcend the limits faced by total extrapolators, computable Bayesian agents must consign themselves to being occasionally very nearly uncertain about what to expect next (no matter how much data they see)—and they must employ priors that do not put probability one on any countable set of sequences (even when they know that their data stream will be drawn from the countably infinite family of computable sequences).

References

Ackerman, N. L., C. E. Freer, and D. M. Roy (2011) "Noncomputable Conditional Distributions." In *26th Annual IEEE Symposium on Logic in Computer Science*. Washington DC: IEEE Computer Society 107–116.

Ambos-Spies, K. and J. Reimann (1998) "Effective Baire Category Concepts." In C. T. Chong, Q. Feng, D. C. Ding, Q. Y. Huang, and M. Yasugi (eds), *Proceedings Of The Sixth Asian Logic Conference*. Singapore: World Scientific 13–29.

Angluin, D. and C. Smith (1983) "Inductive Inference: Theory and Methods." *ACM Computing Surveys* 15 237–269.

Bārzdiņš, J. (1971) *Complexity and Frequency Solution of Some Algorithmically Unsolvable Problems*. Novosibirsk State University doctoral dissertation. In Russian.

Bārzdiņš, J. (1972) "Prognostication of Automata and Functions." In C. V. Frieman (ed), *Information Processing '71*. Volume 1. Amsterdam: North-Holland 81–84.

Bārzdiņš, J. and R. Freivalds (1972) "On the Prediction of General Recursive Functions." *Soviet Mathematics Doklady* 13 1224–1228.

Belot, G. (2020) "Absolutely No Free Lunches!" *Theoretical Computer Science* 845 159–180.

Belot, G. (2024) "Unprincipled." *The Review of Symbolic Logic* 17 435–474.

Blum, L. and M. Blum (1975) "Toward a Mathematical Theory of Inductive Inference." *Information and Control* 28 125–155.

Blum, M. (1967) "A Machine-Independent Theory of the Complexity of Recursive Functions." *Journal of the Association for Computing Machinery* 14 322–336.

Case, J., S. Jain, F. Stephan, and R. Wiehagen (2004) "Robust Learning—Rich and Poor." *Journal of Computer and System Sciences* 69 123–165.

Case, J. and C. Smith (1983) "Comparison of Identification Criteria for Machine Inductive Inference." *Theoretical Computer Science* 25 193–220.

Diaconis, P. and B. Skyrms (2018) *Ten Great Ideas about Chance*. Princeton NJ: Princeton University Press.

Downey, R. and D. Hirschfeldt (2010) *Algorithmic Randomness and Complexity*. New York: Springer.

Earman, J. (1992) *Bayes or Bust? A Critical Examination of Bayeisan Confirmation Theory*. Cambridge MA: MIT Press.

Fortnow, L., R. Freivalds, W. Gasarch, M. Kummer, S. Kurtz, C. Smith, and F. Stephan (1998) "On the Relative Sizes of Learnable Sets." *Theoretical Computer Science* 197 139–156.

Fulk, M. (1990) "Robust Separations in Inductive Inference." In *SFCS '90: Proceedings of the 31st Annual Symposium on Foundations of Computer Science*. Washington DC: IEEE Computer Society 405–410.

Gold, E. M. (1967) "Language Identification in the Limit." *Information and Control* 10 447–474.

Jain, S., D. Osherson, J. Royer, and A. Sharma (1999) *Systems that Learn: An Introduction to Learning Theory* (second edition). Cambridge MA: MIT press.

Jain, S. and F. Stephan (2003) "A Tour of Robust Learning." In *Computability and Models: Perspectives East and West.* Dordrecht: Kluwer 315–247.

Juhl, C. (1993) "Bayesianism and Reliable Scientific Inquiry." *Philosophy of Science* 60 302–319.

Kelly, K. (1996) *The Logic of Reliable Inquiry*. Oxford: Oxford University Press.

Klenke, A. (2020) *Probability Theory: A Comprehensive Course* (third edition). New York: Springer.

Kugel, P. (1977) "Induction, Pure and Simple." *Information and Control* 35 276–336.

Lisagor, L. (1981) "The Banach-Mazur Game." *Mathematics of the USSR-Sbornik* 38 201–216.

Mehlhorn, K. (1973) "On the Size of Sets of Computable Functions." In R. Book, A. Borodin, F. Lewis, A. Mukhopadhyay, A. Rosenberg, R. Strong, and J. Ullman (eds), *14th Annual Symposium on Switching and Automata Theory*. Washington DC: IEEE Computer Society 190–196.

Odifreddi, P. (1989) *Classical Recursion Theory*. Amsterdam: Elsevier.

Odifreddi, P. (1999) *Classical Recursion Theory* II. Amsterdam: Elsevier.

Osherson, D., M. Stob, and S. Weinstein (1988) "Mechanical Learners Pay a Price for Bayesianism." *The Journal of Symbolic Logic* 53 1245–1251.

Oxtoby, J. (1957) "The Banach-Mazur Game and Banach Category Theorem." In M. Dresher, A. Tucker, and P. Wolfe (eds), *Contributions to the Theory of Games*. Volume III. Princeton NJ: Princeton University Press 159–163.

Oxtoby, J. (1980) *Measure and Category: A Survey of the Analogies between Topological and Measure Spaces* (second edition). New York: Springer.

Podnieks, K. (1974) “Comparing Various Concepts of Function Prediction. I.” *Scientific Proceedings of Latvia State University* 210 68–81. In Russian.

Podnieks, K. (1975) “Comparing Various Concepts of Function Prediction. II.” *Scientific Proceedings of Latvia State University* 233 35–44. In Russian.

Putnam, H. (1963) *Probability and Confirmation*. Washington DC: US Information Agency.

Savage, L. (1972) *The Foundations of Statistics* (second edition). New York: Dover.

Schnorr, C.-P. (1971a) “A Unified Approach to the Definition of Random Sequences.” *Mathematical Systems Theory* 5 246–258.

Schnorr, C.-P. (1971b) *Zufälligkeit und Wahrscheinlichkeit: Eine algorithmische Begründung der Wahrscheinlichkeitstheorie*. New York: Springer.

Shen, A., V. A. Uspensky, and N. Vereshchagin (2017) *Kolmogorov Complexity and Algorithmic Randomness*. Providence RI: American Mathematical Society.

Soare, R. (2016) *Turing Computability: Theory and Applications*. New York: Springer.

Sterkenburg, T. (2019) “Putnam’s Diagonal Argument and the Impossibility of a Universal Learning Machine.” *Erkenntnis* 84 633–656.

Strevens, M. (2020) *The Knowledge Machine: How Irrationality Created Modern Science*. New York: Liveright Publishing.

Turing, A. (1936) “On Computable Numbers, with an Application to the Entscheidungsproblem.” *Proceedings of the London Mathematical Society* 42 230–265.

Ville, J. (1939) *Étude critique de la notion de collectif*. Paris: Gauthier-Villars.

Wiehagen, R. (1976) “Limes-Erkennung rekursiver Funktionen durch spezielle Strategien.” *Elektronische Informationsverarbeitung und Kybernetik* 12 93–99.

Zeugmann, T. and S. Zilles (2008) “Learning Recursive Functions: A Survey.” *Theoretical Computer Science* 397 4–56.