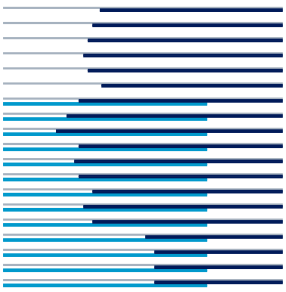
Is there an Elegant Universal Theory of Prediction?

Shane Legg



Technical Report No. IDSIA-12-06
October 19, 2006

IDSIA / USI-SUPSI
Dalle Molle Institute for Artificial Intelligence
Galleria 2, 6928 Manno, Switzerland

Is there an Elegant Universal Theory of Prediction? *

Shane Legg†

October 19, 2006

Abstract

Solomonoff's inductive learning model is a powerful, universal and highly elegant theory of sequence prediction. Its critical flaw is that it is incomputable and thus cannot be used in practice. It is sometimes suggested that it may still be useful to help guide the development of very general and powerful theories of prediction which are computable. In this paper it is shown that although powerful algorithms exist, they are necessarily highly complex. This alone makes their theoretical analysis problematic, however it is further shown that beyond a moderate level of complexity the analysis runs into the deeper problem of Gödel incompleteness. This limits the power of mathematics to analyse and study prediction algorithms, and indeed intelligent systems in general.

1 Introduction

Solomonoff's model of induction rapidly learns to make optimal predictions for any computable sequence, including probabilistic ones [13, 14]. It neatly brings together the philosophical principles of Occam's razor, Epicurus' principle of multiple explanations, Bayes theorem and Turing's model of universal computation into a theoretical sequence predictor with astonishingly powerful properties. Indeed the problem of sequence prediction could well be considered solved [9, 8], if it were not for the fact that Solomonoff's theoretical model is incomputable.

Among computable theories there exist powerful general predictors, such as the Lempel-Ziv algorithm [5] and Context Tree Weighting [18], that can learn to predict some complex sequences, but not others. Some prediction methods, based on the Minimum Description Length principle [12] or the Minimum Message Length principle [17], can even be viewed as computable approximations of Solomonoff induction [10]. However in practice their power and generality are limited by the power of the compression methods employed, as well as having a significantly reduced data efficiency as compared to Solomonoff induction [11].

Could there exist elegant computable prediction algorithms that are in some sense universal? Unfortunately this is impossible, as pointed out by Dawid [4]. Specifically, he notes that for any statistical forecasting system there exist sequences which are not calibrated. Dawid also notes that a forecasting system for a family of distributions is necessarily more complex than any forecasting

*This work was supported by SNF grant 200020-107616.

†shane@idsia.ch

system generated from a single distribution in the family. However, he does not deal with the complexity of the sequences themselves, nor does he make a precise statement in terms of a specific measure of complexity, such as Kolmogorov complexity. The impossibility of forecasting has since been developed in considerably more depth by V'yugin [16], in particular he proves that there is an efficient randomised procedure producing sequences that cannot be predicted (with high probability) by computable forecasting systems.

In this paper we study the prediction of computable sequences from the perspective of Kolmogorov complexity. The central question we look at is the prediction of sequences which have bounded Kolmogorov complexity. This leads us to a new notion of complexity: rather than the length of the shortest program able to generate a given sequence, in other words Kolmogorov complexity, we take the length of the shortest program able to learn to predict the sequence. This new complexity measure has the same fundamental invariance property as Kolmogorov complexity, and a number of strong relationships between the two measures are proven. However in general the two may diverge significantly. For example, although a long random string that indefinitely repeats has a very high Kolmogorov complex, this sequence also has a relatively simple structure that even a simple predictor can learn to predict.

We then prove that some sequences, however, can only be predicted by very complex predictors. This implies that very general prediction algorithms, in particular those that can learn to predict all sequences up to a given Kolmogorov complex, must themselves be complex. This puts an end to our hope of there being an extremely general and yet relatively simple prediction algorithm. We then use this fact to prove that although very powerful prediction algorithms exist, they cannot be mathematically discovered due to Gödel incompleteness. Given how fundamental prediction is to intelligence, this result implies that beyond a moderate level of complexity the development of powerful artificial intelligence algorithms can only be an experimental science.

2 Preliminaries

An *alphabet* $\mathcal{A}$ is a finite set of 2 or more elements which are called *symbols*. In this paper we will assume a binary alphabet $\mathbb{B} := \{0, 1\}$, though all the results can easily be generalised to other alphabets. A *string* is a finite ordered n-tuple of symbols denoted $x := x_1 x_2 \dots x_n$ where $\forall i \in \{1, \dots, n\}$, $x_i \in \mathbb{B}$, or more succinctly, $x \in \mathbb{B}^n$. The 0-tuple is denoted λ and is called the *null string*. The expression $\mathbb{B}^{\leq n}$ has the obvious interpretation, and $\mathbb{B}^* := \bigcup_{n \in \mathbb{N}} \mathbb{B}^n$. The length *lexicographical* ordering is a total order on $\mathbb{B}^*$ defined as $\lambda < 0 < 1 < 00 < 01 < 10 < 11 < 000 < 001 < \cdots$. A *substring* of x is defined $x_{j:k} := x_j x_{j+1} \dots x_k$ where $1 \leq j \leq k \leq n$. By $|x|$ we mean the length of the string x, for example, $|x_{j:k}| = k - j + 1$. We will sometimes need to encode a natural number as a string. Using simple encoding techniques it can be shown that there exists a computable injective function $f : \mathbb{N} \to \mathbb{B}^*$ where no string in the range of f is a prefix of any other, and $\forall n \in \mathbb{N} : |f(n)| \leq \log_2 n + 2 \log_2 \log_2 n + 1 = O(\log n)$.

Unlike strings which always have finite length, a *sequence* ω is an infinite list of symbols $x_1 x_2 x_3 \dots \in \mathbb{B}^\infty$. Of particular interest to us will be the class of sequences which can be generated by an algorithm executed on a universal Turing machine:

2.1 Definition. A **monotone universal Turing machine** $\mathcal{U}$ is defined as a universal Turing machine with one unidirectional input tape, one unidirectional output tape, and some bidirectional work tapes. Input tapes are read only, output tapes are write only, unidirectional tapes are those

where the head can only move from left to right. All tapes are binary (no blank symbol) and the work tapes are initially filled with zeros. We say that $\mathcal{U}$ outputs/computes a sequence ω on input p, and write $\mathcal{U}(p) = \omega$, if $\mathcal{U}$ reads all of p but no more as it continues to write ω to the output tape.

We fix $\mathcal{U}$ and define $\mathcal{U}(p, x)$ by simply using a standard coding technique to encode a program p along with a string $x \in \mathbb{B}^*$ as a single input string for $\mathcal{U}$.

2.2 Definition. A sequence $\omega \in \mathbb{B}^\infty$ is a **computable binary sequence** if there exists a program $q \in \mathbb{B}^*$ that writes ω to a one-way output tape when run on a monotone universal Turing machine $\mathcal{U}$, that is, $\exists q \in \mathbb{B}^* : \mathcal{U}(q) = \omega$. We denote the set of all computable sequences by $\mathcal{C}$.

A similar definition for strings is not necessary as all strings have finite length and are therefore trivially computable.

2.3 Definition. A **computable binary predictor** is a program $p \in \mathbb{B}^*$ that on a universal Turing machine $\mathcal{U}$ computes a total function $\mathbb{B}^* \to \mathbb{B}$.

For simplicity of notation we will often write $p(x)$ to mean the function computed by the program p when executed on $\mathcal{U}$ along with the input string x, that is, $p(x)$ is short hand for $\mathcal{U}(p, x)$. Having $x_{1:n}$ as input, the objective of a predictor is for its output, called its *prediction*, to match the next symbol in the sequence. Formally we express this by writing $p(x_{1:n}) = x_{n+1}$.

As the algorithmic prediction of incomputable sequences, such as the halting sequence, is impossible by definition, we only consider the problem of predicting computable sequences. To simplify things we will assume that the predictor has an unlimited supply of computation time and storage. We will also make the assumption that the predictor has unlimited data to learn from, that is, we are only concerned with whether or not a predictor can learn to predict in the following sense:

2.4 Definition. We say that a predictor p can **learn to predict** a sequence $\omega := x_1 x_2 \ldots \in \mathbb{B}^\infty$ if there exists $m \in \mathbb{N}$ such that $\forall n \geq m : p(x_{1:n}) = x_{n+1}$.

The existence of m in the above definition need not be constructive, that is, we might not know when the predictor will stop making prediction errors for a given sequence, just that this will occur eventually. This is essentially "next value" prediction as characterised by Barzdin [1], which follows from Gold's notion of identifiability in the limit for languages [7].

2.5 Definition. Let $P(\omega)$ be the set of all predictors able to learn to predict ω. Similarly for sets of sequences $S \subset \mathbb{B}^\infty$, define $P(S) := \bigcap_{\omega \in S} P(\omega)$.

A standard measure of complexity for sequences is the length of the shortest program which generates the sequence:

2.6 Definition. For any sequence $\omega \in \mathbb{B}^\infty$ the monotone **Kolmogorov complexity** of the sequence is,

$$K(\omega) := \min_{q \in \mathbb{B}^*} \{|q| : \mathcal{U}(q) = \omega\},$$

where $\mathcal{U}$ is a monotone universal Turing machine. If no such q exists, we define $K(\omega) := \infty$.

It can be shown that this measure of complexity depends on our choice of universal Turing machine $\mathcal{U}$, but only up to an additive constant that is independent of ω. This is due to the fact that a universal Turing machine can simulate any other universal Turing machine with a fixed length program.

In essentially the same way as the definition above we can define the Kolmogorov complexity of a string $x \in \mathbb{B}^n$, written $K(x)$, by requiring that $\mathcal{U}(q)$ halts after generating x on the output tape. For an extensive treatment of Kolmogorov complexity and some of its applications see [10] or [2].

As many of our results will have the above property of holding within an additive constant that is independent of the variables in the expression, we will indicate this by placing a small plus above the equality or inequality symbol. For example, $f(x) \stackrel{+}{<} g(x)$ means that that $\exists c \in \mathbb{R}, \forall x : f(x) < g(x) + c$. When using standard "Big O" notation this is unnecessary as expressions are already understood to hold within an independent constant, however for consistency of notation we will use it in these cases also.

3 Prediction of computable sequences

The most elementary result is that every computable sequence can be predicted by at least one predictor, and that this predictor need not be significantly more complex than the sequence to be predicted.

3.1 Lemma. $\forall \omega \in \mathcal{C}, \exists p \in P(\omega) : K(p) \stackrel{+}{<} K(\omega)$.

Proof. As the sequence ω is computable, there must exist at least one algorithm that generates ω. Let q be the shortest such algorithm and construct an algorithm p that "predicts" ω as follows: Firstly the algorithm p reads $x_{1:n}$ to find the value of n, then it runs q to generate $x_{1:n+1}$ and returns x_{n+1} as its prediction. Clearly p perfectly predicts ω and $|p| < |q| + c$, for some small constant c that is independent of ω and q. □

Not only can any computable sequence be predicted, there also exist very simple predictors able to predict arbitrarily complex sequences:

3.2 Lemma. *There exists a predictor p such that $\forall n \in \mathbb{N}, \exists\, \omega \in \mathcal{C} : p \in P(\omega)$ and $K(\omega) > n$.*

Proof. Take a string x such that $K(x) = |x| \geq 2n$, and from this define a sequence $\omega := x0000\ldots$. Clearly $K(\omega) > n$ and yet a simple predictor p that always predicts 0 can learn to predict ω. □

The predictor used in the above proof is very simple and can only "learn" sequences that end with all 0's, albeit where the initial string can have arbitrarily high Kolmogorov complexity. It may seem that this is due to sequences that are initially complex but where the "tail complexity", defined $\liminf_{i\to\infty} K(\omega_{i:\infty})$, is zero. This is not the case:

3.3 Lemma. *There exists a predictor p such that $\forall n \in \mathbb{N}, \exists\, \omega \in \mathcal{C} : p \in P(\omega)$ and $\liminf_{i\to\infty} K(\omega_{i:\infty}) > n$.*

Proof. A predictor p for eventually periodic sequences can be defined as follows: On input $\omega_{1:k}$ the predictor goes through the ordered pairs $(1,1),(1,2),(2,1),(1,3),(2,2),(3,1),(1,4),\ldots$

checking for each pair (a, b) whether the string $\omega_{1:k}$ consists of an initial string of length a followed by a repeating string of length b. On the first match that is found p predicts that the repeating string continues, and then p halts. If $a + b > k$ before a match is found, then p outputs a fixed symbol and halts. Clearly $K(p)$ is a small constant and p will learn to predict any sequence that is eventually periodic.

For any $(m, n) \in \mathbb{N}^2$, let $\omega := x(y^*)$ where $x \in \mathbb{B}^m$, and $y \in \mathbb{B}^n$ is a random string, that is, $K(y) = n$. As ω is eventually periodic $p \in P(\omega)$ and also we see that $\liminf_{i\to\infty} K(\omega_{i:\infty}) = \min\{K(\omega_{m+1:\infty}), K(\omega_{m+2:\infty}), \ldots, K(\omega_{m+n:\infty})\}$.

For any $k \in \{1, \ldots, n\}$ let q_k^* be the shortest program that can generate $\omega_{m+k:\infty}$. We can define a halting program q_k' that outputs y where this program consists of q_k^*, n and k. Thus, $|q_k'| = |q_k^*| + O(\log n) = K(\omega_{k:\infty}) + O(\log n)$. As $n = K(y) \leq |q_k'|$, we see that $K(\omega_{k:\infty}) > n - O(\log n)$. As n and k are arbitrary the result follows. □

Using a more sophisticated version of this proof it can be shown that there exist predictors that can learn to predict arbitrary regular or primitive recursive sequences. Thus we might wonder whether there exists a computable predictor able to learn to predict all computable sequences. Unfortunately, no universal predictor exists, indeed for every predictor there exists a sequence which it cannot predict at all:

3.4 Lemma. *For any predictor p there constructively exists a sequence $\omega := x_1 x_2 \ldots \in \mathcal{C}$ such that $\forall n \in \mathbb{N} : p(x_{1:n}) \neq x_{n+1}$ and $K(\omega) \stackrel{+}{<} K(p)$.*

Proof. For any computable predictor p there constructively exists a computable sequence $\omega = x_1 x_2 x_3 \ldots$ computed by an algorithm q defined as follows: Set $x_1 = 1 - p(\lambda)$, then $x_2 = 1 - p(x_1)$, then $x_3 = 1 - p(x_{1:2})$ and so on. Clearly $\omega \in \mathcal{C}$ and $\forall n \in \mathbb{N} : p(x_{1:n}) = 1 - x_{n+1}$.

Let p^* be the shortest program that computes the same function as p and define a sequence generation algorithm q^* based on p^* using the procedure above. By construction, $|q^*| = |p^*| + c$ for some constant c that is independent of p^*. Because q^* generates ω, it follows that $K(\omega) \leq |q^*|$. By definition $K(p) = |p^*|$ and so $K(\omega) \stackrel{+}{<} K(p)$. □

Allowing the predictor to be probabilistic does not fundamentally avoid the problem of Lemma 3.4. In each step, rather than generating the opposite to what will be predicted by p, instead q attempts to generate the symbol which p is least likely to predict given $x_{1:n}$. To do this q must simulate p in order to estimate $\Pr\big(p(x_{1:n}) = 1 \big| x_{1:n}\big)$. With sufficient simulation effort, q can estimate this probability to any desired accuracy for any $x_{1:n}$. This produces a computable sequence ω such that $\forall n \in \mathbb{N} : \Pr\big(p(x_{1:n}) = x_{n+1} \big| x_{1:n}\big)$ is not significantly greater than $\frac{1}{2}$, that is, the performance of p is no better than a predictor that makes completely random predictions.

As probabilistic prediction complicates things without avoiding this fundamental problem, in the remainder of this paper we will consider only deterministic predictors. This will also allow us to see the roots of this problem as clearly as possible. With the preliminaries covered, we now move on to the central problem considered in this paper: Predicting sequences of limited Kolmogorov complexity.

4 Prediction of simple computable sequences

As the computable prediction of any computable sequence is impossible, a weaker goal is to be able to predict all "simple" computable sequences.

4.1 Definition. For $n \in \mathbb{N}$, let $\mathcal{C}_n := \{\omega \in \mathcal{C} : K(\omega) \leq n\}$. Further, let $P_n := P(\mathcal{C}_n)$ be the set of predictors able to learn to predict all sequences in $\mathcal{C}_n$.

Firstly we establish that prediction algorithms exist that can learn to predict all sequences up to a given complexity, and that these predictors need not be significantly more complex than the sequences they can predict:

4.2 Lemma. $\forall n \in \mathbb{N}, \exists p \in P_n : K(p) \stackrel{+}{<} n + O(\log n)$.

Proof. Let $h \in \mathbb{N}$ be the number of programs of length n or less which generate infinite sequences. Build the value of h into a prediction algorithm p constructed as follows:

In the k^{th} prediction cycle run in parallel all programs of length n or less until h of these programs have each produced $k+1$ symbols of output. Next predict according to the $k+1^{th}$ symbol of the generated string whose first k symbols is consistent with the observed string. If two generated strings are consistent with the observed sequence (there cannot be more than two as the strings are binary and have length $k+1$), pick the one which was generated by the program that occurs first in a lexicographical ordering of the programs. If no generated output is consistent, give up and output a fixed symbol.

For sufficiently large k, only the h programs which produce infinite sequences will produce output strings of length $k+1$. As this set of sequences is finite, they can be uniquely identified by finite initial strings. Thus for sufficiently large k the predictor p will correctly predict any computable sequence ω for which $K(\omega) \leq n$, that is, $p \in P_n$.

As there are $2^{n+1}-1$ possible strings of length n or less, $h < 2^{n+1}$ and thus we can encode h with $\log_2 h + 2\log_2\log_2 h = n + 1 + 2\log_2(n+1)$ bits. Thus, $K(p) < n + 1 + 2\log_2(n+1) + c$ for some constant c that is independent of n. □

Can we do better than this? Lemmas 3.2 and 3.3 shows us that there exist predictors able to predict at least some sequences vastly more complex than themselves. This suggests that there might exist simple predictors able to predict arbitrary sequences up to a high complexity. Formally, could there exist $p \in P_n$ where $n \gg K(p)$? Unfortunately, these simple but powerful predictors are not possible:

4.3 Theorem. $\forall n \in \mathbb{N} : p \in P_n \Rightarrow K(p) \stackrel{+}{>} n$.

Proof. For any $n \in \mathbb{N}$ let $p \in P_n$, that is, $\forall \omega \in \mathcal{C}_n : p \in P(\omega)$. By Lemma 3.4 we know that $\exists \omega' \in \mathcal{C} : p \notin P(\omega')$. As $p \notin P(\omega')$ it must be the case that $\omega' \notin \mathcal{C}_n$, that is, $K(\omega') \geq n$. From Lemma 3.4 we also know that $K(p) \stackrel{+}{>} K(\omega')$ and so the result follows. □

Intuitively the reason for this is as follows: Lemma 3.4 guarantees that every simple predictor fails for at least one simple sequence. Thus if we want a predictor that can learn to predict all sequences up to a moderate level of complexity, then clearly the predictor cannot be simple. Likewise, if we want a predictor that can predict all sequences up to a high level of complexity, then the predictor itself must be very complex. Thus, even though we have made the generous assumption of unlimited computational resources and data to learn from, only very complex algorithms can be truly powerful predictors.

These results easily generalise to notions of complexity that take computation time into consideration. As sequences are infinite, the appropriate measure of time is the time needed to generate

or predict the next symbol in the sequence. Under any reasonable measure of time complexity, the operation of inverting a single output from a binary valued function can be performed with little cost. If C is any complexity measure with this property, it is trivial to see that the proof of Lemma 3.4 still holds for C. From this, an analogue of Theorem 4.3 for C easily follows.

With similar arguments these results also generalise in a straightforward way to complexity measures that take space or other computational resources into account. Thus, the fact that extremely powerful predictors must be very complex, holds under any measure of complexity for which inverting a single bit is inexpensive.

5 Complexity of prediction

Another way of viewing these results is in terms of an alternate notion of sequence complexity defined as the size of the smallest predictor able to learn to predict the sequence. This allows us to express the results of the previous sections more concisely. Formally, for any sequence ω define the complexity measure,

$$\dot{K}(\omega) := \min_{p \in \mathbb{B}^*} \{|p| : p \in P(\omega)\},$$

and $\dot{K}(\omega) := \infty$ if $P(\omega) = \varnothing$. Thus, if $\dot{K}(\omega)$ is high then the sequence ω is complex in the sense that only complex prediction algorithms are able to learn to predict it. It can easily be seen that this notion of complexity has the same invariance to the choice of reference universal Turing machine as the standard Kolmogorov complexity measure.

It may be tempting to conjecture that this definition simply describes what might be called the "tail complexity" of a sequence, that is, $\dot{K}(\omega)$ is equal to $\liminf_{i \to \infty} K(\omega_{i:\infty})$. This is not the case. In the proof of Lemma 3.3 we saw that there exists a single predictor capable of learning to predict any sequence that consists of a repeating string, and thus for these sequences $\dot{K}$ is bounded. It was further shown that there exist sequences of this form with arbitrarily high tail complexity. Clearly then tail complexity and $\dot{K}$ cannot be equal in general.

Using $\dot{K}$ we can now rewrite a number of our previous results much more succinctly. From Lemma 3.1 it immediately follows that,

$$\forall \omega : 0 \leq \dot{K}(\omega) \stackrel{+}{<} K(\omega).$$

From Lemma 3.2 we know that $\exists c \in \mathbb{N}, \forall n \in \mathbb{N}, \exists\, \omega \in \mathcal{C}$ such that $\dot{K}(\omega) < c$ and $K(\omega) > n$, that is, $\dot{K}$ can attain the lower bound above within a small constant, no matter how large the value of K is. The sequences for which the upper bound on $\dot{K}$ is tight are interesting as they are the ones which demand complex predictors. We prove the existence of these sequences and look at some of their properties in the next section.

The complexity measure $\dot{K}$ can also be generalised to sets of sequences, for $S \subset \mathbb{B}^\infty$ define $\dot{K}(S) := \min_p \{|p| : p \in P(S)\}$. This allows us to rewrite Lemma 4.2 and Theorem 4.3 as simply,

$$\forall n \in \mathbb{N} : n \stackrel{+}{<} \dot{K}(\mathcal{C}_n) \stackrel{+}{<} n + O(\log n).$$

This is just a restatement of the fact that the simplest predictor capable of predicting all sequences up to a Kolmogorov complexity of n, has itself a Kolmogorov complexity of roughly n.

Perhaps the most surprising thing about $\dot{K}$ complexity is that this very natural definition of the complexity of a sequence, as viewed from the perspective of prediction, does not appear to have been studied before.

6 Hard to predict sequences

We have already seen that some individual sequences, such as the repeating string used in the proof of Lemma 3.3, can have arbitrarily high Kolmogorov complexity but nevertheless can be predicted by trivial algorithms. Thus, although these sequences contain a lot of information in the Kolmogorov sense, in a deeper sense their structure is very simple and easily learnt.

What interests us in this section is the other extreme; individual sequences which can only be predicted by complex predictors. As we are only concerned with prediction in the limit, this extra complexity in the predictor must be some kind of special information which cannot be learnt just through observing the sequence. Our first task is to show that these difficult to predict sequences exist.

6.1 Theorem. $\forall n \in \mathbb{N}, \exists\, \omega \in \mathcal{C} : n \stackrel{+}{<} \dot{K}(\omega) \stackrel{+}{<} K(\omega) \stackrel{+}{<} n + O(\log n)$.

Proof. For any $n \in \mathbb{N}$, let $Q_n \subset \mathbb{B}^{<n}$ be the set of programs shorter than n that are predictors, and let $x_{1:k} \in \mathbb{B}^k$ be the observed initial string from the sequence ω which is to be predicted. Now construct a meta-predictor $\hat{p}$:

By dovetailing the computations, run in parallel every program of length less than n on every string in $\mathbb{B}^{\leq k}$. Each time a program is found to halt on all of these input strings, add the program to a set of "candidate prediction algorithms", called $\tilde{Q}_n^k$. As each element of Q_n is a valid predictor, and thus halts for all input strings in $\mathbb{B}^*$ by definition, for every n and k it eventually will be the case that $|\tilde{Q}_n^k| = |Q_n|$. At this point the simulation to approximate Q_n terminates. It is clear that for sufficiently large values of k all of the valid predictors, and only the valid predictors, will halt with a single symbol of output on all tested input strings. That is, $\exists r \in \mathbb{N}, \forall k > r : \tilde{Q}_n^k = Q_n$.

The second part of the $\hat{p}$ algorithm uses these candidate prediction algorithms to make a prediction. For $p \in \tilde{Q}_n^k$ define $d^k(p) := \sum_{i=1}^{k-1} |p(x_{1:i}) - x_{i+1}|$. Informally, $d^k(p)$ is the number of prediction errors made by p so far. Compute this for all $p \in \tilde{Q}_n^k$ and then let $p_k^* \in \tilde{Q}_n^k$ be the program with minimal $d^k(p)$. If there is more than one such program, break the tie by letting p_k^* be the lexicographically first of these. Finally, $\hat{p}$ computes the value of $p_k^*(x_{1:k})$ and then returns this as its prediction and halts.

By Lemma 3.4, there exists $\omega' \in \mathcal{C}$ such that $\hat{p}$ makes a prediction error for every k when trying to predict ω'. Thus, in each cycle at least one of the finitely many predictors with minimal d^k makes a prediction error and so $\forall p \in Q_n : d^k(p) \to \infty$ as $k \to \infty$. Therefore, $\nexists p \in Q_n : p \in P(\omega')$, that is, no program of length less than n can learn to predict ω' and so $n \leq \dot{K}(\omega')$. Further, from Lemma 3.1 we know that $\dot{K}(\omega') \stackrel{+}{<} K(\omega')$, and from Lemma 3.4 again, $K(\omega') \stackrel{+}{<} K(\hat{p})$.

Examining the algorithm for $\hat{p}$, we see that it contains some fixed length program code and an encoding of $|Q_n|$, where $|Q_n| < 2^n - 1$. Thus, using a standard encoding method for integers, $K(\hat{p}) \stackrel{+}{<} n + O(\log n)$.

Chaining these together we get, $n \stackrel{+}{<} \dot{K}(\omega') \stackrel{+}{<} K(\omega') \stackrel{+}{<} K(\hat{p}) \stackrel{+}{<} n + O(\log n)$, which proves the theorem. □

This establishes the existence of sequences with arbitrarily high $\dot{K}$ complexity which also have a similar level of Kolmogorov complexity. Next we establish a fundamental property of high $\dot{K}$ complexity sequences: they are extremely difficult to compute.

For an algorithm q that generates $\omega \in \mathcal{C}$, define $t_q(n)$ to be the number of computation steps performed by q before the n^{th} symbol of ω is written to the output tape. For example, if q is a

simple algorithm that outputs the sequence 010101..., then clearly $t_q(n) = O(n)$ and so ω can be computed quickly. The following theorem proves that if a sequence can be computed in a reasonable amount of time, then the sequence must have a low $\dot{K}$ complexity:

6.2 Lemma. *$\forall \omega \in \mathcal{C}$, if $\exists q : \mathcal{U}(q) = \omega$ and $\exists r \in \mathbb{N}, \forall n > r : t_q(n) < 2^n$, then $\dot{K}(\omega) \stackrel{+}{=} 0$.*

Proof. Construct a prediction algorithm $\tilde{p}$ as follows:

On input $x_{1:n}$, run all programs of length n or less, each for 2^{n+1} steps. In a set W_n collect together all generated strings which are at least $n+1$ symbols long and where the first n symbols match the observed string $x_{1:n}$. Now order the strings in W_n according to a lexicographical ordering of their generating programs. If $W_n = \varnothing$, then just return a prediction of 1 and halt. If $|W_n| > 1$ then return the $n + 1^{th}$ symbol from the first sequence in the above ordering.

Assume that $\exists q : \mathcal{U}(q) = \omega$ such that $\exists r \in \mathbb{N}, \forall n > r : t_q(n) < 2^n$. If q is not unique, take q to be the lexicographically first of these. Clearly $\forall n > r$ the initial string from ω generated by q will be in the set W_n. As there is no lexicographically lower program which can generate ω within the time constraint $t_q(n) < 2^n$ for all $n > r$, for sufficiently large n the predictor $\tilde{p}$ must converge on using q for each prediction and thus $\tilde{p} \in P(\omega)$. As $|\tilde{p}|$ is clearly a fixed constant that is independent of ω, it follows then that $\dot{K}(\omega) < |\tilde{p}| \stackrel{+}{=} 0$. □

We could replace the 2^n bound in the above result with any monotonically growing computable function, for example, 2^{2^n}. In any case, this does not change the fundamental result that sequences which have a high $\dot{K}$ complexity are practically impossible to compute. However from our theoretical perspective these sequences present no problem as they can be predicted, albeit with immense difficulty.

7 The limits of mathematical analysis

One way to interpret the results of the previous sections is in terms of constructive theories of prediction. Essentially, a constructive theory of prediction $\mathcal{T}$, expressed in some sufficiently rich formal system $\mathcal{F}$, is in effect a description of a prediction algorithm with respect to a universal Turing machine which implements the required parts of $\mathcal{F}$. Thus from Theorems 4.3 and 6.1 it follows that if we want to have a predictor that can learn to predict all sequences up to a high level of Kolmogorov complexity, or even just predict individual sequences which have high $\dot{K}$ complexity, the constructive theory of prediction that we base our predictor on must be very complex. Elegant and highly general constructive theories of prediction simply do not exist, even if we assume unlimited computational resources. This is in marked contrast to Solomonoff's highly elegant but non-constructive theory of prediction.

Naturally, highly complex theories of prediction will be very difficult to mathematically analyse, if not practically impossible. Thus at some point the development of very general prediction algorithms must become mainly an experimental endeavour due to the difficulty of working with the required theory. Interestingly, an even stronger result can be proven showing that beyond some point the mathematical analysis is in fact impossible, even in theory:

7.1 Theorem. *In any consistent formal axiomatic system $\mathcal{F}$ that is sufficiently rich to express statements of the form "$p \in P_n$", there exists $m \in \mathbb{N}$ such that for all $n > m$ and for all predictors $p \in P_n$ the true statement "$p \in P_n$" cannot be proven in $\mathcal{F}$.*

In other words, even though we have proven that very powerful sequence prediction algorithms exist, beyond a certain complexity it is impossible to find any of these algorithms using mathematics. The proof has a similar structure to Chaitin's information theoretic proof [3] of Gödel incompleteness theorem for formal axiomatic systems [6].

Proof. For each $n \in \mathbb{N}$ let T_n be the set of statements expressed in the formal system $\mathcal{F}$ of the form "$p \in P_n$", where p is filled in with the complete description of some algorithm in each case. As the set of programs is denumerable, T_n is also denumerable and each element of T_n has finite length. From Lemma 4.2 and Theorem 4.3 it follows that each T_n contains infinitely many statements of the form "$p \in P_n$" which are true.

Fix n and create a search algorithm s that enumerates all proofs in the formal system $\mathcal{F}$ searching for a proof of a statement in the set T_n. As the set T_n is recursive, s can always recognise a proof of a statement in T_n. If s finds any such proof, it outputs the corresponding program p and then halts.

By way of contradiction, assume that s halts, that is, a proof of a theorem in T_n is found and p such that $p \in P_n$ is generated as output. The size of the algorithm s is a constant (a description of the formal system $\mathcal{F}$ and some proof enumeration code) as well as an $O(\log n)$ term needed to describe n. It follows then that $K(p) \stackrel{+}{<} O(\log n)$. However from Theorem 4.3 we know that $K(p) \stackrel{+}{>} n$. Thus, for sufficiently large n, we have a contradiction and so our assumption of the existence of a proof must be false. That is, for sufficiently large n and for all $p \in P_n$, the true statement "$p \in P_n$" cannot be proven within the formal system $\mathcal{F}$. □

The exact value of m depends on our choice of formal system $\mathcal{F}$ and which reference machine $\mathcal{U}$ we measure complexity with respect to. However for reasonable choices of $\mathcal{F}$ and $\mathcal{U}$ the value of m would be in the order of 1000. That is, the bound m is certainly not so large as to be vacuous.

8 Discussion

Solomonoff induction is an elegant and extremely general model of inductive learning. It neatly brings together the philosophical principles of Occam's razor, Epicurus' principle of multiple explanations, Bayes theorem and Turing's model of universal computation into a theoretical sequence predictor with astonishingly powerful properties. If theoretical models of prediction can have such elegance and power, one cannot help but wonder whether similarly beautiful and highly general computable theories of prediction are also possible.

What we have shown here is that there does not exist an elegant constructive theory of prediction for computable sequences, even if we assume unbounded computational resources, unbounded data and learning time, and place moderate bounds on the Kolmogorov complexity of the sequences to be predicted. Very powerful computable predictors are therefore necessarily complex. We have further shown that the source of this problem is computable sequences which are extremely expensive to compute. While we have proven that very powerful prediction algorithms which can learn to predict these sequences exist, we have also proven that, unfortunately, mathematical analysis cannot be used to discover these algorithms due to problems of Gödel incompleteness.

These results can be extended to more general settings, specifically to those problems which are equivalent to, or depend on, sequence prediction. Consider, for example, a reinforcement learning agent interacting with an environment [15, 8]. In each interaction cycle the agent must choose its actions so as to maximise the future rewards that it receives from the environment.

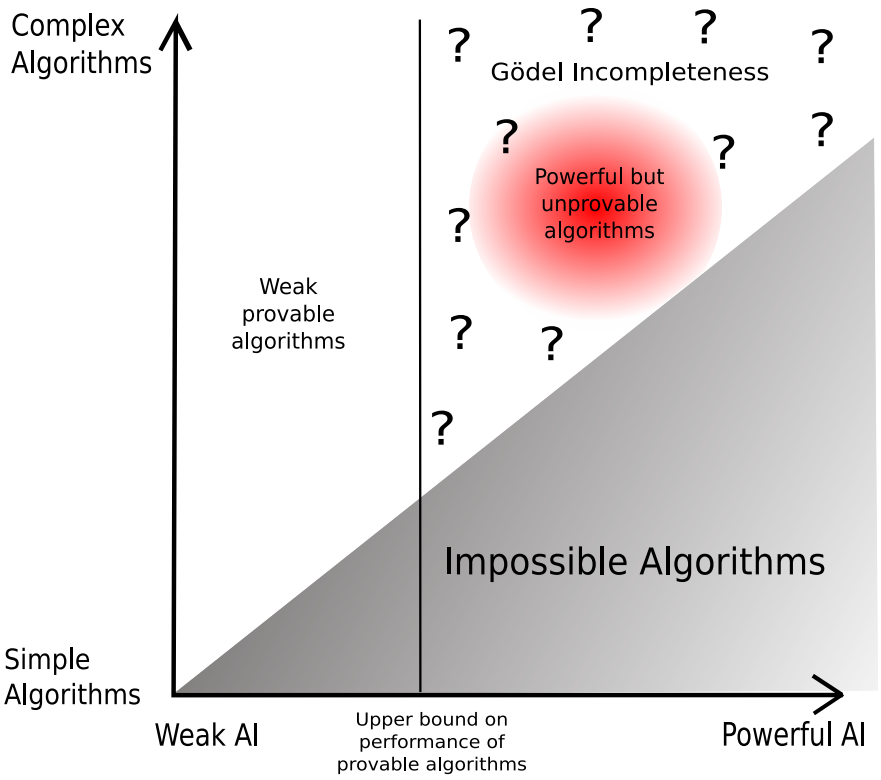


Figure 1: Theorem 4.3 rules out simple but powerful artificial intelligence algorithms, as indicated by the greyed out region on the lower right. Theorem 7.1 upper bounds how powerful an algorithm can be before it can no longer be proven to be a powerful algorithm. This is indicated by the vertical line separating the region of provable algorithms from the region of Gödel incompleteness. Note: This diagram is a correction of the published version.

Of course the agent cannot know for certain whether or not some action will lead to rewards in the future, thus it must predict these. Clearly, at the heart of reinforcement learning lies a prediction problem, and so the results for computable predictors presented in this paper also apply to computable reinforcement learners. More specifically, from Theorem 4.3 it follows that very powerful computable reinforcement learners are necessarily complex, and from Theorem 7.1 it follows that it is impossible to discover extremely powerful reinforcement learning algorithms mathematically. These relationships are illustrated in Figure 1.

It is reasonable to ask whether the assumptions we have made in our model need to be changed. If we increase the power of the predictors further, for example by providing them with some kind of an oracle, this would make the predictors even more unrealistic than they currently are. Clearly this goes against our goal of finding an elegant, powerful and general prediction theory that is more realistic in its assumptions than Solomonoff's incomputable model. On the other hand, if we weaken our assumptions about the predictors' resources to make them more realistic, we are

in effect taking a subset of our current class of predictors. As such, all the same limitations and problems will still apply, as well as some new ones.

It seems then that the way forward is to further restrict the problem space. One possibility would be to bound the amount of computation time needed to generate the next symbol in the sequence. However if we do this without restricting the predictors' resources then the simple predictor from Lemma 6.2 easily learns to predict any such sequence and thus the problem of prediction in the limit has become trivial. Another possibility might be to bound the memory of the machine used to generate the sequence, however this makes the generator a finite state machine and thus bounds its computation time, again making the problem trivial.

Perhaps the only reasonable solution would be to add additional restrictions to both the algorithms which generate the sequences to be predicted, and to the predictors. We may also want to consider not just learnability in the limit, but also how quickly the predictor is able to learn. Of course we are then facing a much more difficult analysis problem.

Acknowledgements

I would like to thank Marcus Hutter, Alexey Chernov, Daniil Ryabko and Laurent Orseau for useful discussions and advice during the development of this paper.

References

[1] J. M. Barzdin. Prognostication of automata and functions. *Information Processing*, 71:81–84, 1972.

[2] C. S. Calude. *Information and Randomness*. Springer, Berlin, 2nd edition, 2002.

[3] G. J. Chaitin. Gödel's theorem and information. *International Journal of Theoretical Physics*, 22:941–954, 1982.

[4] A. P. Dawid. Comment on The impossibility of inductive inference. *Journal of the American Statistical Association*, 80(390):340–341, 1985.

[5] M. Feder, N. Merhav, and M. Gutman. Universal prediction of individual sequences. *IEEE Trans. on Information Theory*, 38:1258–1270, 1992.

[6] K. Gödel. Über formal unentscheidbare Sätze der principia mathematica und verwandter systeme I. *Monatshefte für Matematik und Physik*, 38:173–198, 1931. [English translation by E. Mendelsohn: "On undecidable propositions of formal mathematical systems". In M. Davis, editor, *The undecidable*, pages 39–71, New York, 1965. Raven Press, Hewlitt].

[7] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[8] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005. 300 pages, http://www.idsia.ch/~marcus/ai/uaibook.htm.

[9] M. Hutter. On the foundations of universal sequence prediction. In *Proc. 3rd Annual Conference on Theory and Applications of Models of Computation (TAMC'06)*, volume 3959 of *LNCS*, pages 408–420. Springer, 2006.

[10] M. Li and P. M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2nd edition, 1997.

[11] J. Poland and M. Hutter. Convergence of discrete MDL for sequential prediction. In *Proc. 17th Annual Conf. on Learning Theory (COLT'04)*, volume 3120 of *LNAI*, pages 300–314, Banff, 2004. Springer, Berlin.

[12] J. J. Rissanen. Fisher Information and Stochastic Complexity. *IEEE Trans. on Information Theory*, 42(1):40–47, January 1996.

[13] R. J. Solomonoff. A formal theory of inductive inference: Part 1 and 2. *Inform. Control*, 7:1–22, 224–254, 1964.

[14] R. J. Solomonoff. Complexity-based induction systems: comparisons and convergence theorems. *IEEE Trans. Information Theory*, IT-24:422–432, 1978.

[15] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. Cambridge, MA, MIT Press, 1998.

[16] V. V. V'yugin. Non-stochastic infinite and finite sequences. *Theoretical computer science*, 207:363–382, 1998.

[17] C. S. Wallace and D. M. Boulton. An information measure for classification. *Computer Jrnl.*, 11(2):185–194, August 1968.

[18] F.M.J. Willems, Y.M. Shtarkov, and Tj.J. Tjalkens. The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41(3), 1995.