Kolmogorov Complexity and Universal Probability

Mário S. Alvim
(msalvim@dcc.ufmg.br)

Information Theory

DCC-UFMG
(2017/02)

Kolmogorov complexity - Introduction

- So far in our course we have tackled the problem of evaluating the information contained in an event in terms of the probability of this event.
- Shannon's information theory measures the information contained in an object in terms of the probability distribution of the source that produces the object:
  - less probable objects carry more information,
  - more probable objects carry less information.
- But what if we want to measure the information contained in an object without referring to the probability of a source producing the object?

  In other words, what if we want to find the intrinsic complexity of an object?
- This is problem is the core of **algorithmic information theory**, or **Kolmogorov complexity**.

Simple vs. complex strings

- Example 1 Consider the following binary strings.

(a) 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

(b) 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1

(c) 1 1 0 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1

Which ones you think are "complex", and which ones you think are "simple"?

- String (a) is just 01 repeated 16 times.
- String (b) looks complex, but it is just the decimal expansion of $\sqrt{2} - 1$.
- String (c) looks random, but the frequency of 1's is higher than the frequency of 0's.

  This means that this string can actually be compressed.

◁

Simple vs. complex strings

- The previous example suggests that the complexity of a string is related to how "easily" we can describe this string.
- Based on this intuition, we could try to define the intrinsic complexity of a string as follows:

  "Strings are simpler if they have shorter descriptions."

- That is how we are gonna do define intrinsic complexity.

  But there's a caveat: we have to be cautious about what we mean by "description", as it can be a very tricky term...

The problem of defining "description" - Berry's paradox

- **(Berry's paradox)** Consider the sentence:

  "The smallest positive integer not definable in under sixty letters."

  What number is described by this sentence?

  Note that:

  1. There is a finite number of strings under sixty letters, therefore there is only a finite number of positive integers that can be described by sentences under sixty letters.
  2. From all positive integers not describable in under sixty letters, there is one that is the smallest of all (well-ordering of integers). This is the unique, precise integer this sentence is describing.
  3. But this sentence has itself fewer than 60 letters, so the sentence is a description in under sixty letters of the smallest integer that is not describable in under sixty letters!

     Paradox!

The problem of defining "description" - Berry's paradox

- Berry's paradox tells us that even if we have a well-defined description of a number (or object), it does not necessarily mean that we can compute (i.e., find) the number (or object).

  You may have encountered similar paradoxes before: the halting problem, the barber's problem, etc...

- Since we like things we can compute (we are very pragmatic people), we are only interested in descriptions that can be computed.

  That is:

  We will only consider as valid descriptions those that can produce the object they refer to.

"Description" as computable functions

- We will use as descriptions **computable functions**, that is, functions that can produce the objects they describe.

  But does it mean for a function to be "computable"?

- **Church-Turing thesis:** *"Every effectively computable function can be computed by a Turing machine."*

  In other words, the thesis says that if there is a way to compute a function at all, there is a way to compute this function in a Turing machine.

  Or yet, if you can't compute a function in a Turing machine, that's because the function cannot be computed at all.

- By accepting the Church-Turing thesis (which is the current definition of "computation"), we can use Turing machines as the formal definition of computable function.

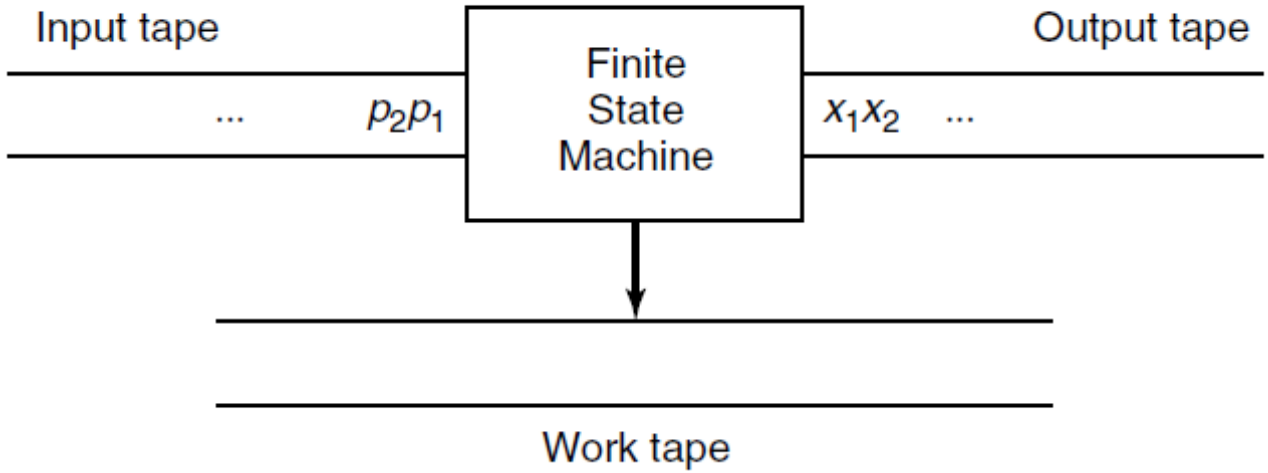
"Description" as computable functions

- A **Universal Turing machine (UTM)** is a Turing machine that can simulate any other Turing machine.

  Intuitively, a UTM is a Machine that can compute any computable function if it is given the right program as input.

  (You are used to UTMs: your laptop is one, as are many modern devices!)

- A UTM reads a **program** (which is a string) from an input tape and writes a string in an output tape.



"Description" as computable functions

- There is broad consensus among Computer Scientists that the Church-Turing thesis is most probably true.
  - If you are curious to know the reasons, just take my course on Theory of Computing (DCC 129 - FTC)!
- By accepting the Church-Turing thesis, we obtain the following formal and valid (non-paradoxical) definition of the description of a string.

  A **valid description of a string** is a program that, when running in a Universal Turing machine, will output the string and then halt.
- We could describe any mathematical object, but here we will focus on binary strings for the sake of simplicity.

Kolmogorov complexity

- Now we are ready to formalize the intuition from our first example: that the complexity of an object is related to the ease of describing such an object.
- The **Kolmogorov complexity** of a string is the length of the shortest possible program that, when running in a Universal Turing machine, outputs the string and then halts.

  We denote by $\mathcal{U}(p)$ the output of a UTM running program p. The outcome of $\mathcal{U}(p)$ may be either:

  a) a finite binary string, and the machine then halts; or

  b) an eternal loop, producing a possibly infinite binary string.

  Then, the Kolmogorov complexity of a string s with respect to (w.r.t.) a UTM $\mathcal{U}$ is defined as

$$K_{\mathcal{U}}(s) = \min_{p:\mathcal{U}(p)=s} \ell(p),$$

  where $\ell(p)$ is the length, in bits, of program p.

Kolmogorov complexity - Properties

- At this point, a natural question may arise. We know that:

  1. we have to pick an UTM to define the Kolmogorov complexity of an object, and
  2. there are several alternative implementations of UTMs,

  So, can a same object be "simple" when described in a particular UTM, but "complex" when described in a different UTM?

- The following results states that the Kolmogorov complexity of a string does not depend (up to an additive factor) on the UTM you decide to describe it.

  Theorem (**Independence from the UTM chosen.**) If you have two different UTMs $\mathcal{U}'$ and $\mathcal{U}''$ you have

  $$K_{\mathcal{U}'}(s) \leq K_{\mathcal{U}''}(s) + c,$$

  where c is a constant that does not depend on s.

Kolmogorov complexity - Properties

- **Proof.** UTMs can simulate each other.

  That is, for every pair of UTMs $\mathcal{U}'$, $\mathcal{U}''$ there is a program p_I working as an **interpreter** that tells $\mathcal{U}'$ how to simulate $\mathcal{U}''$.

  Hence, to describe string s in $\mathcal{U}'$ we can use a program p that is the concatenation of:

  a) the shortest program to produce s in $\mathcal{U}''$, and

  b) the interpreter p_I telling how to simulate $\mathcal{U}''$ in $\mathcal{U}'$.

  Hence, the Kolmogorov complexity of s in $\mathcal{U}'$ satisfies

  $$K_{\mathcal{U}'}(s) \leq K_{\mathcal{U}''}(s) + \ell(p_I),$$

  where $\ell(p_I) = c$ is the constant size of the interpreter. □

Kolmogorov complexity - Properties

- In other words, the previous theorem means that for every string s, the difference in complexity caused by the choice of different UTMs is limited by a constant factor:

$$|K_{\mathcal{U}'}(s) - K_{\mathcal{U}''}(s)| \leq c.$$

  Hence, if the object s is complex enough, the constant factor becomes negligible.

- Because of that, we can drop the subscript $\mathcal{U}$ in $K_{\mathcal{U}}(s)$ and only use $K(s)$ to refer to the Kolmogorov complexity of a string in any UTM.

- For the same reason, it is also usual to describe the Kolmogorov complexity of a string as the shortest program in a high-level (Turing-equivalent) programming language (such as C, Haskell, or Python) that outputs the string.

Kolmogorov complexity - Properties

- Another property states that the Kolmogorov complexity $K(s)$ of a string s of known length $\ell(s)$ is not much longer than the length $\ell(s)$ of the string.

- **Theorem** (**Upper bound on conditional complexity.**)

$$K(s \mid \ell(s)) \leq \ell(s) + c,$$

where c is a constant that does not depend on s.

Proof. The program "Print the following $\ell(s)$-bit long string: s" prints the string s using $\ell(s)$ bits to describe s, plus a constant number of bits c to describe the command "print the following $\ell(s)$-bit long string".

(Note that because the length $\ell(s)$ of the string is known beforehand, the print command has a constant size for every string of the same size.)

Hence $\ell(s) + c$ is an upper bound on $K(s)$. □

Kolmogorov complexity - Properties

- Without knowledge of the length of the string, the print command does not have a constant size, and we will need a way to learn when the string s is over.

  We can use an additional stop symbol or we can use a self-punctuating scheme, which leads to the following bound.

- **Theorem** (**Upper bound on Kolmogorov complexity.**)

$$K(s) \quad \leq \quad K(s \mid \ell(s)) + 2\log \ell(s) + 2 \quad \leq \quad \ell(s) + 2\log \ell(s) + c,$$

  where c is a constant that does not depend on s.

  Proof. If the computer does not know $\ell(s)$, we must have some way of informing it when the input string has come to an end.

  An (ineficient) way of doing it is to use the program for printing $K(s \mid \ell(s))$ (using $K(s \mid \ell(s))$ bitS), but modifying it to explicitly express the lenght of s. This can be done by expressing the binary expansion of $\ell(s)$ twice (using $2\log \ell(s)$ bits), then add a stop symbol (e.g., using the sequence 01, using $c = 2$ bits). □

Kolmogorov complexity - Properties

- The next property states that the complexity $K(ss)$ of the concatenation ss of a string s with itself is not much greater than the complexity $K(s)$ of the string.

- **Theorem** (**Complexity of the concatenation of a string with itself.**)

$$K(ss) \leq K(s) + c,$$

where c is a constant independent of s.

Proof. Homework!

Randomness and Algorithmic Incompressibility

Kolmogorov complexity - Randomness

- A string is **random** or **algorithmically incompressible** if

$$K(s) \geq \ell(s).$$

  Intuitively, a string is algorithically incompressible when the shortest program that outputs it is basically a program that says "Print s".

- An algorithmically incompressible string has no regularities that can be exploited to make its description shorter.

  Note that, intuitively, a very algorithmically compressible string should satisfy

$$K(s) \ll \ell(s).$$

Kolmogorov complexity - Randomness

- **Theorem** There are incompressible strings of any size:

$$\forall n \geq 0 : \exists \text{ string } s : \quad \ell(s) = n \quad \text{and} \quad K(s) \geq n.$$

Proof. There are 2^n strings of size n bits.

The number of programs having size smaller than $\ell(s) = n$ is the sum of all programs of size 0 bits, all programs of size 1 bit, all programs of size 2 bits, up to all programs of size $n-1$ bits.

This sum is calculated as

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1,$$

and hence there only $2^n - 1$ programs shorter than n bits.

Using the pigeonhole principle, we conclude that at least one of the 2^n strings of size n cannot be compressed to under n bits. □

Kolmogorov complexity - Randomness

- So we just saw that there is at least one incompressible string of any size.

  But, in fact, it is not only a few strings of each size that is incompressible, but rather that the vast majority of strings are incompressible.

- **Theorem** The fraction of strings of size n that can be generated by programs of size smaller than $m < n$ is at most

$$2^{m-n}.$$

Kolmogorov complexity - Randomness

- **Proof.** There are 2^n strings of size n, but only $2^m - 1$ programs of size smaller than m.

  If every one of the $2^m - 1$ programs of size smaller than m generates a string of size n, at most a fraction

  $$\frac{2^m - 1}{2^n} < 2^{m-n}$$

  of strings of size n can be contemplated with a short description. □

Kolmogorov complexity - Randomness

- Example 2 Consider a string of size 1 000 000 bits.

  What is the probability that this string can be compressed into programs of size 1% shorter than the string itself?

  What is the probability that this string can be compressed into programs of size 0.01% shorter than the string itself?

  Solution. Programs 1% shorter than the string have a lenght of at most 990 000 bits, so at most a fraction

$$2^{990\,000-1\,000\,000} = 2^{-10\,000}$$

  of the strings of size 1 000 000 can be compressed by 1%.

  If we consider programs 0.01% shorter we get that a most a fraction

$$2^{999\,900-1\,000\,000} = 2^{-100}$$

  of the strings of size 1 000 000 can be compressed by 0.01%.

◁

Komogorov complexity is uncomputable

Kolmogorov complexity - Computability

- **Theorem** Kolmogorov complexity is uncomputable.

Proof. By contradiction. Assume there is a function *Kolmogorov* that takes as input a string s and produces as output $K(s)$, for all s.

Using this function can write the following program:

```
GenerateComplexString()
   for i=1 to infinity
      for all strings s of size i
         if Kolmogorov(s) > size of GenerateComplexString
            return s
```

This program outputs the shortest string s whose Kolmogorov complexity exceeds the size $\ell(\mathit{GenerateComplexString})$ of the function.

But this is a contradiction because the string s was just produced by a program with size exactly $\ell(\mathit{GenerateComplexString})$!

Hence, the function *Kolmogorov* cannot possibly exist. □

Implications of Kolmogorov complexity

Kolmogorov complexity - Implications

1. **The complexity of a part may exceed the complexity of the whole.**

   1. The Kolmogorov complexity of the set of all integers vs. the Kolmogorov complexity of the set prime integers.

      $$K(\textit{set of prime integers}) \gg K(\textit{set of integers})$$

      (Explain why!)

   2. The Kolmogorov complexity of the set of all programs in Python and the Kolmogorov complexity of the set of all programs in Python that never halt.

      $$K(\textit{set of Python programs that never halt}) \gg K(\textit{set of Python programs})$$

      (Explain why!)

Kolmogorov complexity - Implications

2. **Random objects are uncompressible and, therefore, cannot have concise descriptions.**

   That's because you can only find short descriptions for non-random objects.

   1. You can explain the behavior of a set of balls on a friction-free billiard table, after an initial kick, from now to the end of the Universe, by specifying just the initial position of every ball and Newton's laws (which are quite short).

   2. You cannot describe whether stocks are going up or down, or if wars are going to happen, if these events are really random.

      That's why so many models of the Economy fail, and so often!

      This is related to the **hindsight fallacy**: after an utterly surprising event that no one saw coming (e.g., September 11th, the economic crisis of 2008), there's always someone coming up with a post-hoc explanation of why it was obvious all along that the event would happen.

      And yet, these people are invariable unable to predict the next "obvious" big event...

Universal Probability

Universal Probability

- Suppose that a computer is fed a random program.

  For instance:

  - feed a series of fair coin flips into a Universal Turing machine, or
  - (for the fun of it) imagine a monkey sitting at a keyboard and typing the keys at random.

  In either case, most strings will not make sense to the computer, and the computer will produce an error message.

  But with a certain probability the input can make sense, and the computer will then print out something meaningful.

  Will this output sequence look random?

Universal Probability

- To reason about the problem, we start by arguing that the probability of a program p that halts of size $\ell(p)$ being generated at random is given by

$$Pr(p) = 2^{-\ell(p)}.$$

  Intuitively, this means that shorter programs are more likely to be produced.

- We then define the **universal probability of a string** s as the probability that s is produced as the output of an Universal Turing machine fed with a random program:

$$p_{\mathcal{U}}(s) = \sum_{p:\mathcal{U}(p)=s} Pr(p) = \sum_{p:\mathcal{U}(p)=s} 2^{-\ell(p)}.$$

Universal Probability

- It can be shown that the universal probability of a string is closely related to its Kolmogorov complexity as follows:

$$p_{\mathcal{U}}(s) \approx 2^{-K(s)}$$

  The intuition is that the shortest program that produces the string s will contribute exponentially more to the sum $\sum_{p:\mathcal{U}(p)=s} 2^{-\ell(p)}$ than all other programs that produce s (since the probability of programs decay exponentially with their length).

- The intuition behind universal probability is that simpler strings are more likely than complicated strings.

  In other words:
  the lower the Kolmogorov complexity of a string,
  the more likely we will find the string in nature.

Universal Probability

- Example 3 If we wish to find out laws of physics, we should consider the simplest string describing the laws as the most likely.

  This principle, known as **Occam's Razor**, has been a general principle guiding scientific research for centuries:

  "If there are many explanations consistent with the observed data, choose the simplest."

  In our framework, Occam's Razor is equivalent to choosing the shortest program that produces a given string.

  ◁

Universal Probability

- Example 4 Consider we have a monkey at a typewriter and a monkey at a computer keyboard.

  If the monkey types at random on a typewriter, the probability that it types out all the works of Shakespeare (assuming that the text is 1 million bits long) is about

$$p_{typewriter}(\textit{works of Shakespeare}) \approx 2^{-1\,000\,000}.$$

  If the monkey sits at a computer terminal, however, the probability that it types out Shakespeare is now related to the Kolmogorov complexity of the works of Shakespeare, which can be approximated by

$$K(\textit{works of Shakespeare}) \approx 250\,000 \text{ bits},$$

  using a program that compacts these works using 250 000 bits.

  Then:

$$p_{computer}(\textit{works of Shakespeare}) \approx 2^{-K(\textit{works of Shakespeare})} \approx 2^{-250\,000}.$$

Universal Probability

- Example 4 (Continued)

  Although extremely small, the probability of the works of Shakespeare being produced by the monkey at a computer is still

  $$2^{750\,000}$$

  times larger than the probability of the works of Shakespeare being produced by the monkey at a dumb typewriter!

  The example indicates that a random input to a computer is much more likely to produce "interesting" outputs than a random input to a typewriter.

  We all know that a computer is an intelligence amplifier...

  Apparently, it creates sense from nonsense as well!

  ◁

Universal Probability and Bayesian Inference

Universal Probability and Bayesian Inference

- We have studied inference problems in which:
  1. some data or evidence
     $$D = x_1, x_2, \ldots$$
     is observed;
  2. a set of hypotheses or models
     $$H = h_1, h_2, \ldots$$
     is considered to contain all possible explanations for D; and
  3. the inference task is to decide which hypothesis (es) is (are) the most likely to be responsible for the observations.

Universal Probability and Bayesian Inference

- The **Bayesian method for inference** computes the a posteriori probability

$$p(h \mid D) = \frac{p(h)p(D \mid h)}{p(D)} = \frac{p(h)p(D \mid h)}{\sum_{h' \in H} p(h')p(D \mid h')}$$

  for each hypothesis $h \in H$ using the prior probability $p(h)$ of h, and the likelihood $p(D \mid h)$ of data D given h.

  We then pick as a best explanation for D a hypothesis h that maximizes $p(h \mid D)$.

- To compute $p(h \mid D)$ it is usually easy to compute the term $p(D \mid h)$ for each h.

  However, it is often unclear how to compute the prior probability $p(h)$ of each hypothesis.

- The problem is: the quality of your Bayesian inference might depend heavily on the prior probabilities you pick for your hypotheses.

Universal Probability and Bayesian Inference

- Example 5 Assume you want to use Bayes inference to find the best explanation for a series of phenomena.

  For the sake of argument, assume the available evidence you have access to is

  $$D = \text{“the whole entire world”}.$$

  How would you assign prior probabilities $p(h_1)$ and $p(h_2)$ to mutually-exclusive hypotheses h_1 and h_2 in the following cases?

  1. $h_1 =$ *“String theory is true”* vs. $h_2 =$ *“String theory is false”*?
  2. $h_1 =$ *“We live in the Matrix”* vs. $h_2 =$ *“We live in the real world”*?
  3. $h_1 =$ *“God exists”* vs. $h_2 =$ *“God does not exist”*? ◁

- Indeed, a fundamental problem of Bayesian inference is

  “How can one assign probability to a hypothesis before observing any data?”

- A theoretically sound solution to this problem was not known until the 1 960s, when **algorithmic probability theory** was developed.

Universal Probability and Bayesian Inference

- **Algorithmic probability theory** rests upon two philosophical principles.

1. **Epicurus' principle of multiple explanations**:

   "One should keep all hypotheses that are consistent with the data."

   From Bayes' rule it is clear that in order to keep consistent hypotheses what is needed is that we consider all hypotheses h s.t. $p(h) > 0$.

2. **Occam's razor**:

   "Among all hypotheses consistent with the observations, choose the simplest."

   In terms of a prior distribution over hypotheses, this is the same as giving simpler hypotheses higher a priori probability, and more complex ones lower probability.

Universal Probability and Bayesian Inference

- Using Turing's model of universal computation, Solomonoff (1964) produced a universal prior distribution that unifies these two principles.

  This is exactly the concept of **universal probability** we just studied.

- Assign to each hypotheses h the prior probability:

$$p_{\mathcal{U}}(h) \approx 2^{-K(h)}.$$

- Example 6 What would that mean in terms of estimation of prior probabilities for our hypotheses considered in the previous example?

  Solution.

  Homework!

Appendix: Solomonoff Induction and "the Perfect AI"

Solomonoff Induction and "the Perfect AI"

- Universal probability and algorithmic probability theory have several applications to artifical intelligence and machine learning.
- If you are interested in these fields, you may consider **Solomonoff Induction and "the Perfect AI"** as a topic for your seminar:

  `http://lesswrong.com/lw/dhg/an_intuitive_explanation_of_solomonoff_induction/`

- This topic is motivated by the observation that people disagree about things.
  1. Some say that vaccines cause autism; others say they don't.
  2. Some say everything is physical; others believe in a god.
  3. Some say that complicated financial derivatives are essential to a modern competitive economy; others think a nation's economy will do better without them.

  It's hard to know what is true!

Solomonoff Induction and "the Perfect AI"

- And it's hard to know how to figure out what is true.
  1. Some think you should assume the things you are most certain about and then deduce all other beliefs from your original beliefs.
  2. Others think you should accept at face value the most intuitive explanations of your personal experience.
  3. Others think you should generally agree with the scientific consensus until it is disproven.
- Wouldn't it be nice if finding out what's true was like baking a cake?
  - What if there was a recipe for finding out what is true?
  - All you'd have to do is follow the written instruction exactly, and after the last instruction you'd inevitably find yourself with some sweet, tasty truth!
- As it turns out, there is an exact recipe for finding truth.

  It was discovered in the 1960s: It's **Solomonoff Induction**!

  But... it is uncomputable!