Measures of Complexity: Computation Based

Ryan G. James

Complexity Sciences Center
UCDAVIS
UNIVERSITY OF CALIFORNIA

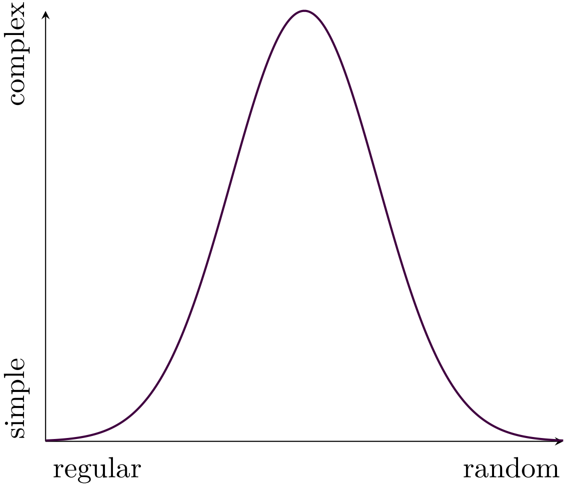
June 19, 2018

What is Complexity?

If you can't measure something, you can't understand it.

H. James Harrington

Humpology: A Rope of Sand



Universal Turing Machines

A *universal turing machine* is an finite-state controller with four tapes:

- a one-way, read-only *program* tape
- a one-way, read-only *data* tape
- a read-write *working* tape
- a one-way, write-only *output* tape

Universal Turing Machines

A *universal turing machine* is an finite-state controller with four tapes:

- a one-way, read-only *program* tape
- a one-way, read-only *data* tape
- a read-write *working* tape
- a one-way, write-only *output* tape

We write $\mathrm{U}(p, d) = x$ to mean that when p is placed on the program tape and d is placed on the data tape, x is written to the output tape after the machine has halted.

Universal Turing Machines

A *universal turing machine* is an finite-state controller with four tapes:

- a one-way, read-only *program* tape
- a one-way, read-only *data* tape
- a read-write *working* tape
- a one-way, write-only *output* tape

If d is empty, we simply write $\mathrm{U}(p) = x$.

We write $\mathrm{U}(p, d) = x$ to mean that when p is placed on the program tape and d is placed on the data tape, x is written to the output tape after the machine has halted.

Universal Turing Machines

A *universal turing machine* is an finite-state controller with four tapes:

- a one-way, read-only *program* tape
- a one-way, read-only *data* tape
- a read-write *working* tape
- a one-way, write-only *output* tape

If d is empty, we simply write $\mathrm{U}(p) = x$.

We write $\mathrm{U}(p, d) = x$ to mean that when p is placed on the program tape and d is placed on the data tape, x is written to the output tape after the machine has halted.

A universal Turing machine U_1 is universal in the sense that:

$$\forall\ \mathrm{U}_2 \quad \exists\ w\ s.t.\quad \mathrm{U}_1(wp, d) = \mathrm{U}_2(p, d)$$

Universal Turing Machines

A *universal turing machine* is an finite-state controller with four tapes:

- a one-way, read-only *program* tape
- a one-way, read-only *data* tape
- a read-write *working* tape
- a one-way, write-only *output* tape

If d is empty, we simply write $\mathrm{U}(p) = x$.

Let time(p) denote the number of steps the Turing machine runs before halting.

We write $\mathrm{U}(p, d) = x$ to mean that when p is placed on the program tape and d is placed on the data tape, x is written to the output tape after the machine has halted.

A universal Turing machine U_1 is universal in the sense that:

$$\forall\ \mathrm{U}_2 \quad \exists\ w\ s.t. \quad \mathrm{U}_1(wp, d) = \mathrm{U}_2(p, d)$$

Kolmogorov Complexity [1, 2, 3, 4, 5]

The *Kolmogorov complexity* of a string x, $\mathrm{K}(x)$, is the length of the shortest program running on universal turing machine U which outputs x:

$$x^* = \arg\min_p \{|p| : \mathrm{U}(p) = x\}$$

$$\mathrm{K}_{\mathrm{U}}(x) = |x^*|$$

Kolmogorov Complexity [1, 2, 3, 4, 5]

The *Kolmogorov complexity* of a string x, $\mathrm{K}(x)$, is the length of the shortest program running on universal turing machine U which outputs x:

$$x^* = \arg\min_p \{|p| : \mathrm{U}(p) = x\}$$

$$\mathrm{K}_\mathrm{U}(x) = |x^*|$$

And conditionally:

$$\mathrm{K}_\mathrm{U}(x \mid d) = \min_p \{|p| : \mathrm{U}(p, d) = x\}$$

Kolmogorov Complexity [1, 2, 3, 4, 5]

The *Kolmogorov complexity* of a string x, $\mathrm{K}(x)$, is the length of the shortest program running on universal turing machine U which outputs x:

$$x^* = \arg\min_p \{|p| : \mathrm{U}(p) = x\}$$
$$\mathrm{K_U}(x) = |x^*|$$

And conditionally:

$$\mathrm{K_U}(x \mid d) = \min_p \{|p| : \mathrm{U}(p, d) = x\}$$

When choice of U is clear from context, we will simply write $\mathrm{K}(x)$

Choice of U Doesn't Really Matter

$$\forall\ \mathrm{U}_1(x), \mathrm{U}_2(x)$$
$$\exists\ c \in \mathbb{Z}^+\ s.t.$$
$$\forall\ x$$
$$|\mathrm{K}_{\mathrm{U}_1}(x) - \mathrm{K}_{\mathrm{U}_2}(x)| \leq c$$

Choice of U Doesn't Really Matter

$$\forall\ \mathrm{U}_1(x)\,,\mathrm{U}_2(x)$$
$$\exists\ c \in \mathbb{Z}^+\ s.t.$$
$$\forall\ x$$
$$|\mathrm{K}_{\mathrm{U}_1}(x) - \mathrm{K}_{\mathrm{U}_2}(x)| \leq c$$

Proof:

Let $c = |w|$ from definition of universal.

$\mathrm{K}(x)$ is Bound From Above

$$\mathrm{K}(x) \leq |x| + c$$

$\mathrm{K}(x)$ is Bound From Above

$$\mathrm{K}(x) \leq |x| + c$$

```python
x = ...

def f():
    print(x)
```

so for Python, $c = 25$.

Most x are incompressible

Let S be the number of strings of length n compressible by c:

$$S = \left\{ x : \begin{array}{l} |x| = n \\ \mathrm{K}(x) \leq n - c - 1 \end{array} \right\}$$

The size of this set is bound by:

$$|S| \leq 2^{n-c} - 1$$

And therefore the percentage of strings of length n compressible by c is:

$$\frac{|S|}{2^n} \leq \frac{2^{n-c} - 1}{2^n} \leq \frac{1}{2^c}$$

Kolmogorov Complexity is Uncomputable

Assume we have a Python function `K(x)`.
Consider the following code:

```
def paradox():
    from sys import getsizeof

    N = getsizeof(K) + \
        getsizeof(paradox) + \
        getsizeof(all_strings)
    for x in all_strings():
        if K(x) > N:
            return x
```

Code Golf!

Program (575 B)

```
x="WM n=straQRsF=loB7Erules3s=d=IXA full
   commitSnt'sKhatVFhink;of7KTldn'tUetFhis fromLny9guy.-AC if?Lsk S1Don'tFP
   S?<bliCF=see//X82)8002)-.//"
i=45
for r in"XXW'BHn each9for s=loQ7r hear6ach;but7<shyF=s@InsideKe
   bothHKha6go;onXWEgaS3weM:pl@|XI justKannaFP?1Gotta >uCRstaC/|X4g24let?
   down4runLrTC3desRt?4>cry4sayUoodbye4tPL lie3hurt?|2)J)4giB, n5giBX(G|
   howV feeliQX|iB? up|LC |XN5|eBr:|t's been |XYT|J,U| othR |Uonna |iQ
   |MFo=|o |make? | yT|ay itX|A|ve|nd|D|HFhe | t|G| know|I|X(Ooh| w|
   a|'re|N|O|ell|ng|er|me|ou| g|
   I'm|We|\n".split("|"):x=x.replace(chr(i),r);i+=1
print(x)
```

Code Golf! [6]

Output (1872 B)

We're no strangers to love
You know the rules and so do I
A full commitment's what I'm thinking of
You wouldn't get this from any other guy
I just wanna tell you how I'm feeling
Gotta make you understand

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you

We've known each other for so long
Your heart's been aching but
You're too shy to say it
Inside we both know what's been going on
We know the game and we're gonna play it
And if you ask me how I'm feeling
Don't tell me you're too blind to see

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you

(Ooh, give you up)
(Ooh, give you up)
(Ooh)
Never gonna give, never gonna give
(Give you up)
(Ooh)
Never gonna give, never gonna give
(Give you up)

We've known each other for so long
Your heart's been aching but
You're too shy to say it
Inside we both know what's been going on
We know the game and we're gonna play it

I just wanna tell you how I'm feeling
Gotta make you understand

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you

What Does Kolmogorov Complexity Quantify?

What strings have large K?

What Does Kolmogorov Complexity Quantify?

What strings have large K ?

Those with no structure/regularities/patterns.

What Does Kolmogorov Complexity Quantify?

What strings have large K?

Those with no structure/regularities/patterns.

Kolmogorov Complexity more accurately quantifies randomness than complexity.

Program Length is the Wrong Metric

Just as the plausibility a scientific theory depends on the economy of its assumptions, not on the length of the deductive path connecting them with observed phenomena, so a slow execution time is not evidence against the plausibility of a program; rather, if there are no comparably concise programs to compute the same output quickly, it is evidence of the nontriviality of that output.

Charles H. Bennett

Logical Depth [7]

Logical Depth is the fastest running time among all "reasonably" optimal programs:

$$\mathrm{depth}_{\mathrm{c}}(x) = \min_{p} \left\{ \mathrm{time}(p) : \begin{array}{l} \mathrm{U}(p) = x \\ |p| \leq \mathrm{K}(x) + c \end{array} \right\}$$

Physical Complexity

The depth of a crystal is small:

```
def crystal():
    print('01' * 500_000)
```

Physical Complexity

The depth of a crystal is small:

```python
def crystal():
    print('01' * 500_000)
```

The depth of a gas is small:

```python
x = ...

def f():
    print(x)
```

Physical Complexity

The depth of a crystal is small:

```
def crystal():
    print('01' * 500_000)
```

The depth of a gas is small:

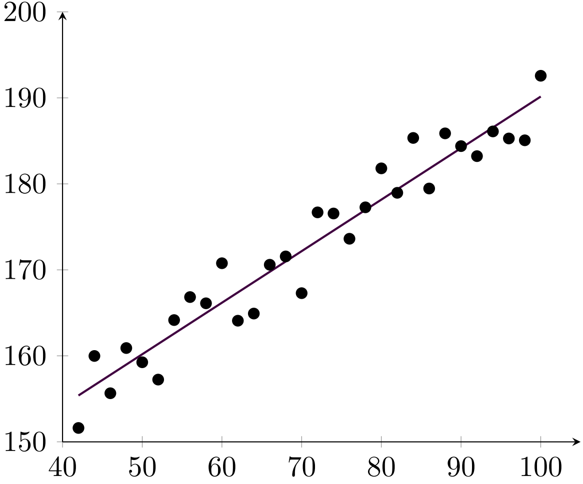
```
x = ...

def f():
    print(x)
```

The depth of π is large:

```
from math import sqrt

def pi():
    return sqrt(6*sum(1/n**2 for n in range(1,
   1_000_000)))
```


200
190
180
170
160
150
40
50
60
70
80
90
100

Sophistication [8]

Among all general model and data pairs which are optimal for x, sophistication is the smallest model:

$$\mathrm{soph_c}(x) = \min_{p,d} \left\{ |p| : \begin{array}{l} \mathrm{U}(p,d) = x \\ |p| + |d| \leq \mathrm{K}(x) + c \\ \mathrm{U}(p,d) \text{ is defined for all } d \end{array} \right\}$$

Static vs. Dynamic

Logical Depth and Sophistication are somewhat equivalent, at least for infinite strings:

$$\exists\, c \quad \forall\, x : \quad \begin{cases} \mathrm{depth_c}(x) = \mathrm{soph_c}(x) = \infty \quad \text{or} \\ |\mathrm{depth_c}(x) - \mathrm{soph_c}(x)| < c \end{cases}$$

Summary

- Kolmogorov Complexity is the length of the shortest program producing x
- Kolmogorov Complexity quantifies how random x is
- Logical Depth quantifies how long it takes to produce x given a good (short) program for it
- Sophistication quantifies the "essential" regularities of x
- Both Logical Depth and Sophistication are closer to intuitive notions of complexity
- All these quantities are uncomputable, but of philosophical interest

Preview

- How are Algorithmic Information Theory and (Shannon) Information Theory related?
- What is the complexity of a distribution of strings/time series?
- What is the complexity of an unstructured distribution?
- How do we quantify shared information?
- Are there different kinds of shared information?

References I

[1] Andrei Nikolaevich Kolmogorov. "Three approaches to the quantitative definition of information". In: *International journal of computer mathematics* 2.1-4 (1968), pp. 157–168.

[2] Gregory J Chaitin. "On the simplicity and speed of programs for computing infinite sets of natural numbers". In: *Journal of the ACM (JACM)* 16.3 (1969), pp. 407–422.

[3] Ray J Solomonoff. "A formal theory of inductive inference. Part I". In: *Information and control* 7.1 (1964), pp. 1–22.

[4] Ray J Solomonoff. "A formal theory of inductive inference. Part II". In: *Information and control* 7.2 (1964), pp. 224–254.

[5] Li Ming and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Heidelberg, 1997.

References II

[6] Programming Puzzles & Code Golf. *We're no strangers to code golf, you know the rules, and so do I.* URL: https://codegolf.stackexchange.com/questions/6043/were-no-strangers-to-code-golf-you-know-the-rules-and-so-do-i (visited on 06/13/2018).

[7] Charles H Bennett. "Logical depth and physical complexity". In: *The Universal Turing Machine A Half-Century Survey* (1995), pp. 207–235.

[8] Moshe Koppel. "Complexity, depth, and sophistication". In: *Complex Systems* 1.6 (1987), pp. 1087–1091.

Levin Complexity

Levin Complexity considers both program size and running time:

$$\mathrm{L}(x) = \min_{p} \left\{ |p| + \log_2 \left(\mathrm{time}(p) \right) : \mathrm{U}(p) = x \right\}$$

Levin Complexity

Levin Complexity considers both program size and running time:

$$\mathrm{L}(x) = \min_{p} \{|p| + \log_2 (\mathrm{time}(p)) : \mathrm{U}(p) = x\}$$

What does Levin bring to the table?

- computable!
- a component of Universal Search, which is optimal for any problem (up to monstrously huge multiplicative constant)

Gennerating All Strings/Programs

```
from itertools import count,
    product

def all_strings():
    for length in count():
        for word in product('01',
    repeat=length):
            yield ''.join(word)
```