What does it mean for a number to be random?

Tristan Hodgson

15 Oct 2025

Motivation

- It is obvious that some sequences seem more random than others
  - $x = 10101010101010101010101010101010$
  - $y = 11001111110101000001101110010001$
- Yet it is not immediately obvious how we would formalise this notion

A first idea

- Perhaps one approach could be to use a form of Run Length Encoding (RLE)
- x now becomes `"10"*16`, or in words: the sequence 10 repeated 16 times
- This is nice because we can now tell apart random numbers from non-random numbers by simply using an encoding method to compress all the repetitive sections

I lied

$$
\begin{aligned}
y &= 11001111110101000001101110010001_2 \\
&= 3,486,784,401_{10} \\
&= 3^{20} \\
&= 10000000000000000000_3
\end{aligned}
$$

So our approach needs some work as we just characterised 3^{20} as random when clearly it isn't

Another Approach?

- Our RLE based method effectively used a compression algorithm to define randomness
- Our new approach is largely the same but instead of restricting ourselves to just RLE we allow any lossless compression algorithm
- To formalise this, we require the idea of Kolmogorov Complexity

Definition

Kolmogorov complexity, $K(x)$ is the length of the shortest possible computer program that can produce a specific piece of data, x as output.

Example

```
>>> print("1"*32)
11111111111111111111111111111111
```

Note this case is $O(\log \log x)$

```
>>> print("01010111111010000011001111111000")
01010111111010000011001111111000
```

And that an $O(\log x)$ case is always possible

y has now been replaced by a random number (generated from atmospheric noise)

Randomness

We define randomness similarly to before: if $K(x) \approx \log x$ then x is random, else x is not random. This leads to the natural question of are there any random numbers?

- Consider all programs with length $< c$ that unambiguously output one number
- Each program can be written in binary so we have at most $1 + 2 + 4 + \ldots 2^{c-1} = 2^c - 1 < 2^c$ possible programs
- Consider all numbers of length $n > c$, we note that by the pigeonhole principle the proportion of random numbers must be at least $1 - 2^{c-n}$. Hence, most numbers are random.

Computability

We do, however, have an issue: it is impossible to know when we have found the optimal program, this means that, in general, we can only bound $K(x)$ rather than be able to calculate it fully.

- Suppose a computable function $K(x)$ does exist, let L be the length of the sub-routine that finds $K(x)$
- Define $y \in \mathbb{N}$ as the smallest number with $K(y) \geq 100 + L$
- Define the following short program

```
1  def f(x):
2      return x if K(x) >= 100 + L else f(x+1)
3
4  y = f(1)
```

- This program is short and returns y, this is a contradiction as $100 + L \leq K(y)$

Applications: NID

Take the following formulas that represent the distance between any two binary strings x, y:

$$\begin{aligned}\mathrm{NID} &:= \frac{\max(K(x|y), K(y|x))}{\max(K(x), K(y))} \\ &\approx \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))} =: \mathrm{NCD}(x, y)\end{aligned}$$

The issue with NID is that it is obviously uncomputable so we use NCD as an approximation. The use of this formula allows us to, in a very general way, express the distance between any two pieces of data

Applications: NID, Example

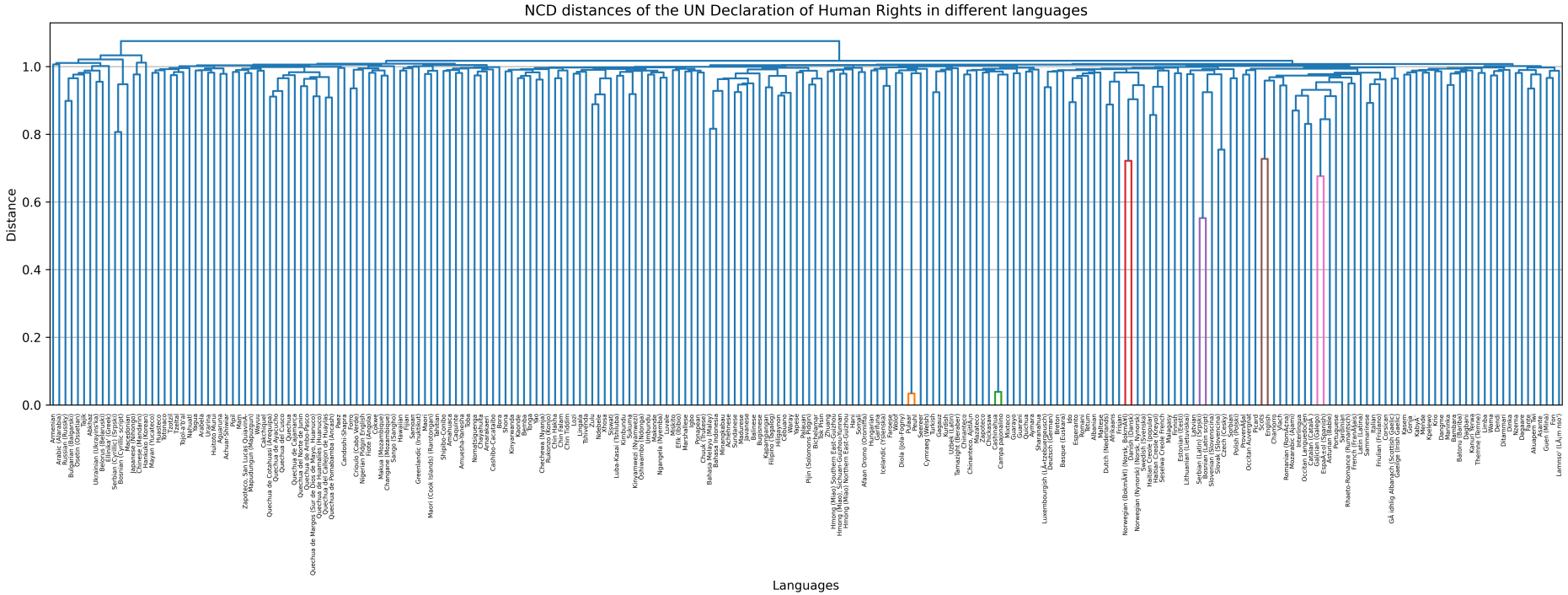

Figure: A dendrogram of NCD between translated copies of the UN Declaration of Human Right

NCD is useful for this as it doesn't require any domain specific knowledge or special skill, I wrote the code for this in a couple of hours with relatively little in the way of skill or knowledge.

Applications: Occam's Razor

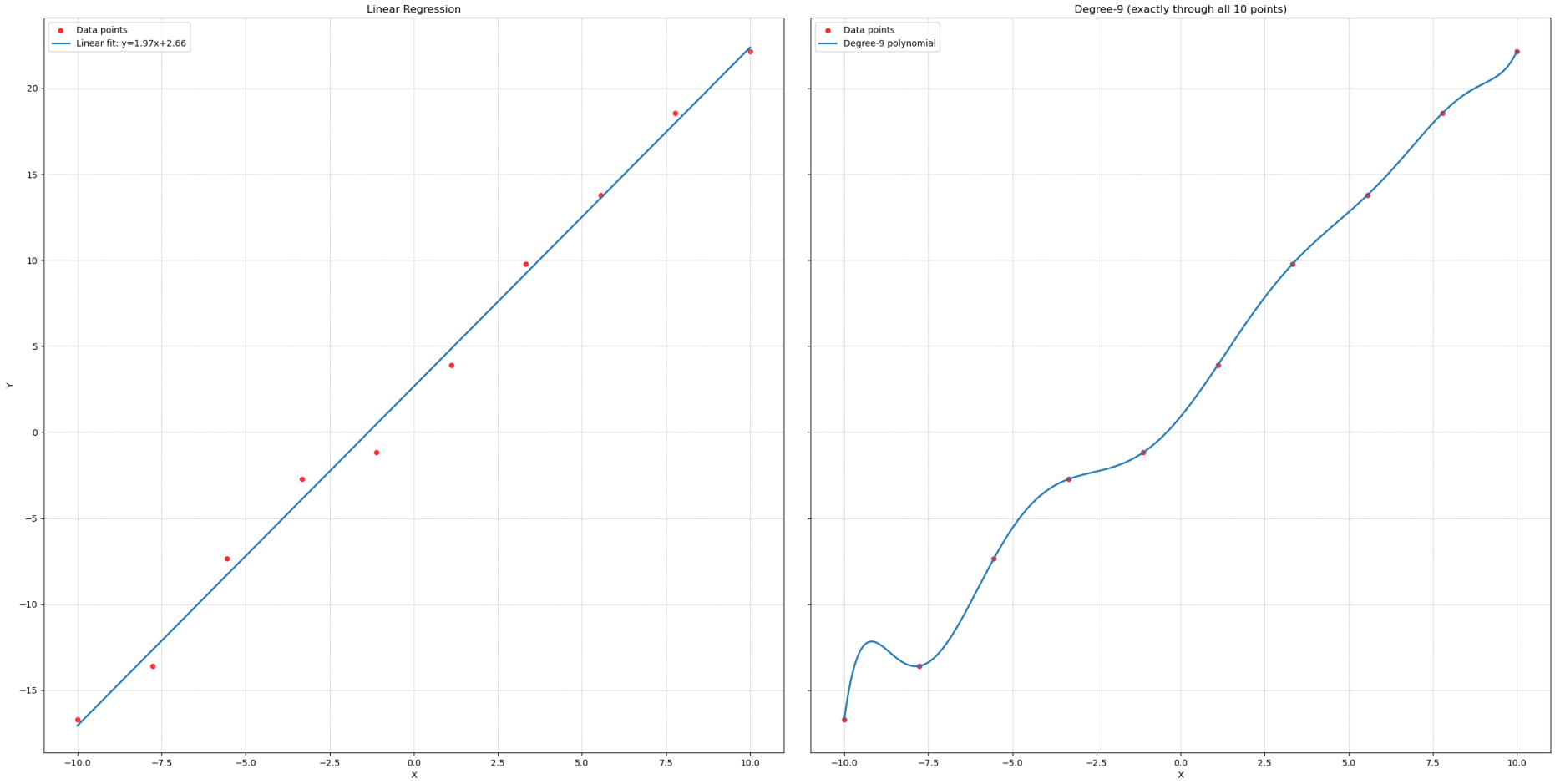

Figure: Linear model vs degree 9 polynomial as a method of predicted data

Ideal MDL finds the model that minimises $K(\text{Model}) + K(\text{Data}|\text{Model})$.

Questions?

Thank you for your attention

Any Questions?

Optimality

Our definition of $K(x)$ used the idea of the smallest possible program, so we need to check that this is well-defined

- Consider S=$\{|$programs that produce $x|\}$ (i.e. S is the set containing the lengths of all programs that produce x)
- $S \subseteq \mathbb{N}$
- S is non-empty as we can always have our $O(\log x)$ solution

Well ordering property

Every non-empty subset of the natural numbers has a least element

The Invariance Theorem

- In all the code I've shown you so far we have used Python for its relative simple syntax and ubiquity, however our definition would be somewhat useless if the same number could be random wrt one language but not another
- If we have any two programming languages (formally any two Universal Turing Machines, which includes all programming languages but also some unexpected things such as MS PowerPoint transitions and Minecraft Redstone) then we can write a compiler for the first language in the second
- Hence, the complexity of a string differs by at most a constant when we consider different languages
- Hence we consider the asymptotic growth of $K(x)$ rather than its specific value

Applications: NID, Proof that $\mathrm{NID} \approx \mathrm{NCD}$

- First note that $\forall x \in \{0,1\}^* \quad K(x) \approx C(x)$ assuming a reasonably good compressor, this fairly easily gives us the denominator
- Turning our attention to the numerator, $K(x|y) \approx K(xy) - K(y)$. This makes sense intuitively because if I gave you an algorithm to produce x given y and an algorithm to produce y then you could fairly reasonably produce xy.
- $C(xy) = C(yx)$ for many compressors (because most compression algorithms involve block-coding which is symmetric by definition)

$$\begin{aligned}\max(K(x|y), K(y|x)) &\approx \max(K(xy) - K(y), K(yx) - K(x)) \\ &\approx \max(C(xy) - C(y), C(yx) - C(x)) \\ &= C(xy) - \min(C(x), C(y))\end{aligned}$$

Berry Paradox

Berry Paradox

The smallest positive integer not definable in under sixty letters (a phrase with fifty-seven letters).

Assume that there are n symbols in the alphabet, then there are $\approx n^{61}$ phrases with 60 symbols or fewer. $n^{61} \ll \infty$ so there definitely exists such a number, yet we managed to define it in fewer than 60 symbols.

Citations

- Li, Ming. An Introduction to Kolmogorov Complexity and Its Applications. Ed. by Paul Vitányi. 4th ed. 2019. Cham: Springer Nature, 2019. Web.
- Nannen, Volker. "A Short Introduction to Kolmogorov Complexity." (2010): n. pag. Web.
- Cilibrasi, Rudi, and Paul Vitanyi. "Clustering by Compression." (2003): n. pag. Web.
- Cover, Thomas M, and Joy A Thomas. "Kolmogorov Complexity." Elements of Information Theory. Ed. by Thomas M Cover and Joy A Thomas. United States: Wiley, 2006. 463–508. Web.
- Cover, T. M, and Joy A Thomas. Elements of Information Theory. 2nd ed. Hoboken, N.J: Wiley-Interscience, 2006. Print.
- Peter Grünwald, 2005. "Introducing the Minimum Description Length Principle", Advances in Minimum Description Length: Theory and Applications, Peter D. Grünwald, Jay Injae Myung, Mark A. Pitt
- Vitanyi, P.M.B, and Ming Li. "Minimum Description Length Induction, Bayesianism, and Kolmogorov Complexity." IEEE transactions on information theory 46.2 (2000): 446–464. Web.
- Gleick, James. The Information: A History, a Theory, a Flood. Fourth Estate paperback edition. London: Fourth Estate, 2012. Print.