Information Theory and Networks

Lecture 20: Kolmogorov Complexity

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/Lecture_notes/InformationTheory/

School of Mathematical Sciences,
University of Adelaide

September 18, 2013

Part I

Kolmogorov Complexity

Clutter and confusion are failures of design, not attributes of information.

Edward Tuft

Formal Kolmogorov Complexity

Definition (Kolmogorov Complexity)

The Kolmogorov complexity $K_{\mathcal{U}}(\mathbf{x})$ of a string $\mathbf{x}$ with respect to a universal computer $\mathcal{U}$ is defined as

$$K_{\mathcal{U}}(\mathbf{x}) = \min_{\{\mathbf{p} | \mathcal{U}(\mathbf{p}) = \mathbf{x}\}} \ell(\mathbf{p})$$

So we are

- minimising the length $\ell(\mathbf{p})$ of the input $\mathbf{p}$
- such that the output $\mathcal{U}(\mathbf{p}) = \mathbf{x}$
- and then it halts

Universality

Theorem

If $\mathcal{U}$ is a universal computer, then for any other computer $\mathcal{A}$

$$K_{\mathcal{U}}(\mathbf{x}) \leq K_{\mathcal{A}}(\mathbf{x}) + c_{\mathcal{A}}$$

for all strings $\mathbf{x} \in \{0,1\}^*$, where the constant $c_{\mathcal{A}}$ doesn't depend on $\mathbf{x}$.

- this says that all universal computers are equivalent (from the point of view of Kolmogorov complexity) up to a constant.
- so the details don't matter (too) much
  - the constant might be quite large
- so we normally drop any mention of the actual machine in the definition of complexity

Universality

Proof.

Assume program $\mathbf{p}_{\mathcal{A}}$ for computer $\mathcal{A}$ prints $\mathbf{x}$, i.e., $\mathcal{A}(\mathbf{p}_{\mathcal{A}}) = \mathbf{x}$.

A $\mathcal{U}$ is a universal computer we can write a simulator for $\mathcal{A}$ in $\mathcal{U}$, call it $\mathbf{s}_{\mathcal{A}}$.

So the program $\mathbf{s}_{\mathcal{A}}\,\mathbf{p}_{\mathcal{A}}$, input to $\mathcal{U}$ will simulate the output $\mathcal{A}(\mathbf{p}_{\mathcal{A}})$, i.e., the desired output.

The length of this program is

$$\ell(\mathbf{s}_{\mathcal{A}}\,\mathbf{p}_{\mathcal{A}}) = \ell(\mathbf{s}_{\mathcal{A}}) + \ell(\mathbf{p}_{\mathcal{A}})$$

where $\ell(\mathbf{s}_{\mathcal{A}}) = c_{\mathcal{A}}$ is constant with respect to $\mathbf{x}$.

The Kolmogorov complexity is the minimum over such programs, and so it becomes an inequality, because there might be a better way to generate the same sequence.

□

Examples

- An integer n (written in binary) has

$$K(n) \leq 2\log_2 n + c$$

  To describe n, repeat every bit of the binary expansion of n twice; then end the description with a 01.
  Example:
  - $n = 5$, which in binary is 101
  - write as $11, 00, 11, 01$
- The first n digits of π
  - we know a program to generate digits
  - we also need to let it know how many to generate

Conditional Kolmogorov Complexity

Definition (Conditional Kolmogorov Complexity)

The Kolmogorov complexity $K_{\mathcal{U}}(\mathbf{x})$ of a string $\mathbf{x}$ with respect to a universal computer $\mathcal{U}$, assuming the computer knows the length $\ell(\mathbf{x})$ is defined as

$$K_{\mathcal{U}}(\mathbf{x}|\ell(\mathbf{x})) = \min_{\{\mathbf{p}|\mathcal{U}(\mathbf{p},\ell(\mathbf{x}))=\mathbf{x}\}} \ell(\mathbf{p})$$

- this is the shortest program given the computer knows the length of the output
- the subtlety is that if it knows $\ell(\mathbf{x})$ then it knows when to stop, without any extra computation
- we'll usually just write something like $K(\mathbf{x}|y)$

Examples

- $K(0000\ldots0|\ell) = c$ for all ℓ
  `Print` ℓ `zeros`
  Similar for any simple repeated sequence.
- $K(\pi_1\pi_2\ldots\pi_\ell|\ell) = c$ for all ℓ
  We know (short) constant length programs to output the digits of π, given we know how many to output.
- $K(image|\ell) \leq \ell/3 + c$
  Use standard compression algorithms, which can probably compress it by about a factor of 3, without any loss.
- A sequence with n bits and k ones?

Bounds 1

Theorem

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

Proof.

Intuitively, we just write a program that says

`Print the following` ℓ`-bit sequence` $x_1 x_2 \ldots x_\ell$

No bits are needed for ℓ as that is given. □

Bounds 2

Theorem

$$K(\mathbf{x}) \leq K(\mathbf{x}|\ell(\mathbf{x})) + 2\log \ell(\mathbf{x}) + c$$

Proof.

If the computer doesn't know $\ell(\mathbf{x})$ it needs some way to know to halt, i.e., to know that it has reached the end of the sequence.

Suppose $\ell(\mathbf{x}) = n$. To describe $\ell(\mathbf{x})$, repeat every bit of the binary expansion of n twice; then end the description with a 01 (as in earlier example). So including the length in the program only takes $2\log(n) + c$ bits.

Then we just use the complexity given we know $\ell(\mathbf{x})$. □

Bounds 3

Theorem

The number of strings with complexity $K(\mathbf{x}) < k$ satisfies

$$\left| \left\{ \mathbf{x} \in \{0,1\}^* \mid K(\mathbf{x}) < k \right\} \right| < 2^k$$

Proof.

List all of the (binary) programs i, and we get 2^i.

Add up all the programs shorter than k and we get

$$\sum_{i=0}^{k-1} 2^i = 2^k - 1 < 2^k$$

Since each program can produce only one output sequence, the number of sequences with complexity $< k$ is $< 2^k$. □

Theorem

$$\frac{1}{n+1} 2^{nH(k/n)} \leq \binom{n}{k} \leq 2^{nH(k/n)}$$

Proof.

Stirling's approximation

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Combinations

$$\begin{aligned} \binom{n}{k} &= \frac{n!}{k!(n-k)!} \\ &\sim \sqrt{\frac{n}{2\pi k(n-k)}} \left(\frac{n}{k}\right)^k \left(\frac{n}{n-k}\right)^{n-k} \end{aligned}$$

□

Proof.

And

$$\begin{aligned} 2^{nH(k/n)} &= 2^{-k\log_2(k/n)-(n-k)\log_2(1-k/n)} \\ &= 2^{k\log_2(n/k)}\, 2^{(n-k)\log_2(n/(n-k))} \\ &= \left(\frac{n}{k}\right)^k \left(\frac{n}{n-k}\right)^{n-k} \end{aligned}$$

Also for $k = 1, \ldots, n-1$ the term $\sqrt{\frac{n}{2\pi k(n-k)}}$ takes its minimum value for $k = n/2$

$$\sqrt{\frac{n}{2\pi k(n-k)}} = \sqrt{\frac{2}{\pi n}} \geq \frac{1}{n+1}$$

and maximum for $k = 1$, so

$$\sqrt{\frac{n}{2\pi k(n-k)}} = \sqrt{\frac{n}{2\pi (n-1)}} \leq 1$$

□

Example

Can we compress a sequence of n bits with k ones?

- earlier result was there is no universal compression, so we might guess no
- but the problem is subtlety different

Use the following program:

```
Generate, in lexicographic order,
             all sequences with k ones;
Of these, print the ith
```

- The program has fixed length
- We need to specify
  - k which has range $0, \ldots, n$
  - i has conditional range $\binom{n}{k}$

Example

Use the following program:

```
Generate, in lexicographic order,
                all sequences with k ones;
Of these, print the ith
```

The length of the above is

$$\ell(p) = c + 2\log_2(k) + \log_2 \binom{n}{k}$$

- The program has fixed length c_0 bits
- We need to specify
  - k which takes $2\log_2(k) + c_1$ bits
  - i which takes up to $\log_2 \binom{n}{k} + c_2$ bits
    - worst case is that $i = \binom{n}{k}$

Example

Theorem

The Kolmogorov complexity of a binary string $\mathbf{x}$ *with* k *ones is bounded by*

$$K(x_1 x_2 \ldots x_n | n) \leq nH\left(\frac{k}{n}\right) + 2\log n + c$$

Proof.

Use the program from the last example, and note that $k \leq n$ and (from result above)

$$\log_2 \binom{n}{k} \leq nH\left(\frac{k}{n}\right)$$

□

Incomputability

Theorem

The Kolmogorov complexity $K(\mathbf{x})$ is not a computable function (i.e., no program with input $\mathbf{x}$ produces $K(\mathbf{x})$ as output).

Proof.

Imagine such a program exists. Now consider the function

```
function GenerateComplexString(n);
input: Integer n.
output: A string s with complexity K(s) at least n.
for i = 1 to ∞ do
    foreach string s of length exactly i do
        if K(s) ≥ n then
            return s
        end
    end
end
```

Incomputability

Proof.

Comments about GenerateComplexString(n)

- There is always at least one string with complexity $\geq n$, otherwise all possible strings could be generated a program of length n.
- So GenerateComplexString(n) always halts (and returns a string with complexity at least n)
- GenerateComplexString(n) has fixed length U, with input n (which we can give with $2\log_2 n$ bits).

□

Incomputability

Proof.

Now define

```
function GeneratePardoxialString;
output: A string s with complexity K(s) at least n_0.
return GenerateComplexString(n_0)
```

- The length of GeneratePardoxialString is at most

$$U + 2\log_2(n_0) + c$$

- Since n grows faster than $\log_2 n$, there must be a value n_0 such that

$$U + 2\log_2(n_0) + c < n_0$$

But that means there is a function to generate s, whose length is less than n_0, but the function GenerateComplexString(n_0), created a string s whose complexity was at least n_0. Hence we have a contradiction. □

Berry Paradox

> The smallest positive integer not definable in under eleven words.
>
> *G.G.Berry (1867-1928)*

Think about it:

- There are a finite number of words, and hence finite number of sentences with less than eleven words.
- Hence a finite number of positive integers describable, and hence an infinite number that aren't.
- By well ordering property of integers, there is therefore a least such integer.
- But the above description is 10 words, and hence, it is defined with under 11 words.
- Thus it no longer is described by the words, so it isn't ...

This leads to Chaitin, Gödel and Escher

Further reading I

Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, John Wiley and Sons, 1991.

William Feller, *An introduction to probability theory and its applications*, second ed., vol. I, John Wiley and Sons, New York, 1971.