Information-Theoretic Modeling

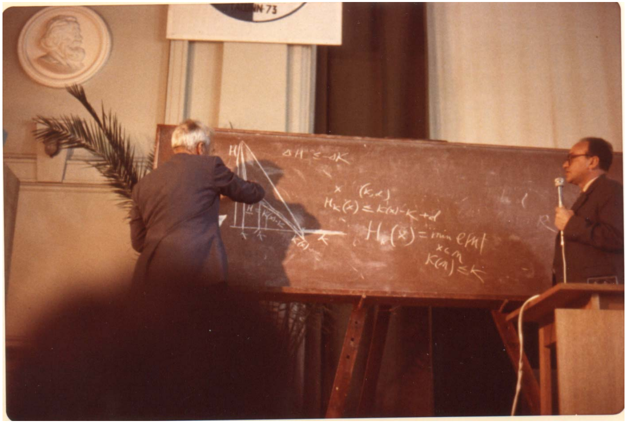
Lecture 13: Kolmogorov Complexity

Teemu Roos

Department of Computer Science, University of Helsinki

Fall 2014

Lecture 13: Kolmogorov Complexity


TALLINN-73
$\Delta H \leq -\Delta K$
$H_k(x) \leq K(x) - K + d$
$H_k(x) = \min_{\substack{x \in A \\ K(A) \leq K}} \log |A|$

Andrej Kolmogorov giving a talk in Tallinn, 1973
(Photo: Terrence L. Fine, Licenced under CC BY 2.5)

Kolmogorov Complexity

We probably agree that the string

$$\underbrace{\texttt{101010101010101010\ldots10}}_{\text{10 million characters}}$$

is “simple”.

Why?

(One) Solution: The string has a short description:

“10 repeated 5 000 000 times”.

Remark: “Description” should be understood to mean a code that can be decoded by some algorithm (a formal procedure that halts).

Turing machines

To precisely define Kolmogorov complexity we need to fix a formal notion of algorithm.

Following tradition, we use here Turing Machine (TM), but any other universal model of computation could be used as well. For simplicity, we assume that the inputs and outputs are strings over the binary alphabet $\{ 0, 1 \}$.

If TM U on input $p \in \{ 0, 1 \}^*$ halts and outputs $x \in \{ 0, 1 \}^*$, we write $U(p) = x$.
If U does not halt on input p, we say that $U(p)$ in undefined and write $U(p) = \emptyset$.
We use $| \, p \, |$ to denote the length of string x.

Kolmogorov Complexity

Kolmogorov Complexity

The **Kolmogorov complexity** of string $x \in \{0,1\}^*$ with respect to a Turing machine U is defined as the length of shortest input on which U outputs x:

$$K_U(x) = \min \{\, |p| \mid U(p) = x \,\} \ .$$

Notice that $K_U(x)$ depends on our choice of U, which at this stage is abritrary. However, we can remove most of this dependence by limiting the set of machines U we consider.

- Turing machine U is a *prefix machine* if the set of inputs on which U halts is prefix-free.
- Turing machine U is *universal* if for any other TM V there is a string q_V such that $V(p) = U(q_v p)$ for all $p \in \{0,1\}^*$.

Here qp means concatenation of strings q and p.

Prefix property in practice

Requiring prefix property may seem a bit technical, but intuitively it just means we must be able to tell when the input ends.

In practical programming, this is done by methods such as end-of-file markers, or having the operating system keep track of file lengths.

End-of-file markers actually are a way of keeping the input set prefix free. However reserving one symbol for this special use is not generally acceptable if we are interested in optimal code lengths.

Theoretically more satisfying way to make a set of inputs prefix-free is to include the lenght of (the rest of) the input string using some prefix code.

Prefix code for integers

A straighforward prefix code for integers is the following:

Consider integer x with n bit binary representation $x_1 x_2 x_3 \dots x_n$.
We encode x as $x_1 x_1 x_2 x_2 x_3 x_3 \dots x_n x_n 01$.
We denode this code for x by $\langle x \rangle$.

For example, for $x = 19 = 10011_2$ we get $\langle x \rangle = 110000111101$.

The lenght of the prefix-free encoding is $|\langle x \rangle| = 2n + 2$ bits, where $n = \lceil \log_2(x+1) \rceil \leq \log_2 x + 1$ is the length of the original binary representation.

Prefix property for Turing Machines

We can now make the set of inputs prefix-free by inserting before each input x its length encoded as $\langle | x | \rangle$.

For example, input 1000100101101, which has 13 bits, becomes

$$\underbrace{1111001101}_{\text{code for } 13 = 1101_2} \underbrace{1000100101101}_{\text{actual input}} .$$

More generally, an input of n bits gets code length of at most $n + 2 \log_2 n + 2$ bits.

For large n this is much better than $2n + 2$ we would get by applying the prefix-free encoding from previous slide directly to x.

To summarize, the prefix property is reasonable from a practical point of view, and from a theoretical point of view can be assumed without increasing input lengths too much.

Universal Turing Machines

For any fixed x, there are Turing machines that output x with empty input, and other Turing machines that don't output x with any input.

Similarly, for any pair of different strings $x \neq y$, there are Turing machines U and V such that $K_U(x) \ll K_U(y)$ but $K_V(x) \gg K_V(y)$.

For comparisons of Kolmogorov complexities to be meaningful, we require U to be *universal*.

Universal Turing Machines

Universality

A Turing Machine U is said to be **universal**, if for *any* other Turing Machine V there is a string $q_V \in \{0, 1\}^*$ (which depends on V) such that for all strings p we have

$$U(q_V p) = V(p) \ .$$

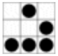
That is when given the concatenated input qp, TM U outputs the same string as TM V when given input p.

If we think of strings p as programs in the "machine language" of V, then q_v is an "interpreter" or "compiler" for V's machine language, written for machine U (in the machine language of U).

Examples

Examples of (virtually) universal ‘computers’:

1. C (compiler + operating system + computer),
2. Java (compiler + operating system + computer),
3. your favorite programming language (compiler/interpreter + OS + computer),
4. Universal Turing machine,
5. Universal recursive function,
6. Lambda calculus,
7. Arithmetics,
8. Game of Life
9. ...



Each of the above can mimic all the others.

Kolmogorov Complexity

For any *universal* computer U, and any other computer V, we have

$$K_U(x) \leq K_V(x) + C \ ,$$

where C is a constant independent of x.

Proof: Let q_V be such that $U(q_V p) = V(p)$ for all p. Let $p^*_V(x)$ be the shortest program for which $V(p^*_V(x)) = x$. Then $U(q_V p^*_V(x)) = x$, so

$$K_U(x) \leq |q_V p^*_V(x)| = |p^*_V(x)| + |q_V| = K_V(x) + |q_V| \ . \quad \Box$$

Since we are restricting ourselves to prefix machines, we don't need to worry about any overhead caused by encoding the pair (q_V, p), we can just concatenate them.

Invariance Theorem

From now on we restrict the choice of the computer U in K_U to *universal* computers.

Invariance Theorem

Kolmogorov complexity is invariant (up to an additive constant) under a change of the universal computer. In other words, for any two universal computers, U and V, there is a constant C such that

$$|K_U(x) - K_V(x)| \leq C \quad \text{for all } x \in \{0,1\}^* \ .$$

Proof: Since U is universal, we have $K_U(x) \leq K_V(x) + C_1$. Since V is universal, we have $K_V(x) \leq K_U(x) + C_2$. The theorem follows by setting $C = \max\{C_1, C_2\}$. □

Kolmogorov Complexity

Upper Bound

We have the following upper bound on $K_U(x)$:

$$K_U(x) \leq |x| + 2\log_2 |x| + C$$

for some constant C which depends on the computer U but not on the string x.

Proof: Remember that we have a prefix code where the code length for x is

$$\ell(x) = |\, x \,| + 2\log_2 |\, x \,| + 2.$$

Let V be a TM that decodes this encoding. Then $K_V(x) = \ell(x)$. Therefore, for universal U we have $K_U(x) \leq \ell(x) + C$. $\square$

Kolmogorov Complexity

Conditional Kolmogorov Complexity

The **conditional Kolmogorov complexity** is defined as the length of the shortest program to print x when y is given:

$$K_U(x \mid y) = \min \{ \, |p| \mid U(\bar{y}\, p) = x \, \} \ ,$$

where $\bar{y}$ is a prefix-encoded representation of y.

Upper Bound 2

We have the following upper bound on $K_U(x \mid |x|)$:

$$K_U(x \mid |x|) \leq |x| + C$$

for some constant C independent x.

Examples

Let $n = |x|$.

1. $K_U(0101010101\ldots01 \mid n) = C$.
   Program: `print` $n/2$ `times 01.`
2. $K_U(\pi_1\,\pi_2\,\ldots\,\pi_n \mid n) = C$.
   Program: `print the first` n `bits of` π`.`
3. $K_U(\text{English text} \mid n) \approx 1.3 \times n + C$.
   Program: `Huffman code.`
   (Entropy of English is about 1.3 bits per symbol.)
4. $K_U(\text{fractal}) = C$.
   Program: print # of iterations until $z_{n+1} = z_n^2 + c > T$.

Examples

Martin-Löf Randomness

Examples (contd.):

5. $K_U(x \mid n) \approx n$, for almost all $x \in \{0,1\}^n$.
   Proof: Upper bound $K_U(x \mid n) \leq n + C$. Lower bound by a counting argument: less than 2^{-k} of strings compressible by more than k bits (Lecture 1).

Martin-Löf Randomness

String x is said to be **Martin-Löf random** iff $K_u(x \mid n) \geq n$.

Consequence of point 5 above: An i.i.d. sequence of unbiased coin flips is with high probability Martin-Löf random.

Universal Prediction

Since the set of valid (halting) programs is required to be **prefix-free** we can consider the probability distribution p_U^n:

$$p_U^n(x) = \frac{2^{-K_U(x|n)}}{C} \;, \quad \text{where } C = \sum_{x \in \mathcal{X}^n} 2^{-K_U(x|n)}.$$

Universal Probability Distribution

The distribution p_U^n is universal in the sense that for any other computable distribution q, there is a constant $C > 0$ such that

$$p_U^n(x) \geq C\, q(x) \quad \text{for all } x \in \mathcal{X}^n.$$

Proof idea: The universal computer U can imitate the Shannon-Fano prefix code with codelengths $\left\lceil \log_2 \frac{1}{q(x)} \right\rceil$.

Universal Prediction

The universal probability distribution p_U^n is a good predictor.

This follows from the relationship between codelengths and probabilities (Kraft!):

$$\begin{aligned} K_U(x) \text{ is small } &\Rightarrow\ p_U^n(x) \text{ is large} \\ &\Rightarrow \prod_{i=1}^{n} p_U^n(x_i \mid x_1, \ldots, x_{i-1}) \text{ is large} \\ &\Rightarrow\ p_U^n(x_i \mid x_1, \ldots, x_{i-1}) \text{ is large for most } i \in \{1, \ldots, n\}, \end{aligned}$$

where x_i denotes the ith bit in string x.

Berry Paradox

The smallest integer that cannot be described in ten words?

Whatever this number is, we have just described (?) it in ten words.

The smallest uninteresting number?

Whatever this number is, it is quite interesting!

Non-computability

It is impossible to construct a general procedure (algorithm) to compute $K_U(x)$.

Non-Computability

Kolmogorov complexity $K_U : \{0,1\}^* \rightarrow \mathbb{N}$ is **non-computable**.

Proof: Assume, by way of contradiction, that it would be possible to compute $K_U(x)$. Then for any $M > 0$, the program

```
print a string $x$ for which $K_U(x) > M$.
```

would print a string with $K_U(x) > M$. A contradiction follows by letting M be larger than the Kolmogorov complexity of this program. Hence, it cannot be possible to compute $K_U(x)$. $\square$

Summary: Kolmogorov complexity and MDL

Universal codes (or models) in MDL were defined to be universal *with respect to* some model class $\mathcal{M}$, which from an application point of view is “user-specified”.

There are universal codes with respect to quite general model classes (such as Lempel-Ziv for finite-order Markov models), but still this may feel a bit unsatisfactory from a philosophical point of view.

Kolmogorov complexity gives a code that is universal with respect to any computable model class, which seems the best we can hope for.

Summary: Kolmogorov complexity and MDL

Unfortunately Kolmogorov complexity itself is not computable, limiting its applicability in practive. However Kolmogorov complexity is useful as an idealization and for understanding our limitations.

One should also remember that even in principle, Kolmogorov complexity is defined only up to an additive constant (depending on the choice of U).

Last Slide

The End.