Kolmogorov complexity*

In this section we study how we can define the amount of information in a bitstring. Consider the following strings:

0101010101010101010101010101010101010101010101010101010101010101010101
0100011011000001010011100101110111000000010010001101000101011001111000
1100001100111101001100010111110011100111111111001111100110011010110101100

The top string can be described as "repeat '01' 35 times". This string has little information. The second string is less random then it might look, it is obtained from the concatenation of all strings in lexicographic order

0 1 00 01 10 11 000 ...

For all $k \geq 1$ all blocks of length k appear in this sequence with equal frequency. Despite this, the sequence is highly predictable and can be generated by a small program. The last string is obtained from www.random.org/bytes. It seems to have no clear pattern and a program that prints the string needs to store all its bits individually. In some sense, this string contains a lot of information.

The mathematical measure for this type of information in a bit string is Kolmogorov complexity. It is defined relative to a Turing machine and is given by the minimal length of a program for the machine that prints the string. Kolmogorov complexity is sometimes used in computability theory and computational complexity to formulate diagonalization arguments in a more elegant way. A second area of applications is to formulate rather philosophical principles and concepts from machine learning, such as the minimal description length principle and the information distance. Some of you will learn more about the these topics in the course on machine learning. The goal of this small chapter is twofold: provide some background for that course and to illustrate several concepts from the chapters on computability and complexity theory.

Notation: For a bitstring x, let $|x|$ be the length. n-bit strings are strings of length n. We use asymptotic notation to compare the growth of functions f and g with the same domain. For functions f and g on natural numbers, we write $f \leq O(g)$ and $f(n) \leq O(g(n))$ if there exist constants c and e such that $f(n) \leq cg(n)$ for all $n \geq e$. Similar for functions with more arguments. $f \leq g + O(h)$ means $f - g \leq O(h)$. $f = g + O(h)$ means $f \leq g + O(h)$ and $g \leq f + O(h)$. For example, $3n^2 + 5n \leq O(n^2)$, and $(n+5)^3 = n^3 + O(n^2)$.

*Bruno Bauwens, Theoretical Computer Science, 2016-2017, National Research University Higher School of Economics, January 23, 2017.

1 Definition and elementary properties

Definition 1. *The* Kolmogorov complexity *of a string x relative to a Turing machine U is*

$$C_U(x) = \min\{|p| : U(p) = x\}.$$

This definition depends on the machine U. However, we can chose machines that minimize this function within an additive constant.

Definition 2. *A Turing machine U is* optimal *if for every other machine V there exists a c such that $C_U \leq C_V + c$.*

The existence of optimal machines is a direct consequence of the existence of optimal Gödel machines: if $U(wp) = V(p)$ for all p, then it follows that $C_U(x) \leq C_V(x) + |w|$. Let us illustrate these definitions by proving some inequalities.

$$\textit{If } U \textit{ is optimal, then } C_U(x) \leq |x| + O(1).$$

Let V be a machine that implements the identity function $V(p) = p$. Hence, $C_V(x) \leq |x|$. In fact, $C_V(x) = |x|$. By definition of optimality we have $C_U(x) \leq C_V(x) + O(1) \leq |x| + O(1)$.

Note that for two optimal machines U and V, we have that $C_U = C_V + O(1)$. Notation: *If a mathematical statement with some terms C_U is true for all optimal machines U, we drop the index U.* For example, $C(x) \leq |x| + O(1)$. As soon as an inequality holds for an optimal machine within an additive $O(1)$ constant, we can drop the index U in $C_U(\cdot)$. As usual in theoretical computer science, let $\log = \log_2$.

Some strings have small complexity:

$$C(\overbrace{00\ldots0}^{n}) \leq \log n + O(1).$$

Indeed, there exists a machine V that transforms n in binary to the sequence of n zeros, and hence $C_V(00\ldots0) \leq \log n$. Therefore the equation above holds (for any optimal machine).

Let $C(n)$ be the complexity of $\overbrace{00\ldots0}^{n}$. Thus $C(n) \leq \log n + O(1)$.

Exercise 1. Show that:

- $C(C(x)) \leq \log |x| + O(1)$,
- $C(xx) = C(x) + O(1)$,
- if x is a palindrome, then $C(x) \leq |x|/2 + O(1)$.

Lemma 3. *For all partial computable $f : \{0,1\}^* \to \{0,1\}^*$ there exists a constant c_f such that*

$$C(f(x)) \leq C(x) + c_f$$

for all x on which f is defined.

Proof. Fix an optimal U. We need to prove that $C_U(f(x)) \leq C_U(x) + c_f$ for some c_f. We construct a machine V that first simulates a program p on U and if this simulation terminates, computes f on $U(p)$. This shows that $C_V(f(x)) \leq C_U(x)$. By optimality of U, the left hand side increases at most a constant after replacing V by U. Hence we obtain the equation of the lemma. □

Exercise 2. Show that $C(x) \leq C(y) + O(\log |x|)$ for all x and y of equal length that differ by at most one bit, i.e., $x_i \neq y_i$ for at most one $i \leq |x|$.

Exercise 3. Show that every infinite recognizable set contains infinitely many strings x with $C(x) \leq \log |x| + O(1)$.

For every machine U and for every length n, there exists an n-bit string x with $C_U(x) \geq n$. Indeed, the number of binary programs of length less than n is $2^0 + 2^1 + \cdots + 2^{n-1} = 2^n - 1$. On the other hand, there are 2^n strings of length n, hence, at least one string has complexity at least n.

Remark: From this and the previous exercise we obtain an alternative proof of Gödel's incompleteness theorem. For some optimal machine U, consider statements of the form "$C_U(x) \geq |x|$" for all x. These statements can be expressed as an arithmetical formula. For all lengths there is an x for which the statement and hence the formula is true. Consider the set of all x for which the formula for "$C_U(x) \geq |x|$" has a proof. This set is recognizable; because on input x, a machine can search for a proof. By the exercise above, there are infinitely many x for which $C_U(x) \leq \log |x|$. Hence, for large x, either the proof system proves a false formula, or it is unable to prove some true formula.

Exercise 4. Show that for all $n \geq c$, the fraction of n-bit strings with complexity less than $n - c$ is less than 2^{-c}. Thus, most strings have complexity close to their length.

Exercise 5. For all n, let P_n be the uniform distribution on the set of all strings of length n. Show that $\mathrm{E}_{x \sim P_n}[C(x)] = n + O(\log n)$.

2 Kolmogorov complexity is not computable

Theorem 4. *For every optimal Turing machine U, the function $x \mapsto C_U(x)$ is not computable.*

Proof. The proof is an example of Berry's paradox. Consider the following number:

The smallest number that has no description in less than 20 words.

If this number is well defined, then it has a description of 12 words... Let

$$f(n) = \min \{m : C(m) > n\}.$$

Thus by definition $C(f(n)) > n$. If C is computable, then also f is computable. Hence, by Lemma 3, $C(f(n)) \leq C(n) + O(1) \leq \log n + O(1)$, and this contradicts $C(f(n)) > n$ for large n. Our assumption must be false, i.e., C is not computable. □

Definition 5. *A function $f : \{0,1\}^* \to \mathbb{N}$ is* upper semicomputable *if the set $\{(x,k) : f(x) \leq k\}$ is recognizable.*

Exercise 6. A function f is upper semicomputable if and only if there exists a computable function $g : \{0,1\}^* \times \mathbb{N} \to \mathbb{N}$ that is non-increasing in its second argument such that

$$f(x) = \lim_{t\to\infty} g(x,t).$$

Exercise 7. Show that for each Turing machine U the set $\{(x,k) : C_U(x) \leq k\}$ is recognizable, (i.e. C_U is upper semicomputable). Conclude that the set $\{(x,k) : C(x) \geq k\}$ is not recognizable, (i.e. C_U is not lower semicomputable).

3 Optimal information measures are not computable

Why is this popular measure of information not computable? It turns out that there is a conflict between computability and optimality. To explain this conflict, we need to define a "measure of information" more generally. Not any function is a useful candidate to be such a measure, for example the constant zero function is obviously useless. In exercise 4 it is proven that most strings are random by showing that there are less than 2^k programs of length less than k. This observation inspires the following definition.

Definition 6. *An* information measure *is a function $F : \{0,1\}^* \to \mathbb{N}$ such that for each k*

$$|\{x : F(x) < k\}| < 2^k.$$

Note that for any machine U, the complexity C_U is an information measure (that is upper semicomputable). The following lemma shows that the reverse is also true:

Lemma 7. *For each upper semicomputable information measure F there exists a machine V such that $C_V = F + 1$.*

Proof. Because F is upper semicomputable, there exists an algorithm that on input k enumerates all x with $F(x) < k$.

The algorithm for machine V on input p. Let n_p be the value of p when interpreted as a number in binary. V enumerates all x for which $F(x) < |p|$ and outputs the n_pth such string that appears. End of the algorithm.

By definition of an information measure, the maximum number of strings x for which $F(x) < n$ is less than 2^n. Hence, to each such x corresponds an n-bit string p on which V outputs x. This implies that $C_V(x) = F(x) + 1$ for all x. □

Definition 8. *An information measure F is* optimal in a set S *of information measures if $F \in S$ and if for every $G \in S$ there exists a c such that $F \leq G + c$.*

Corollary 9 (of Lemma 7)**.** *There exists an information measure that is optimal in the set of all upper semicomputable information measures.*

Proof. For an optimal machine U, the information measure C_U is optimal among all upper semicomputable measures. Indeed, if F is an upper semicomputable measure, then there is a machine V such that $F = C_V + O(1)$ and by optimality of U we have $C_V \geq C_U + O(1)$, hence $F \geq C_U + O(1)$. □

The following theorem implies that there is no information measure that is optimal in the set of computable information measures. More precisely, it states that for any computable measure F, there is a "more clever" computable measure G that "sees" almost maximal structure in strings that seem random to F.

Theorem 10. *For every computable information measure F there exists a computable information measure G such that $G \leq F + 1$ and such that for each length there is a string x for which $F(x) \geq |x|$ and $G(x) \leq \log |x| + 1$.*

Proof. The idea is as follows. The set of all x for which $F(x) \geq |x|$ is decidable and contains strings of all lengths. For each length we can select one such string and define G to have a small value on this string. If for all other strings G takes the same value as F, then the condition in the definition of information measure might be violated. It turns out that if we increase G only a little bit, the condition can be restored.

By definition of information measure, there exists for each $n \geq 1$ at least one n-bit x such that $F(x) \geq n$. Let y_n be the lexicographically first such string. Note that if F is computable, then we can compute y_n from n. The information measure G is given by

$$G(x) = \begin{cases} \log |x| + 1 & \text{if } x = y_{|x|} \\ F(x) + 1 & \text{otherwise.} \end{cases}$$

Clearly G is computable. For each length n, we have $F(y_n) \geq n$ and $G(n) \leq \log n + 1$. It remains to verify that G is an information measure. Let us count the strings for which G is less than k. There are two cases in the definition of G. The number of strings that appear in the second case, is at most the number of strings of F-complexity less than $k - 1$, which is less than 2^{k-1}. For each length there is precisely one string that appears in the first case, and the number of lengths n for which $\log n + 1 < k$ equals 2^{k-1}. Hence, the total number of strings for which G is less than k is less than $2^{k-1} + 2^{k-1} = 2^k$. □

4 The coding theorem

In machine learning, one usually faces a task of modelling a probability density function to some data. The goal is to find a probability function that optimizes two conflicting criteria: on one hand, we want the probability of the data to be as large as possible, and on the other hand, the selected model should be simple. In this section we argue that finding better models results in better compression algorithms for data. We also relate Kolmogorov complexity to a measure of information for probability density functions, called Shannon entropy.

In this section we use computable probability density functions which are functions that take rational values. We define computability for such functions.

Definition 11. *A function $P : S \to \mathbb{Q}$ is* computable *if there exist computable functions $f : S \to \mathbb{Z}$ and $g : S \to \mathbb{N}$ such that $P(s) = f(s)/g(s)$ for all $s \in S$.*

Theorem 12. *If $P : \mathbb{N} \to \mathbb{Q}$ is computable and $\sum_{n \in \{0,1\}^*} P(n) = 1$, then there exists a constant c such that for all n:*

$$C(n) \leq \log \frac{1}{P(n)} + c.$$

The same holds for functions $P : \{0,1\}^* \to \mathbb{Q}$.

Proof. With each $x \in \{0,1\}^*$, we can associate a rational number in $[0,1] \subseteq \mathbb{Q}$ whose binary representation is $0.x$. If $2^{-k} \leq r - s$, then there exists a k-bit x such that $0.x \in [r,s]$.

Let P_c be the cumulative probability density function associated to P, which is given by $P_c(n) = P(1) + \cdots + P(n)$. For some n, let $D(n)$ be the string x of length at most $k = 1 + \log \frac{1}{P(n)}$ such that

$$0.x \in [P_c(n-1), P_c(n)].$$

By construction $|D(n)| \leq 1 + \log \frac{1}{P(n)}$. Note that $D(n) \neq D(m)$ for $n \neq m$ and that D is computable.

Now we construct a machine V that on input $D(n)$ outputs 0^n. On input $p \in \{0,1\}^*$, it evaluates $D(1), D(2), \ldots$ until it finds an n such that $D(n) = p$. If this happens, it outputs 0^n. For this machine we have $C_V(n) = |D(n)|$. Hence,

$$C(n) \leq |D(n)| + O(1) \leq \log \frac{1}{P(n)} + O(1). \qquad \square$$

Remark: We give a more precise version of the theorem above. Let $C_U(P)$ denote the smallest length of a program p such that $U(p,x)$ outputs $P(x)$ for all x. Then the above result can be refined as follows:

$$C(x) \leq C(P) + \log \frac{1}{P(x)} + O(\log C(P)).$$

In other words, the complexity of a string is at most the complexity of a model plus the probability of the string according to this model.

How does this relate to machine learning? We explain this with an example. Imagine there is some dynamical system from which at regular intervals some measurements with precision $\varepsilon > 0$ are made. Let these measurements be $d_1, \ldots, d_N$. x is and encoding of these measurements. On the other hand, some model might predict that these values are $\mu_1, \mu_2, \cdots, \mu_N$ and that the deviations from these values are independent and normally distributed with constant variance. The probability that a measurement generates x is the product over all i of

$$\frac{\epsilon}{\sqrt{2\pi\sigma}} e^{-\frac{(d_i - \mu_i)^2}{2\sigma^2}}.$$

Hence, $\log \frac{1}{P(x)}$ is of the form $a + b\sum (d_i - \mu_i)^2$ and is proportional to the squared error of the predictions. $C(P)$ is a measure of the complexity of the model that computes $\mu_1, \ldots, \mu_N$.

Shannon entropy is a measure of information content in a probability density function that is useful for machine learning.

Definition 13. *For $P : S \to \mathbb{Q}$ let*

$$H(P) = \sum_{\substack{s \in S \\ P(s) > 0}} P(s) \log \frac{1}{P(x)},$$

From Theorem 12 we get a bound for the P-expected value of $C(x)$.

Corollary 14. *For each $P : \{0,1\}^* \to \mathbb{Q}$ and $\sum_x P(x) = 1$ there exists*

$$\sum_x P(x) C(x) \leq H(P) + C(P) + O(\log C(P)).$$

Also the $\geq$-inequality holds within a logarithmic term, but we will not prove this here. This means that if the complexity of P is small, then Shannon entropy is approximately equal to the expected value of Kolmogorov complexity.

5 Symmetry of information

The Kolmogorov complexity of x conditional to y represents the information x that is not already in y. In this section we show that the information in a pair of strings (x, y) is approximately equal to the information in x plus the information in y conditional to x.

Definition 15. *The* conditional *Kolmogorov complexity of x given y is*

$$C_U(x|y) = \min\{|p| : U(p, y) = x\}.$$

There exist machines U such that for all other machines V we have

$$C_U(\cdot|\cdot) \leq C_V(\cdot|\cdot) + O(1).$$

This follows again from the existence of optimal Gödel machines. We use the same convention as before for optimal machines. For example, $C(x|x) \leq O(1)$, and more generally: "For all partial computable f, there exists a c such that $C(f(x)|x) \leq c$ whenever $f(x)$ is defined."

Note that we can interpret Turing machines as devices that compute several output strings: when the machine halts, the output is the sequence of strings on the tape that are separated by single blank symbols. In fact, optimal Gödel machines exist for any number of arguments. In this way, we can define complexity functions like $C(x, y, z|u, v, w)$. All terms in Theorem 16 are now defined.

Exercise 8.

1. $C(xy) = C(x, y) + O(\log |x|)$,
2. Find x and y of length at most n such that $C(x, y) \geq C(xy) + \log n - O(\log \log n)$. Hint: choose x and y that only contain zeros.
3. Show that $C(x|y, k) \leq C(x|y) + O(\log k)$.

Theorem 16. $C(x) + C(y|x) = C(x, y) + O(\log C(x, y))$

This theorem follows from Lemma 17 and 18.

Lemma 17. $C(x, y) \leq C(x) + C(y|x) + O(\log C(x))$

Proof. Let p be a program of length $C(x)$ that prints x and let q be the program that prints y on input x. An obvious idea would be to simply concatenate p and q, to obtain a program for (x, y). But this does might not work for several reasons. For example, after p is executed, the machine needs to scan the beginning of q, but there might be no way to know where q starts.

An obvious solution: lets encode p with a prefix-free code. We already used such code when we constructed prefix-Gödel machines. A string $p = p_1 \dots p_n$ can be encoded as $\hat{p} = p_1 0 p_2 0 \dots p_n 1$. But this does not work either, it would give us a program for (x, y) of length $2C(x) + C(y|x) + O(1)$.

We use the coding above in a more clever way: we code p and q as $\hat{n}pq$, where $n = |p|$. Now it is easy to construct a machine V that on input $\hat{n}pq$ computes (x, y). The first part has length $2 \log |p| = 2 \log C(x)$. The other parts have lengths $C(x)$ and $C(y|x)$. The lemma is proven. □

We now prove the other inequality of Theorem 16.

Lemma 18. $C(x) + C(y|x) \leq C(x,y) + O(\log C(x,y))$

Proof. The lemma follows by showing that: *For any numbers k, ℓ such that $k + \ell = C(x,y)$ we have either*

$$\begin{aligned} & C(x) \leq k + O(\log(k+\ell)) \\ \textit{or} \quad & C(y|x) \leq \ell + O(\log(k+\ell)). \end{aligned}$$

To see that this implies the lemma, apply the claim with k a bit smaller than $C(x)$ to conclude that $C(y|x) \leq \ell + O(\log(k+\ell))$. Now observe that $\ell = (k+\ell) - k = C(x,y) - C(x) + O(\log k + \ell)$. This implies the lemma.

It remains to prove the claim. Generate a list of all pairs (u,v) computed by programs of length exactly $k + \ell$. Note that this list:

- contains (x,y),
- can be enumerated given k and ℓ,
- has at most $2^{k+\ell}$ elements.

We consider two cases:

1. Suppose x appears less than 2^ℓ times as a first element in this list. From x, k, ℓ, we can enumerate all pairs whose first element is x. We can specify y by the index of the element (x,y) in this enumeration. This implies that $C(y|x,k,\ell) \leq \ell + O(1)$. By a similar reasoning as in exercise 8.3 this implies the second possibility of the claim.
2. Otherwise, x appears at least 2^ℓ times as a first element. The number of strings u that appear at least 2^ℓ times is bounded by $2^{\ell+k}/2^\ell = 2^k$. These strings u can be enumerated, and the index of x in this enumeration specifies x. Hence, this implies $C(x|k,\ell) \leq k + O(1)$ and this implies the first possibility of the claim. □