Kolmogorov Complexity

Pei Wang

School of Mathematical Sciences
Peking University

pei.wng@gmail.com

May 15, 2012

Outline

A Turing Machine

- a finite-state machine operating on a finite symbol set,
- at each unit of time, the machine inspects the program tape, writes some symbols on a work tape, changes its state according to its transition table, and calls for more program.
- reads from left to right only.

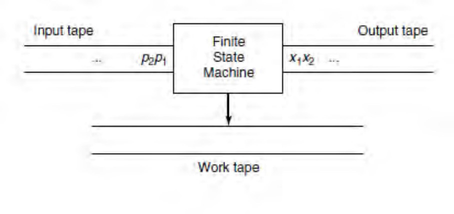

Figure: A Turing Machine

A Turing Machine

- a finite-state machine operating on a finite symbol set,
- at each unit of time, the machine inspects the program tape, writes some symbols on a work tape, changes its state according to its transition table, and calls for more program.
- reads from left to right only.

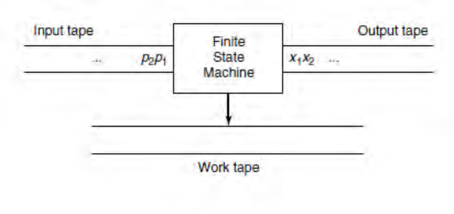

Figure: A Turing Machine

A Turing Machine

- a finite-state machine operating on a finite symbol set,
- at each unit of time, the machine inspects the program tape, writes some symbols on a work tape, changes its state according to its transition table, and calls for more program.
- reads from left to right only.

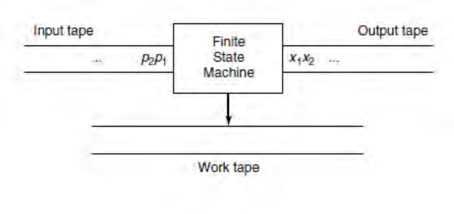

Figure: A Turing Machine

A Turing Machine

- The programs this Turing Machine reads form a prefix-free set:
  - no program leading to a halting computation can be the prefix of another such program.
- We can view the Turing machine as a map:
  - from a set of finite-length binary strings to the set of finite- or infinite-length binary strings,
- The computation may halt or not.

Definition

The set of functions $f : \{0,1\}^* \to \{0,1\}^* \cup \{0,1\}^\infty$ computable by Turing machines is called the set of `partial recursive functions`.

A Turing Machine

- The programs this Turing Machine reads form a prefix-free set:
  - no program leading to a halting computation can be the prefix of another such program.
- We can view the Turing machine as a map:
  - from a set of finite-length binary strings to the set of finite- or infinite-length binary strings,
- The computation may halt or not.

Definition

The set of functions $f : \{0,1\}^* \to \{0,1\}^* \cup \{0,1\}^\infty$ computable by Turing machines is called the set of `partial recursive functions`.

A Turing Machine

- The programs this Turing Machine reads form a prefix-free set:
  - no program leading to a halting computation can be the prefix of another such program.
- We can view the Turing machine as a map:
  - from a set of finite-length binary strings to the set of finite- or infinite-length binary strings,
- The computation may halt or not.

Definition

The set of functions $f : \{0,1\}^* \to \{0,1\}^* \cup \{0,1\}^\infty$ computable by Turing machines is called the set of `partial recursive functions`.

A Turing Machine

- The programs this Turing Machine reads form a prefix-free set:
  - no program leading to a halting computation can be the prefix of another such program.
- We can view the Turing machine as a map:
  - from a set of finite-length binary strings to the set of finite- or infinite-length binary strings,
- The computation may halt or not.

Definition
The set of functions $f : \{0,1\}^* \to \{0,1\}^* \cup \{0,1\}^\infty$ computable by Turing machines is called the set of `partial recursive functions`.

A Turing Machine

- The programs this Turing Machine reads form a prefix-free set:
  - no program leading to a halting computation can be the prefix of another such program.
- We can view the Turing machine as a map:
  - from a set of finite-length binary strings to the set of finite- or infinite-length binary strings,
- The computation may halt or not.

Definition

The set of functions $f : \{0,1\}^* \to \{0,1\}^* \cup \{0,1\}^\infty$ computable by Turing machines is called the set of `partial recursive functions`.

A Turing Machine

- The programs this Turing Machine reads form a prefix-free set:
  - no program leading to a halting computation can be the prefix of another such program.
- We can view the Turing machine as a map:
  - from a set of finite-length binary strings to the set of finite- or infinite-length binary strings,
- The computation may halt or not.

Definition

The set of functions $f : \{0,1\}^* \to \{0,1\}^* \cup \{0,1\}^\infty$ computable by Turing machines is called the set of `partial recursive functions`.

Turing Machine

- Turing argued that this machine could mimic the computational ability of a human being.
- After Turing's work, it turned out that every new computational system could be reduced to a Turing machine, and conversely.
- In particular,the familiar digital computer with its CPU, memory, and input output devices could be simulated by and could simulate a Turing machine.

Turing Machine

- Turing argued that this machine could mimic the computational ability of a human being.
- After Turing's work, it turned out that every new computational system could be reduced to a Turing machine, and conversely.
- In particular,the familiar digital computer with its CPU, memory, and input output devices could be simulated by and could simulate a Turing machine.

Turing Machine

- Turing argued that this machine could mimic the computational ability of a human being.
- After Turing's work, it turned out that every new computational system could be reduced to a Turing machine, and conversely.
- In particular,the familiar digital computer with its CPU, memory, and input output devices could be simulated by and could simulate a Turing machine.

Turing's Thesis

- The class of algorithmically computable numerical functions (in the intuitive sense) coincides with the class of partial recursive functions [Li and Vitányi, 2008].
- All (sufficiently complex) computational models are equivalent in the sense that they can compute the same family of functions.

Turing's Thesis

- The class of algorithmically computable numerical functions (in the intuitive sense) coincides with the class of partial recursive functions [Li and Vitányi, 2008].
- All (sufficiently complex) computational models are equivalent in the sense that they can compute the same family of functions.

The Universal Turing Machine

- Universal:
  - All but the most trivial computers are universal,in the sense that they can mimic the actions of other computers.
- The universal Turing machine:
  - the conceptually simplest universal computer.

The Universal Turing Machine

- Universal:
  - All but the most trivial computers are universal,in the sense that they can mimic the actions of other computers.
- The universal Turing machine:
  - the conceptually simplest universal computer.

The Universal Turing Machine

- Universal:
  - All but the most trivial computers are universal,in the sense that they can mimic the actions of other computers.
- The universal Turing machine:
  - the conceptually simplest universal computer.

The Universal Turing Machine

- Universal:
  - All but the most trivial computers are universal,in the sense that they can mimic the actions of other computers.
- The universal Turing machine:
  - the conceptually simplest universal computer.

Outline

Kolmogorov Complexity of a String

Definition

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

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

- x is a finite-length binary string,
- $\mathcal{U}$ is a universal computer,
- $\ell(x)$ is the length of the string x,
- $\mathcal{U}(p)$ is the output of the computer $\mathcal{U}$ when presented with a program p.

Kolmogorov Complexity of a String

Definition

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

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

- x is a finite-length binary string,
- $\mathcal{U}$ is a universal computer,
- $\ell(x)$ is the length of the string x,
- $\mathcal{U}(p)$ is the output of the computer $\mathcal{U}$ when presented with a program p.

Kolmogorov Complexity of a String

Definition

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

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

- x is a finite-length binary string,
- $\mathcal{U}$ is a universal computer,
- $\ell(x)$ is the length of the string x,
- $\mathcal{U}(p)$ is the output of the computer $\mathcal{U}$ when presented with a program p.

Kolmogorov Complexity of a String

Definition

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

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

- x is a finite-length binary string,
- $\mathcal{U}$ is a universal computer,
- $\ell(x)$ is the length of the string x,
- $\mathcal{U}(p)$ is the output of the computer $\mathcal{U}$ when presented with a program p.

Kolmogorov Complexity of a String

Definition

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

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

- x is a finite-length binary string,
- $\mathcal{U}$ is a universal computer,
- $\ell(x)$ is the length of the string x,
- $\mathcal{U}(p)$ is the output of the computer $\mathcal{U}$ when presented with a program p.

Kolmogorov Complexity of a String

Remark

- *A useful technique for thinking about Kolmogorov complexity is the following–if one person can describe a sequence to another person in such a manner as to lead unambiguously to a computation of that sequence in a finite amount of time.*
- *The number of bits in that communication is an upper bound on the Kolmogorov complexity.*

Kolmogorov Complexity of a String

Remark

- *A useful technique for thinking about Kolmogorov complexity is the following–if one person can describe a sequence to another person in such a manner as to lead unambiguously to a computation of that sequence in a finite amount of time.*
- *The number of bits in that communication is an upper bound on the Kolmogorov complexity.*

Kolmogorov Complexity of a String

Remark

- *A useful technique for thinking about Kolmogorov complexity is the following–if one person can describe a sequence to another person in such a manner as to lead unambiguously to a computation of that sequence in a finite amount of time.*
- *The number of bits in that communication is an upper bound on the Kolmogorov complexity.*

Kolmogorov Complexity of a String

Example

One can say"Print out the first 1,239,875,981,825,931 bits of the square root of e." We see that the Kolmogorov complexity of this huge number is no greater than $(8)(73) = 584$ *bits.*

The fact that there is a simple algorithm to calculate the square root of e provides the saving in descriptive complexity.

Conditional Kolmogorov Complexity

Definition

If we assume that the computer already knows the length of x, we can define `the conditional Kolmogorov complexity knowing` $\ell(x)$ *as:*

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

Universality of Kolmogorov Complexity

Theorem

If $\mathcal{U}$ is a universal computer, for any other computer $\mathcal{A}$ there exists a constant $c_{\mathcal{A}}$ such that

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

Universality of Kolmogorov Complexity

Proof. Assume $p_{\mathcal{A}}$ is a program for computer $\mathcal{A}$ to print x. Thus, $\mathcal{A}(p_{\mathcal{A}}) = x$. $s_{\mathcal{A}}$ is a simulation program which tells computer $\mathcal{U}$ how to simulate computer $\mathcal{A}$. The program for $\mathcal{U}$ to print x is $p = s_{\mathcal{A}} p_{\mathcal{A}}$ and its length is

$$\ell(p) = \ell(s_{\mathcal{A}}) + \ell(p_{\mathcal{A}}) = c_{\mathcal{A}} + \ell(p_{\mathcal{A}}),$$

where $c_{\mathcal{A}}$ is the length of the simulation program. Hence,

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} \ell(p) \leq \min_{p:\mathcal{A}(p)=x} (\ell(p) + c_{\mathcal{A}}) = K_{\mathcal{A}}(x) + c_{\mathcal{A}}$$

for all strings x. $\square$

Universality of Kolmogorov Complexity

Remark

- *The constant $c_{\mathcal{A}}$ in the theorem may be very large.*
- *The crucial point is that the length of this simulation program is independent of the length of x.*
- *For sufficiently long x, the length of this simulation program can be neglected.*
- *We discuss Kolmogorov complexity without talking about the constants.*

Universality of Kolmogorov Complexity

Remark

- *The constant $c_{\mathcal{A}}$ in the theorem may be very large.*
- *The crucial point is that the length of this simulation program is independent of the length of x.*
- *For sufficiently long x, the length of this simulation program can be neglected.*
- *We discuss Kolmogorov complexity without talking about the constants.*

Universality of Kolmogorov Complexity

Remark

- *The constant $c_{\mathcal{A}}$ in the theorem may be very large.*
- *The crucial point is that the length of this simulation program is independent of the length of x.*
- *For sufficiently long x, the length of this simulation program can be neglected.*
- *We discuss Kolmogorov complexity without talking about the constants.*

Universality of Kolmogorov Complexity

Remark

- *The constant $c_{\mathcal{A}}$ in the theorem may be very large.*
- *The crucial point is that the length of this simulation program is independent of the length of x.*
- *For sufficiently long x, the length of this simulation program can be neglected.*
- *We discuss Kolmogorov complexity without talking about the constants.*

Universality of Kolmogorov Complexity

Remark

- *The constant $c_{\mathcal{A}}$ in the theorem may be very large.*
- *The crucial point is that the length of this simulation program is independent of the length of x.*
- *For sufficiently long x, the length of this simulation program can be neglected.*
- *We discuss Kolmogorov complexity without talking about the constants.*

Universality of Kolmogorov Complexity

Remark

If $\mathcal{A}$ and $\mathcal{U}$ are both universal, we have $|K_{\mathcal{U}}(x) - K_{\mathcal{A}}(x)| < c$ for all x. Hence, we will drop all mention of $\mathcal{U}$ in all further definitions. We will assume that the unspecified computer $\mathcal{U}$ is a fixed universal computer.

Upper Bound on Conditional Complexity

Theorem
(Conditional complexity is less than the length of the sequence)

$$K_{\mathcal{U}}(x|\ell(x)) \leq \ell(x) + c.$$

Proof. A program for printing x is

Print the following ℓ-bit sequence: $x_1 x_2 \ldots x_{\ell(x)}$.

Note that no bits are required to describe ℓ since ℓ is given. The program is self-delimiting because $\ell(x)$ is provided and the end of the program is thus clearly defined. The length of this program is $\ell(x) + c$. □

Upper Bound on Conditional Complexity

Remark

Without knowledge of the length of the string, we will need an additional stop symbol or we can use a self-punctuating scheme like the one described in the proof of the next theorem.

Upper Bound on Kolmogorov Complexity

Theorem

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{U}}(x|\ell(x)) + 2\log \ell(x) + c.$$

Proof.

- If the computer does not know $\ell(x)$,we must have some way of informing the computer when it has come to the end of the string of bits that describes the sequence.
- We describe a simple but inefficient method that uses a sequence 01 as a "comma."

Upper Bound on Kolmogorov Complexity

Theorem

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{U}}(x|\ell(x)) + 2\log\ell(x) + c.$$

Proof.

- If the computer does not know $\ell(x)$,we must have some way of informing the computer when it has come to the end of the string of bits that describes the sequence.
- We describe a simple but inefficient method that uses a sequence 01 as a "comma."

Upper Bound on Kolmogorov Complexity

Theorem

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{U}}(x|\ell(x)) + 2 \log \ell(x) + c.$$

Proof.

- If the computer does not know $\ell(x)$,we must have some way of informing the computer when it has come to the end of the string of bits that describes the sequence.
- We describe a simple but inefficient method that uses a sequence 01 as a "comma."

Proof Cont.

- Suppose that $\ell(x) = n$. To describe $\ell(x)$, repeat every bit of the binary expansion of n twice; then end the description with a 01 so that the computer knows that it has come to the end of the description of n.
- For example, the number 5 (binary 101) will be described as 11001101. This description requires $2\lceil \log n \rceil + 2$ bits.
- Thus, inclusion of the binary representation of $\ell(x)$ does not add more than $2\lceil \log n \rceil + 2$ bits to the length of the program, and we have the bound in the theorem.

$\square$

Proof Cont.

- Suppose that $\ell(x) = n$. To describe $\ell(x)$, repeat every bit of the binary expansion of n twice; then end the description with a 01 so that the computer knows that it has come to the end of the description of n.
- For example, the number 5 (binary 101) will be described as 11001101. This description requires $2\lceil \log n \rceil + 2$ bits.
- Thus, inclusion of the binary representation of $\ell(x)$ does not add more than $2\lceil \log n \rceil + 2$ bits to the length of the program, and we have the bound in the theorem.

$\square$

Proof Cont.

- Suppose that $\ell(x) = n$. To describe $\ell(x)$, repeat every bit of the binary expansion of n twice; then end the description with a 01 so that the computer knows that it has come to the end of the description of n.
- For example, the number 5 (binary 101) will be described as 11001101. This description requires $2\lceil \log n \rceil + 2$ bits.
- Thus, inclusion of the binary representation of $\ell(x)$ does not add more than $2\lceil \log n \rceil + 2$ bits to the length of the program, and we have the bound in the theorem.

$\square$

Lower Bound on Kolmogorov Complexity

Remark

A more efficient method for describing n is to do so recursively. The theorem above can be improved to

$$K_{\mathcal{U}}(x) \le K_{\mathcal{U}}(x|\ell(x)) + \log^* \ell(x) + c.$$

Lower Bound on Kolmogorov Complexity

Theorem

The number of strings x with complexity $K(x) < k$ satisfies

$$|\{x \in \{0,1\}^* : K(x) < k\}| < 2^k. \quad (1)$$

Lower Bound on Kolmogorov Complexity

Proof. We list all the programs of length $< k$, we have

$$\underbrace{\Lambda,}_{1}\ \underbrace{0, 1,}_{2}\ \underbrace{00, 01, 10, 11,}_{4} \ldots, \underbrace{\ldots, \overbrace{11\ldots1}^{k-1}}_{2^{k-1}}$$

and the total number of such programs is

$$1 + 2 + 4 + \ldots + 2^{k-1} = 2^k - 1 \leq 2^k.$$

Since each program can produce only one possible output sequence, the number of sequences with complexity $< k$ is less than 2^k. $\square$

Outline

- The Kolmogorov complexity will depend on the computer, but only up to an additive constant.
- We consider a computer that can accept unambiguous commands in English (with numbers given in binary notation).

- The Kolmogorov complexity will depend on the computer, but only up to an additive constant.
- We consider a computer that can accept unambiguous commands in English (with numbers given in binary notation).

Sequences of Zeros

Example

(A sequence of n zeros) If we assume that the computer knows n, a short program to print this string is

Print the specified number of zeros.

The length of this program is a constant number of bits. This program length does not depend on n. Hence, the Kolmogorov complexity of this sequence is c, and

$$K(000\ldots0|n) = c.$$

Integer

Example

(Integer n) If the computer knows the number of bits in the binary representation of the integer, we need only provide the values of these bits. This program will have length $(c + \log n)$*.*

In general, the computer will not know the length of the binary representation of the integer. By informing the computer in a recursive way when the description ends, we see that the Kolmogorov complexity of an integer is bounded by

$$K(n) \leq \log^* n + c.$$

Pei Wang

A Notation

We introduce a notation for the binary entropy function

$$H_0(p) = -(1-p)\log(1-p) - p\log p.$$

Thus, when we write $H_0(\frac{1}{n}\sum_{i=1}^{n} x_i)$, we will mean $-\bar{X}_n \log \bar{X}_n - (1-\bar{X}_n)\log(1-\bar{X}_n)$, and not the entropy of random variable $\bar{X}_n$. When there is no confusion, we shall simply write $H(p)$ for $H_0(p)$.

An Inequality

Lemma

For $k \neq 0, n$, we have

$$\binom{n}{k} \leq \sqrt{\frac{n}{\pi k(n-k)}} 2^{nH(k/n)}. \qquad (2)$$

Proof. Applying a strong form of Stirling's approximation [Feller, 2008], which states that

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}} \qquad (3)$$

We obtain

$$\begin{aligned}\binom{n}{k} &\leq \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}}{\sqrt{2\pi k}\left(\frac{k}{e}\right)^k \sqrt{2\pi(n-k)}\left(\frac{n-k}{e}\right)^{(n-k)}} \\ &\leq \sqrt{\frac{n}{\pi k(n-k)}}\left(\frac{k}{n}\right)^{-k}\left(\frac{n-k}{n}\right)^{-(n-k)} e^{\frac{1}{12n}} \\ &\leq \sqrt{\frac{n}{\pi k(n-k)}} 2^{nH(k/n)},\end{aligned}$$

since $e^{\frac{1}{12n}} < e^{\frac{1}{12}} = 1.087 < \sqrt{2}$.

Sequence Compress

Example

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

Consider the following program:

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

This program will print out the required sequence. The only variables in the program are k (with known range $\{0, 1, \ldots, n\}$)and i (with conditional range $\{1, 2, \ldots, \binom{n}{k}\}$).

Sequence Compress

Example

The total length of this program is

$$\ell(p) = c + \underbrace{\log n}_{\text{to express } k} + \underbrace{\log \binom{n}{k}}_{\text{to express } i} \tag{4}$$

$$\leq c' + \log n + nH\left(\frac{k}{n}\right) - \frac{1}{2}\log n, \tag{5}$$

Since $\binom{n}{k} \leq \sqrt{\frac{n}{\pi k(n-k)}}\, 2^{nH(k/n)}$ *by (2) for* $p = \frac{k}{n}$ *and* $q = 1 - p$ *and* $k \neq 0$ *and* $k \neq n$. *We have used* $\log n$ *bits to represent* k. *Thus, if* $\sum_{i=1}^{n} x_i = k$*, then*

$$K(x_1, x_2, \ldots, x_n | n) \leq nH\left(\frac{k}{n}\right) + \frac{1}{2}\log n + c. \tag{6}$$

Sequence Compress

Theorem

The Kolmogorov complexity of a binary string x is bounded by

$$K(x_1, x_2, \ldots, x_n | n) \leq nH\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) + \frac{1}{2}\log n + c. \quad (7)$$

Sequence Compress

Remark

- *In general, when the length $\ell(x)$ of the sequence x is small, the constants that appear in the expressions for the Kolmogorov complexity will overwhelm the contributions due to $\ell(x)$.*
- *Hence, the theory is useful primarily when $\ell(x)$ is very large. In such cases we can safely neglect the terms that do not depend on $\ell(x)$.*

Sequence Compress

Remark

- *In general, when the length $\ell(x)$ of the sequence x is small, the constants that appear in the expressions for the Kolmogorov complexity will overwhelm the contributions due to $\ell(x)$.*
- *Hence, the theory is useful primarily when $\ell(x)$ is very large. In such cases we can safely neglect the terms that do not depend on $\ell(x)$.*

Sequence Compress

Remark

- *In general, when the length $\ell(x)$ of the sequence x is small, the constants that appear in the expressions for the Kolmogorov complexity will overwhelm the contributions due to $\ell(x)$.*
- *Hence, the theory is useful primarily when $\ell(x)$ is very large. In such cases we can safely neglect the terms that do not depend on $\ell(x)$.*

Outline

Definition
Models of Computation
Definitions
Examples
Kolmogorov Complexity and Entropy
Prefix Code
Kolmogorov Complexity and Entropy
Algorithmically Random and Incompressible
Random and Incompressible
Asymptotic Equipartition Property
A Random Bernoulli Sequence
Ω
Kolmogorov Complexity and Universal Probability
Summary
References

Prefix Code

Definition

A code is called a `prefix code` or an `instantaneous code` if no codeword is a prefix of any other codeword

Theorem

(Kraft inequality) For any instantaneous code (prefix code) over an alphabet of size D, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy the inequality

$$\sum_i D^{-\ell_i} \leq 1. \tag{8}$$

Conversely, given a set of codeword lengths that satisfy this inequality,there exists an instantaneous code with these word lengths.

Prefix Code

Definition

A code is called a `prefix code` *or an* `instantaneous code` *if no codeword is a prefix of any other codeword*

Theorem

(Kraft inequality) For any instantaneous code (prefix code) over an alphabet of size D, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ *must satisfy the inequality*

$$\sum_i D^{-\ell_i} \leq 1. \tag{8}$$

Conversely, given a set of codeword lengths that satisfy this inequality,there exists an instantaneous code with these word lengths.

Kraft Inequality

Proof. Consider a D-ary tree.

- Let the branches of the tree represent the symbols of the codeword.
- Then each codeword is represented by a leaf on the tree. The path from the root traces out the symbols of the codeword.
- A binary example of such a tree is shown in Figure (2).
- The prefix condition on the codewords implies that each codeword eliminates its descendants as possible codewords.

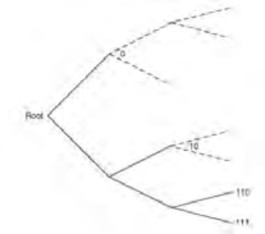

Figure: Code tree for the Kraft inequality

Kraft Inequality

Proof. Consider a D-ary tree.

- Let the branches of the tree represent the symbols of the codeword.
- Then each codeword is represented by a leaf on the tree. The path from the root traces out the symbols of the codeword.
- A binary example of such a tree is shown in Figure (2).
- The prefix condition on the codewords implies that each codeword eliminates its descendants as possible codewords.

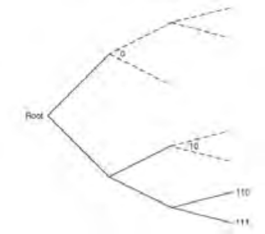

Figure: Code tree for the Kraft inequality

Kraft Inequality

Proof. Consider a D-ary tree.

- Let the branches of the tree represent the symbols of the codeword.
- Then each codeword is represented by a leaf on the tree. The path from the root traces out the symbols of the codeword.
- A binary example of such a tree is shown in Figure (2).
- The prefix condition on the codewords implies that each codeword eliminates its descendants as possible codewords.

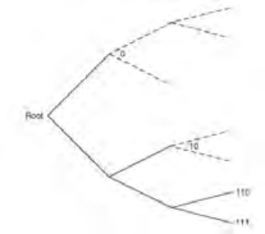

Figure: Code tree for the Kraft inequality

Kraft Inequality

Proof. Consider a D-ary tree.

- Let the branches of the tree represent the symbols of the codeword.
- Then each codeword is represented by a leaf on the tree. The path from the root traces out the symbols of the codeword.
- A binary example of such a tree is shown in Figure (2).
- The prefix condition on the codewords implies that each codeword eliminates its descendants as possible codewords.

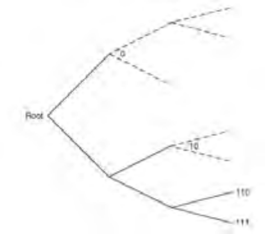

Figure: Code tree for the Kraft inequality

Proof Cont.

- Consider all nodes of the tree at level ℓ_{max}. Some of them are codewords, some are descendants of codewords, and some are neither.
- A codeword at level ℓ_i has $D^{\ell_{max}-\ell_i}$ descendants at level ℓ_{max}. Each of these descendant sets must be disjoint.
- Also, the total number of nodes in these sets must be less than or equal to $D^{\ell_{max}}$.

Hence,

$$\sum D^{\ell_i - l_{max}} \leq D^{\ell_{max}} \tag{9}$$

Thus,

$$\sum D^{\ell_i} \leq 1. \tag{10}$$

Proof Cont.

- Consider all nodes of the tree at level ℓ_{max}. Some of them are codewords, some are descendants of codewords, and some are neither.
- A codeword at level ℓ_i has $D^{\ell_{max}-\ell_i}$ descendants at level ℓ_{max}. Each of these descendant sets must be disjoint.
- Also, the total number of nodes in these sets must be less than or equal to $D^{\ell_{max}}$.

Hence,

$$\sum D^{\ell_i - l_{max}} \leq D^{\ell_{max}} \tag{9}$$

Thus,

$$\sum D^{\ell_i} \leq 1. \tag{10}$$

Proof Cont.

- Consider all nodes of the tree at level ℓ_{max}. Some of them are codewords, some are descendants of codewords, and some are neither.
- A codeword at level ℓ_i has $D^{\ell_{max}-\ell_i}$ descendants at level ℓ_{max}. Each of these descendant sets must be disjoint.
- Also, the total number of nodes in these sets must be less than or equal to $D^{\ell_{max}}$.

Hence,

$$\sum D^{\ell_i - l_{max}} \leq D^{\ell_{max}} \tag{9}$$

Thus,

$$\sum D^{\ell_i} \leq 1. \tag{10}$$

Proof Cont.

- Conversely, given any set of codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ that satisfy the Kraft inequality,we can always construct a tree like the one in Figure (2)
- Label the first node (lexicographically) of depth ℓ_1 as codeword 1, and remove its descendants from the tree.
- Then label the first remaining node of depth ℓ_2 as codeword 2, and so on.
- Proceeding this way, we construct a prefix code with the specified $\ell_1, \ell_2, \ldots, \ell_m$. $\square$

Proof Cont.

- Conversely, given any set of codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ that satisfy the Kraft inequality,we can always construct a tree like the one in Figure (2)
- Label the first node (lexicographically) of depth ℓ_1 as codeword 1, and remove its descendants from the tree.
- Then label the first remaining node of depth ℓ_2 as codeword 2, and so on.
- Proceeding this way, we construct a prefix code with the specified $\ell_1, \ell_2, \ldots, \ell_m$. □

Proof Cont.

- Conversely, given any set of codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ that satisfy the Kraft inequality,we can always construct a tree like the one in Figure (2)
- Label the first node (lexicographically) of depth ℓ_1 as codeword 1, and remove its descendants from the tree.
- Then label the first remaining node of depth ℓ_2 as codeword 2, and so on.
- Proceeding this way, we construct a prefix code with the specified $\ell_1, \ell_2, \ldots, \ell_m$. □

Proof Cont.

- Conversely, given any set of codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ that satisfy the Kraft inequality,we can always construct a tree like the one in Figure (2)
- Label the first node (lexicographically) of depth ℓ_1 as codeword 1, and remove its descendants from the tree.
- Then label the first remaining node of depth ℓ_2 as codeword 2, and so on.
- Proceeding this way, we construct a prefix code with the specified $\ell_1, \ell_2, \ldots, \ell_m$. $\square$

Extended Kraft Inequality

Theorem

For any countably infinite set of codewords that form a prefix code, the codeword lengths satisfy the extended Kraft inequality,

$$\sum_{i}^{\infty} D^{-\ell_i} \leq 1. \qquad (11)$$

Conversely, given any $\ell_1, \ell_2, \ldots$ satisfying the extended Kraft inequality, we can construct a prefix code with these codeword lengths.

Extended Kraft Inequality

Proof. Let the D-ary alphabet be $0, 1, \ldots, D-1$. Consider the ith codeword $y_1 y_2 \ldots y_{\ell_i}$. Let $0.y_1 y_2 \ldots y_{\ell_i}$ be the real number given by the D-ary expansion

$$0.y_1 y_2 \ldots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j}. \tag{12}$$

This codeword corresponds to the interval

$$\left[0.y_1 y_2 \ldots y_{\ell_i}, 0.y_1 y_2 \ldots y_{\ell_i} + \frac{1}{D^{-\ell_i}}\right)$$

the set of all real numbers whose D-ary expansion begins with $0.y_1 y_2 \ldots y_{\ell_i}$. This is a subinterval of the unit interval [0, 1]. By the prefix condition, these intervals are disjoint. Hence, the sum of their lengths has to be less than or equal to 1. This proves that

$$\sum_{i}^{\infty} D^{-\ell_i} \leq 1. \tag{13}$$

Proof Cont.

- Just as in the finite case, we can reverse the proof to construct the code for a given $\ell_1, \ell_2, \ldots$ that satisfies the Kraft inequality.
- First, reorder the indexing so that $\ell_1 \leq \ell_2 \leq \ldots$ Then simply assign the intervals in order from the low end of the unit interval.
- For example, if we wish to construct a binary code with $\ell_1 = 1, \ell_2 = 2, \ldots$, we assign the intervals $[0, \frac{1}{2}), [\frac{1}{2}, \frac{1}{4}), \ldots$ to the symbols, with corresponding codewords $0, 10, \ldots$

$\square$

Proof Cont.

- Just as in the finite case, we can reverse the proof to construct the code for a given $\ell_1, \ell_2, \ldots$ that satisfies the Kraft inequality.
- First, reorder the indexing so that $\ell_1 \leq \ell_2 \leq \ldots$ Then simply assign the intervals in order from the low end of the unit interval.
- For example, if we wish to construct a binary code with $\ell_1 = 1, \ell_2 = 2, \ldots$, we assign the intervals $[0, \frac{1}{2}), [\frac{1}{2}, \frac{1}{4}), \ldots$ to the symbols, with corresponding codewords $0, 10, \ldots$

$\square$

Proof Cont.

- Just as in the finite case, we can reverse the proof to construct the code for a given $\ell_1, \ell_2, \ldots$ that satisfies the Kraft inequality.
- First, reorder the indexing so that $\ell_1 \leq \ell_2 \leq \ldots$ Then simply assign the intervals in order from the low end of the unit interval.
- For example, if we wish to construct a binary code with $\ell_1 = 1, \ell_2 = 2, \ldots$, we assign the intervals $[0, \frac{1}{2}), [\frac{1}{2}, \frac{1}{4}), \ldots$ to the symbols, with corresponding codewords $0, 10, \ldots$

$\square$

Expected Code Length

Theorem

The expected length L of any instantaneous D-ary code for a random variable X is greater than or equal to the entropy $H_D(X)$; that is,

$$L \geq H_D(X). \tag{14}$$

Expected Code Length

Proof. We can write the difference between the expected length and the entropy as

$$L - H_D(X) = \sum p_i \ell_i - \sum p_i \log_D \frac{1}{p_i} \tag{15}$$

$$= -\sum p_i \log_D D^{-\ell_i} + \sum p_i \log_D p_i. \tag{16}$$

Letting $r_i = D^{-\ell_i} / \sum_j D^{-\ell_j}$ and $c = \sum_j D^{-\ell_j}$,we obtain

$$L - H = -\sum p_i \log_D \frac{p_i}{r_i} - \log_D c \tag{17}$$

$$= D(\mathbf{p}||\mathbf{r}) + \log_D \frac{1}{c} \tag{18}$$

$$\geq 0 \tag{19}$$

by the nonnegativity of relative entropy and the fact (Kraft inequality) that $c \leq 1$. Hence, $L \leq H$ with equality iff $p_i = D^{-\ell_i}$ (i.e., iff $-\log_D p_i$ is an integer for all i). □

Outline

Kraft Inequality

Lemma

For any computer $\mathcal{U}$,

$$\sum_{p:\mathcal{U}(p)\,halts} 2^{-\ell(p)} \leq 1. \quad (20)$$

Proof. If the computer halts on any program, it does not look any further for input. Hence, there cannot be any other halting program with this program as a prefix. Thus, the halting programs form a prefix-free set, and their lengths satisfy the Kraft inequality. $\square$

Kolmogorov Complexity and Entropy

Theorem

Let the stochastic process $\{X_i\}$ be drawn i.i.d. according to the probability mass function $f(x), x \in \mathcal{X}$, where $\mathcal{X}$ is a finite alphabet. Let $f(x^n) = \prod_{i=1}^{n} f(x_i)$. Then there exists a constant c such that

$$H(X) \leq \frac{1}{n} \sum_{x^n} f(x^n) K(x^n|n) \leq H(X) + \frac{(|\mathcal{X}| - 1) \log n}{n} + \frac{c}{n} \tag{21}$$

for all n. Thus

$$E \frac{1}{n} K(x^n|n) \to H(X). \tag{22}$$

Kolmogorov Complexity and Entropy

Proof.

- Consider the lower bound. The allowed programs satisfy the prefix property, and thus their lengths satisfy the Kraft inequality.
- We assign to each x^n the length of the shortest program p such that $\mathcal{U}(p, n) = x^n$.These shortest programs also satisfy the Kraft inequality.
- We know from the theory of source coding (9) that the expected codeword length must be greater than the entropy.

Hence,

$$\sum_{x^n} f(x^n)K(x^n|n) \geq H(X_1, X_2, \ldots, X_n) = nH(x). \qquad (23)$$

We first prove the upper bound when X is binary (i.e.,$X_1, X_2, \ldots, X_n$ are i.i.d. Bernoulli(θ)).

Kolmogorov Complexity and Entropy

Proof.

- Consider the lower bound. The allowed programs satisfy the prefix property, and thus their lengths satisfy the Kraft inequality.
- We assign to each x^n the length of the shortest program p such that $\mathcal{U}(p, n) = x^n$.These shortest programs also satisfy the Kraft inequality.
- We know from the theory of source coding (9) that the expected codeword length must be greater than the entropy.

Hence,

$$\sum_{x^n} f(x^n) K(x^n|n) \geq H(X_1, X_2, \ldots, X_n) = nH(x). \tag{23}$$

We first prove the upper bound when X is binary (i.e.,$X_1, X_2, \ldots, X_n$ are i.i.d. Bernoulli(θ)).

Kolmogorov Complexity and Entropy

Proof.

- Consider the lower bound. The allowed programs satisfy the prefix property, and thus their lengths satisfy the Kraft inequality.
- We assign to each x^n the length of the shortest program p such that $\mathcal{U}(p, n) = x^n$.These shortest programs also satisfy the Kraft inequality.
- We know from the theory of source coding (9) that the expected codeword length must be greater than the entropy.

Hence,

$$\sum_{x^n} f(x^n)K(x^n|n) \geq H(X_1, X_2, \ldots, X_n) = nH(x). \qquad (23)$$

We first prove the upper bound when X is binary (i.e.,$X_1, X_2, \ldots, X_n$ are i.i.d. Bernoulli(θ)).

Proof Cont.

Using result (7), we can bound the complexity of a binary string by

$$K(x_1, x_2, \ldots, x_n|n) \leq nH_0\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) + \frac{1}{2}\log n + c. \tag{24}$$

$$EK(X_1, X_2, \ldots, X_n|n) \leq nEH_0\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) + \frac{1}{2}\log n + c \tag{25}$$

$$\stackrel{(a)}{\leq} nH_0\left(\frac{1}{n}\sum_{i=1}^{n} EX_i\right) + \frac{1}{2}\log n + c \tag{26}$$

$$= nH_0(\theta) + \frac{1}{2}\log n + c, \tag{27}$$

where (a) follows from Jensen's inequality and the concavity of the entropy. Thus, we have proved the upper bound in the theorem for binary processes.

Proof Cont.

- We can use the same technique for the case of a nonbinary finite alphabet.
- We first describe the type of the sequence (the empirical frequency of occurrence of each of the alphabet symbols as defined later) using $(|\mathcal{X}| - 1) \log n$ bits (the frequency of the last symbol can be calculated from the frequencies of the rest).
- Then we describe the index of the sequence within the set of all sequences having the same type. The type class has less than $2^{nH(P_{x^n})}$ elements (where P_{x^n} is the type of the sequence x^n) as shown later.

Proof Cont.

- We can use the same technique for the case of a nonbinary finite alphabet.
- We first describe the type of the sequence (the empirical frequency of occurrence of each of the alphabet symbols as defined later) using $(|\mathcal{X}| - 1) \log n$ bits (the frequency of the last symbol can be calculated from the frequencies of the rest).
- Then we describe the index of the sequence within the set of all sequences having the same type. The type class has less than $2^{nH(P_{x^n})}$ elements (where P_{x^n} is the type of the sequence x^n) as shown later.

Proof Cont.

- We can use the same technique for the case of a nonbinary finite alphabet.
- We first describe the type of the sequence (the empirical frequency of occurrence of each of the alphabet symbols as defined later) using $(|\mathcal{X}| - 1) \log n$ bits (the frequency of the last symbol can be calculated from the frequencies of the rest).
- Then we describe the index of the sequence within the set of all sequences having the same type. The type class has less than $2^{nH(P_{x^n})}$ elements (where P_{x^n} is the type of the sequence x^n) as shown later.

Proof Cont.

Therefore the two-stage description of a string x^n has length

$$K(x_n|n) \leq nH(P_{x^n})) + (|\mathcal{X} - 1|) \log n + c \qquad (28)$$

Again, taking the expectation and applying Jensen's inequality as in the binary case, we obtain

$$EK(X_n|n) \leq nH(X) + (|\mathcal{X} - 1|) \log n + c \qquad (29)$$

Dividing this by n yields the upper bound of the theorem. $\square$

Type of Sequence

Let $X_1, X_2, \ldots, X_n$ be a sequence of n symbols from an alphabet $\mathcal{X} = \{a_1, a_2, \ldots, a_{|\mathcal{X}|}\}$. We use the notation x^n and $\mathbf{x}$ interchangeably to denote a sequence $x_1, x_2, \ldots, x_n$.

Definition

The `type` *$P_{\mathbf{x}}$ (or empirical probability distribution) of a sequence $x_1, x_2, \ldots, x_n$ is the relative proportion of occurrences of each symbol of $\mathcal{X}$ (i.e., $P_{\mathbf{x}}(a) = N(a|\mathbf{x})/n$ for all $a \in \mathcal{X}$, where $N(a|\mathbf{x})$ is the number of times the symbol a occurs in the sequence $\mathbf{x} \in \mathcal{X}^n$).It is a probability mass function on $\mathcal{X}$.*

Type of Sequence

Definition

Let $\mathcal{P}_n$ denote the `set of types` *with denominator n.*

For example, if $\mathcal{X} = \{0, 1\}$, the set of possible types with denominator n is

$$\mathcal{P}_n = \left\{ (P(0), P(1)) : \left(\frac{0}{n}, \frac{n}{n}\right), \left(\frac{1}{n}, \frac{n-1}{n}\right), \ldots, \left(\frac{n}{n}, \frac{0}{n}\right) \right\}.$$

Type Class

Definition

If $P \in \mathcal{P}_n$, the set of sequences of length n and type P is called the `type class` *of P, denoted $T(P)$:*

$$T(P) = \{\mathbf{x} \in \mathcal{X}^n : P_{\mathbf{x}} = P\}.$$

The type class is sometimes called the composition class of P.

Type Class

Example

Let $\mathcal{X} = \{1, 2, 3\}$, a ternary alphabet. Let $\mathbf{x}$ *= 11321.Then the type* $P_{\mathbf{x}}$ *is:*

$$P_{\mathbf{x}}(1) = \frac{3}{5}, \quad P_{\mathbf{x}}(2) = \frac{1}{5}, \quad P_{\mathbf{x}}(3) = \frac{1}{5}.$$

The type class of $P_{\mathbf{x}}$ *is the set of all sequences of length 5 with three 1's, one 2, and one 3. There are 20 such sequences, and*

$$T(P_{\mathbf{x}}) = \{11123, 11132, 11213, \ldots, 32111\}.$$

The number of elements in $T(P)$*is*

$$|T(P)| = \binom{5}{3, 1, 1} = \frac{5!}{3!1!1!} = 20.$$

Probability to Same Type Class

Theorem

If $X_1, X_2, \ldots, X_n$ are drawn i.i.d. according to $Q(x)$, the probability of $\mathbf{x}$ depends only on its type and is given by

$$Q^n(x) = 2^{-n(H(P_{\mathbf{x}})+D(P_{\mathbf{x}}\|Q))}.$$

Probability to Same Type Class

Proof.

$$
\begin{aligned}
Q^n(\mathbf{x}) &= \prod_{i=1}^{n} Q(x_i) \\
&= \prod_{a \in \mathcal{X}} Q(a)^{N(a|\mathbf{x})} \\
&= \prod_{a \in \mathcal{X}} Q(a)^{nP_{\mathbf{x}}} \\
&= \prod_{a \in \mathcal{X}} 2^{nP_{\mathbf{x}} \log Q(a)} \\
&= \prod_{a \in \mathcal{X}} 2^{n(P_{\mathbf{x}} \log Q(a) - P_{\mathbf{x}} \log P_{\mathbf{x}} + P_{\mathbf{x}} \log P_{\mathbf{x}})} \\
&= 2^{n \sum_{a \in \mathcal{X}} (-P_{\mathbf{x}} \log \frac{P_{\mathbf{x}}}{Q(a)} + P_{\mathbf{x}} \log P_{\mathbf{x}})} \\
&= 2^{-n(H(P_{\mathbf{x}}) + D(P_{\mathbf{x}} \| Q))}.
\end{aligned}
$$

□

Probability to Same Type Class

Corollary

If **x** *is in the type class of Q, then*

$$Q^n(x) = 2^{-nH(Q)}.$$

Size of a Type Class

Theorem

For any type $P \in \mathcal{P}_n$,

$$|T(P)| \leq 2^{nH(P)}.$$

Proof. Since a type class must have probability ≤ 1, we have

$$\begin{aligned} 1 &\geq P^n(T(P)) \\ &= \sum_{\mathbf{x} \in T(P)} P^n(\mathbf{x}) \\ &= \sum_{\mathbf{x} \in T(P)} 2^{-nH(P)} \\ &= |T(P)| 2^{-nH(P)}, \end{aligned}$$

Thus, $|T(P)| \leq 2^{nH(P)}$. $\square$

Outline

Definition
- Models of Computation
- Definitions
- Examples

Kolmogorov Complexity and Entropy
- Prefix Code
- Kolmogorov Complexity and Entropy

Algorithmically Random and Incompressible
- Random and Incompressible
- Asymptotic Equipartition Property
- A Random Bernoulli Sequence

Ω

Kolmogorov Complexity and Universal Probability

Summary

Compression

- We now show that although there are some simple sequences, most sequences do not have simple descriptions.
- Hence, if we draw a sequence at random, we are likely to draw a complex sequence.
- The next theorem shows that the probability that a sequence can be compressed by more than k bits is no greater than 2^{-k}.

Compression

- We now show that although there are some simple sequences, most sequences do not have simple descriptions.
- Hence, if we draw a sequence at random, we are likely to draw a complex sequence.
- The next theorem shows that the probability that a sequence can be compressed by more than k bits is no greater than 2^{-k}.

Compression

- We now show that although there are some simple sequences, most sequences do not have simple descriptions.
- Hence, if we draw a sequence at random, we are likely to draw a complex sequence.
- The next theorem shows that the probability that a sequence can be compressed by more than k bits is no greater than 2^{-k}.

Compression

Theorem

Let $X_1, X_2, \ldots, X_n$ be drawn according to a Bernoulli$(\frac{1}{2})$ process. Then

$$P(K(X_1, X_2, \ldots, X_n | n) < n - k) < 2^{-k}. \quad (30)$$

Compression

Proof.

$$
\begin{aligned}
& P(K(X_1, X_2, \ldots, X_n|n) < n - k) \\
= & \sum_{(x_1, x_2, \ldots, x_n : K(x_1, x_2, \ldots, x_n|n) < n-k)} p(x_1, x_2, \ldots, x_n) \\
= & \sum_{(x_1, x_2, \ldots, x_n : K(x_1, x_2, \ldots, x_n|n) < n-k)} 2^{-n} \\
= & |(x_1, x_2, \ldots, x_n : K(x_1, x_2, \ldots, x_n|n) < n - k)| 2^{-n} \\
< & 2^{n-k} 2^{-n} \\
= & 2^{-k}.
\end{aligned}
$$

$\square$

Compression

Remark

Thus, most sequences have a complexity close to their length. For example, the fraction of sequences of length n that have complexity less than $n - 5$ is less than $1/32$.

Algorithmically Random

Definition

A sequence $x_1, x_2, \ldots, x_n$ is said to be `algorithmically random` *if*

$$K(x_1, x_2, \ldots, x_n | n) \geq n. \quad (31)$$

Remark

Note that by the counting argument, there exists, for each n, at least one sequence x^n such that $K(x^n|n) \geq n$.

Algorithmically Random

Definition

A sequence $x_1, x_2, \ldots, x_n$ is said to be `algorithmically random` *if*

$$K(x_1, x_2, \ldots, x_n | n) \geq n. \tag{31}$$

Remark

Note that by the counting argument, there exists, for each n, at least one sequence x^n such that $K(x^n | n) \geq n$.

Incompressible String

Definition

We call an infinite string x `incompressible` *if*

$$\lim_{n\to\infty} \frac{K(x_1, x_2, x_3, \ldots, x_n | n)}{n} = 1. \qquad (32)$$

Incompressible String

Theorem

(Strong law of large numbers for incompressible sequences) If a string $x_1, x_2, \ldots$ is incompressible, it satisfies the law of large numbers in the sense that

$$\frac{1}{n}\sum_{i=1}^{n} x_i \to \frac{1}{2}. \tag{33}$$

Hence the proportions of 0's and 1's in any incompressible string are almost equal.

Incompressible String

Proof.

- Let $\theta_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ denote the proportion of 1's in $x_1, x_2, \ldots, x_n$.
- Then using the method of Example in section 1 one can write a program of length $nH(\theta_n) + 2log(n\theta_n) + c$ to print x^n.
- Thus,

$$\frac{K(x^n|n)}{n} < H_0(\theta_n) + 2\frac{\log n}{n} + \frac{c'}{n}. \tag{34}$$

Incompressible String

Proof.

- Let $\theta_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ denote the proportion of 1's in $x_1, x_2, \ldots, x_n$.
- Then using the method of Example in section 1 one can write a program of length $nH(\theta_n) + 2log(n\theta_n) + c$ to print x^n.
- Thus,

$$\frac{K(x^n|n)}{n} < H_0(\theta_n) + 2\frac{\log n}{n} + \frac{c'}{n}. \qquad (34)$$

Incompressible String

Proof.

- Let$\theta_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ denote the proportion of 1's in $x_1, x_2, \ldots, x_n$.
- Then using the method of Example in section 1 one can write a program of length $nH(\theta_n) + 2log(n\theta_n) + c$ to print x^n.
- Thus,

$$\frac{K(x^n|n)}{n} < H_0(\theta_n) + 2\frac{\log n}{n} + \frac{c'}{n}. \tag{34}$$

Proof Cont.

By the incompressibility assumption, we also have the lower bound for large enough n,

$$1 - \varepsilon \leq \frac{K(x^n|n)}{n} < H_0(\theta_n) + 2\frac{\log n}{n} + \frac{c'}{n}. \qquad (35)$$

Thus,

$$H_0(\theta_n) > 1 - 2\frac{\log n + c'}{n} - \epsilon. \qquad (36)$$

Inspection of the graph of $H_0(p)$ (Figure (3)) shows that θ_n is close to $\frac{1}{2}$ for large n. Specifically, the inequality above implies that

$$\theta_n \in \left(\frac{1}{2} - \delta_n, \frac{1}{2} + \delta_n\right), \qquad (37)$$

Proof Cont.

where δ_n is chosen so that

$$H_0(\frac{1}{2} - \delta_n) = 1 - 2\frac{\log n + c_n + c'}{n}. \quad (38)$$

which implies that $\delta_n \to 0$ as $n \to \infty$. Thus, $\theta_n = \frac{1}{n}\sum_{i=1}^{n} x_i \to \frac{1}{2}$ as $n \to \infty$. $\square$

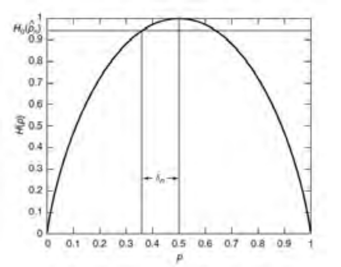

Figure: $H_0(P)$ vs. p

Incompressible String

Remark

- We have now proved that incompressible sequences look random in the sense that the proportion of 0's and 1's are almost equal.
- In general, we can show that if a sequence is incompressible, it will satisfy all computable statistical tests for randomness.
- Otherwise, identification of the test that x fails will reduce the descriptive complexity of x, yielding a contradiction.
- In this sense, the algorithmic test for randomness is the ultimate test, including within it all other computable tests for randomness.

Incompressible String

Remark

- *We have now proved that incompressible sequences look random in the sense that the proportion of 0's and 1's are almost equal.*
- *In general, we can show that if a sequence is incompressible, it will satisfy all computable statistical tests for randomness.*
- *Otherwise, identification of the test that x fails will reduce the descriptive complexity of x, yielding a contradiction.*
- *In this sense, the algorithmic test for randomness is the ultimate test, including within it all other computable tests for randomness.*

Incompressible String

Remark

- *We have now proved that incompressible sequences look random in the sense that the proportion of 0's and 1's are almost equal.*
- *In general, we can show that if a sequence is incompressible, it will satisfy all computable statistical tests for randomness.*
- *Otherwise, identification of the test that x fails will reduce the descriptive complexity of x, yielding a contradiction.*
- *In this sense, the algorithmic test for randomness is the ultimate test, including within it all other computable tests for randomness.*

Incompressible String

Remark

- *We have now proved that incompressible sequences look random in the sense that the proportion of 0's and 1's are almost equal.*
- *In general, we can show that if a sequence is incompressible, it will satisfy all computable statistical tests for randomness.*
- *Otherwise, identification of the test that x fails will reduce the descriptive complexity of x, yielding a contradiction.*
- *In this sense, the algorithmic test for randomness is the ultimate test, including within it all other computable tests for randomness.*

Incompressible String

Remark

- *We have now proved that incompressible sequences look random in the sense that the proportion of 0's and 1's are almost equal.*
- *In general, we can show that if a sequence is incompressible, it will satisfy all computable statistical tests for randomness.*
- *Otherwise, identification of the test that x fails will reduce the descriptive complexity of x, yielding a contradiction.*
- *In this sense, the algorithmic test for randomness is the ultimate test, including within it all other computable tests for randomness.*

Outline

AEP

- In information theory, the analog of the law of large numbers is the asymptotic equipartition property (AEP).
- The law of large numbers states that for independent, identically distributed (i.i.d.) random variables, $\frac{1}{n}\sum_{i=1}^{n} X_i$ is close to its expected value EX for large values of n.
- The AEP states that $\frac{1}{n}\log\frac{1}{p(X_1,X_2,\ldots,X_n)}$ is close to the entropy H,where $X_1, X_2, \ldots, X_n$ are i.i.d. random variables.
- Thus, the probability $p(X_1, X_2, \ldots, X_n)$ assigned to an observed sequence will be close to 2^{-nH}.

AEP

- In information theory, the analog of the law of large numbers is the asymptotic equipartition property (AEP).
- The law of large numbers states that for independent, identically distributed (i.i.d.) random variables, $\frac{1}{n}\sum_{i=1}^{n} X_i$ is close to its expected value EX for large values of n.
- The AEP states that $\frac{1}{n}\log\frac{1}{p(X_1,X_2,\ldots,X_n)}$ is close to the entropy H,where $X_1, X_2, \ldots, X_n$ are i.i.d. random variables.
- Thus, the probability $p(X_1, X_2, \ldots, X_n)$ assigned to an observed sequence will be close to 2^{-nH}.

AEP

- In information theory, the analog of the law of large numbers is the asymptotic equipartition property (AEP).
- The law of large numbers states that for independent, identically distributed (i.i.d.) random variables, $\frac{1}{n}\sum_{i=1}^{n} X_i$ is close to its expected value EX for large values of n.
- The AEP states that $\frac{1}{n}\log\frac{1}{p(X_1,X_2,\ldots,X_n)}$ is close to the entropy H,where $X_1, X_2, \ldots, X_n$ are i.i.d. random variables.
- Thus, the probability $p(X_1, X_2, \ldots, X_n)$ assigned to an observed sequence will be close to 2^{-nH}.

AEP

- In information theory, the analog of the law of large numbers is the asymptotic equipartition property (AEP).
- The law of large numbers states that for independent, identically distributed (i.i.d.) random variables, $\frac{1}{n}\sum_{i=1}^{n} X_i$ is close to its expected value EX for large values of n.
- The AEP states that $\frac{1}{n}\log\frac{1}{p(X_1,X_2,\ldots,X_n)}$ is close to the entropy H,where $X_1, X_2, \ldots, X_n$ are i.i.d. random variables.
- Thus, the probability $p(X_1, X_2, \ldots, X_n)$ assigned to an observed sequence will be close to 2^{-nH}.

AEP

- This enables us to divide the set of all sequences into two sets, the typical set, where the sample entropy is close to the true entropy, and the nontypical set, which contains the other sequences.
- Any property that is proved for the typical sequences will then be true with high probability and will determine the average behavior of a large sample.

AEP

- This enables us to divide the set of all sequences into two sets, the typical set, where the sample entropy is close to the true entropy, and the nontypical set, which contains the other sequences.
- Any property that is proved for the typical sequences will then be true with high probability and will determine the average behavior of a large sample.

AEP

Theorem
If $X_1, X_2, \ldots$ are i.i.d.$\sim p(x)$, then

$$-\frac{1}{n}\log p(X_1, X_2, \ldots, X_n) \to H(X) \text{ in probability.} \qquad (39)$$

Proof. Functions of independent random variables are also independent random variables. Thus, since the X_i are i.i.d., so are $\log p(X_i)$.Hence,by the weak law of large numbers,

$$\begin{aligned}-\frac{1}{n}\log p(X_1, X_2, \ldots, X_n) &= -\frac{1}{n}\sum_i \log p(X_i) \\ &\to -E\log p(X) \text{ in probability} \\ &= H(x).\end{aligned}$$

□

Typical Set

Definition

The `typical set` *$A_\epsilon^{(n)}$ with respect to $p(x)$ is the set of sequences $(X_1, X_2, \ldots, X_n) \in \mathcal{X}^n$ with the property*

$$2^{-n(H(X)+\epsilon)} \leq p(X_1, X_2, \ldots, X_n) \leq 2^{-n(H(X)-\epsilon)}. \qquad (40)$$

Typical Set

As a consequence of the AEP, we can show that the set $A_\epsilon^{(n)}$ has the following properties:

Theorem

1. *If* $(X_1, X_2, \ldots, X_n) \in A_\epsilon^{(n)}$ *, then* $H(X) - \epsilon \leq -\frac{1}{n} \log p(X_1, X_2, \ldots, X_n) \leq H(X) + \epsilon$*.*
2. $\mathbf{Pr}(A_\epsilon^{(n)}) > 1 - \epsilon$ *for n sufficiently large.*
3. $|A_\epsilon^{(n)}| \leq 2^{n(H(x)+\epsilon)}$*.*
4. $|A_\epsilon^{(n)}| \geq (1 - \epsilon) 2^{n(H(x)-\epsilon)}$ *for n sufficiently large.*

Typical Set

As a consequence of the AEP, we can show that the set $A_\epsilon^{(n)}$ has the following properties:

Theorem

1. *If* $(X_1, X_2, \ldots, X_n) \in A_\epsilon^{(n)}$ *, then* $H(X) - \epsilon \leq -\frac{1}{n} \log p(X_1, X_2, \ldots, X_n) \leq H(X) + \epsilon$*.*
2. $\mathbf{Pr}(A_\epsilon^{(n)}) > 1 - \epsilon$ *for n sufficiently large.*
3. $|A_\epsilon^{(n)}| \leq 2^{n(H(x)+\epsilon)}$.
4. $|A_\epsilon^{(n)}| \geq (1 - \epsilon)2^{n(H(x)-\epsilon)}$ *for n sufficiently large.*

Typical Set

As a consequence of the AEP, we can show that the set $A_\epsilon^{(n)}$ has the following properties:

Theorem

1. *If* $(X_1, X_2, \dots, X_n) \in A_\epsilon^{(n)}$ *, then* $H(X) - \epsilon \le -\frac{1}{n} \log p(X_1, X_2, \dots, X_n) \le H(X) + \epsilon$.
2. $\mathbf{Pr}(A_\epsilon^{(n)}) > 1 - \epsilon$ *for n sufficiently large.*
3. $|A_\epsilon^{(n)}| \le 2^{n(H(x)+\epsilon)}$.
4. $|A_\epsilon^{(n)}| \ge (1-\epsilon)2^{n(H(x)-\epsilon)}$ *for n sufficiently large.*

Typical Set

As a consequence of the AEP, we can show that the set $A_\epsilon^{(n)}$ has the following properties:

Theorem

1. *If* $(X_1, X_2, \ldots, X_n) \in A_\epsilon^{(n)}$ *, then* $H(X) - \epsilon \leq -\frac{1}{n} \log p(X_1, X_2, \ldots, X_n) \leq H(X) + \epsilon$*.*
2. $\mathbf{Pr}(A_\epsilon^{(n)}) > 1 - \epsilon$ *for n sufficiently large.*
3. $|A_\epsilon^{(n)}| \leq 2^{n(H(x)+\epsilon)}$*.*
4. $|A_\epsilon^{(n)}| \geq (1 - \epsilon) 2^{n(H(x)-\epsilon)}$ *for n sufficiently large.*

Typical Set

As a consequence of the AEP, we can show that the set $A_\epsilon^{(n)}$ has the following properties:

Theorem

1. *If* $(X_1, X_2, \ldots, X_n) \in A_\epsilon^{(n)}$ *, then* $H(X) - \epsilon \leq -\frac{1}{n} \log p(X_1, X_2, \ldots, X_n) \leq H(X) + \epsilon$*.*
2. $\mathbf{Pr}(A_\epsilon^{(n)}) > 1 - \epsilon$ *for n sufficiently large.*
3. $|A_\epsilon^{(n)}| \leq 2^{n(H(x)+\epsilon)}$*.*
4. $|A_\epsilon^{(n)}| \geq (1-\epsilon)2^{n(H(x)-\epsilon)}$ *for n sufficiently large.*

Typical Set

Proof. Property (3) follows from:

$$
\begin{aligned}
1 &= \sum_{\mathbf{x}\in\mathcal{X}^n} p(\mathbf{x}) \\
&\geq \sum_{\mathbf{x}\in A_\epsilon^{(n)}} p(\mathbf{x}) \\
&\geq \sum_{\mathbf{x}\in A_\epsilon^{(n)}} 2^{-n(H(X)+\epsilon)} \\
&= 2^{-n(H(X)+\epsilon)} |A_\epsilon^{(n)}|.
\end{aligned}
$$

Proof Cont.

For sufficiently large n,

$$\mathbf{Pr}(A_\epsilon^{(n)}) > 1 - \epsilon,$$

so that

$$\begin{aligned} 1 - \epsilon < \mathbf{Pr}(A_\epsilon^{(n)}) \\ &\leq \sum_{\mathbf{x} \in A_\epsilon^{(n)}} 2^{-n(H(X)-\epsilon)} \\ &= 2^{-n(H(X)-\epsilon)} |A_\epsilon^{(n)}|. \end{aligned}$$

$\square$

Outline

A Random Bernoulli Sequence

Theorem

Let $X_1, X_2, \ldots, X_n$ be drawn i.i.d.$\sim$ Bernoulli(θ). Then

$$\frac{1}{n} K(X_1, X_2, \ldots, X_n | n) \to H_0(\theta) \text{in probability.} \quad (41)$$

A Random Bernoulli Sequence

Proof. Let $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ denote the proportion of 1's in $X_1, X_2, \ldots, X_n$. Then using the method described in section 1, we have

$$K(X_1, X_2, \ldots, X_n|n) < nH_0(\bar{X}_n) + 2\log n + c, \qquad (42)$$

and since by the weak law of large numbers, $\bar{X}_n \to \theta$ in probability, we have

$$\mathbf{Pr}(\frac{1}{n}K(X_1, X_2, \ldots, X_n|n) - H_0(\theta) \geq \epsilon) \to 0. \qquad (43)$$

Proof cont.

- Conversely, we can bound the number of sequences with complexity significantly lower than the entropy.
- From the AEP, we can divide the set of sequences into the typical set and the nontypical set.
- There are at least $(1-\epsilon)2^{n(H_0(\theta)-\epsilon)}$ sequences in the typical set.
- At most $2^{n(H_0(\theta)-c)}$ sequences in the typical set. of these typical sequences can have a complexity less than $n(H_0(\theta)-c)$.

Proof cont.

- Conversely, we can bound the number of sequences with complexity significantly lower than the entropy.
- From the AEP, we can divide the set of sequences into the typical set and the nontypical set.
- There are at least $(1-\epsilon)2^{n(H_0(\theta)-\epsilon)}$ sequences in the typical set.
- At most $2^{n(H_0(\theta)-c)}$ sequences in the typical set. of these typical sequences can have a complexity less than $n(H_0(\theta)-c)$.

Proof cont.

- Conversely, we can bound the number of sequences with complexity significantly lower than the entropy.
- From the AEP, we can divide the set of sequences into the typical set and the nontypical set.
- There are at least $(1-\epsilon)2^{n(H_0(\theta)-\epsilon)}$ sequences in the typical set.
- At most $2^{n(H_0(\theta)-c)}$ sequences in the typical set. of these typical sequences can have a complexity less than $n(H_0(\theta)-c)$.

Proof cont.

- Conversely, we can bound the number of sequences with complexity significantly lower than the entropy.
- From the AEP, we can divide the set of sequences into the typical set and the nontypical set.
- There are at least $(1-\epsilon)2^{n(H_0(\theta)-\epsilon)}$ sequences in the typical set.
- At most $2^{n(H_0(\theta)-c)}$ sequences in the typical set. of these typical sequences can have a complexity less than $n(H_0(\theta)-c)$.

Proof cont.

The probability that the complexity of the random sequence is less than $n(H_0(\theta) - c)$is

$$\begin{aligned}
&\mathbf{Pr}(K(X^n|n) < n(H_0(\theta) - c)) \\
&\leq \mathbf{Pr}(X^n \notin A_\epsilon^{(n)}) + \mathbf{Pr}(X^n \in A_\epsilon^{(n)}, K(K^n|n) < n(H_0(\theta) - c)) \\
&\leq \epsilon + \sum_{X^n \in A_\epsilon^{(n)}, K(K^n|n) < n(H_0(\theta) - c)} p(x^n) \\
&\leq \epsilon + \sum_{X^n \in A_\epsilon^{(n)}, K(K^n|n) < n(H_0(\theta) - c)} 2^{-n(H_0(\theta) - \epsilon)} \\
&\leq \epsilon + 2^{n(H_0(\theta) - c)} 2^{-n(H_0(\theta) - \epsilon)} \\
&= \epsilon + 2^{-n(c-\epsilon)},
\end{aligned}$$

which is arbitrarily small for appropriate choice of ϵ, n, and c.

Proof cont.

Hence with high probability, the Kolmogorov complexity of the random sequence is close to the entropy, and we have

$$\frac{1}{n}K(X_1, X_2, \ldots, X_n|n) \rightarrow H_0(\theta)\text{in probability.} \qquad (44)$$

$\square$

Epimenides Liar Paradox

Consider the following paradoxical statement:

This statement is false.

This paradox is sometimes stated in a two-statement form:

The next statement is false.
The preceding statement is true.

It illustrates the pitfalls involved in self-reference.

Gödel's Incompleteness Theorem

- In 1931, Gödel used this idea of self-reference to show that any interesting system of mathematics is not complete; there are statements in the system that are true but that cannot be proved within the system.
- To accomplish this, he translated theorems and proofs into integers and constructed a statement of the above form, which can therefore not be proved true or false.

Gödel's Incompleteness Theorem

- In 1931, Gödel used this idea of self-reference to show that any interesting system of mathematics is not complete; there are statements in the system that are true but that cannot be proved within the system.
- To accomplish this, he translated theorems and proofs into integers and constructed a statement of the above form, which can therefore not be proved true or false.

The Halting Problem

- The halting problem is an essential example of Gödel's incompleteness theorem.
- In essence, it states that for any computational model, there is no general algorithm to decide whether a program will halt or not (go on forever).
- Note that it is not a statement about any specific program. The halting problem says that we cannot answer this question for all programs.

The Halting Problem

- The halting problem is an essential example of Gödel's incompleteness theorem.
- In essence, it states that for any computational model, there is no general algorithm to decide whether a program will halt or not (go on forever).
- Note that it is not a statement about any specific program. The halting problem says that we cannot answer this question for all programs.

The Halting Problem

- The halting problem is an essential example of Gödel's incompleteness theorem.
- In essence, it states that for any computational model, there is no general algorithm to decide whether a program will halt or not (go on forever).
- Note that it is not a statement about any specific program. The halting problem says that we cannot answer this question for all programs.

The Noncomputability of Kolmogorov Complexity

- One of the consequences of the nonexistence of an algorithm for the halting problem is the noncomputability of Kolmogorov complexity.
- The only way to find the shortest program in general is to try all short programs and see which of them can do the job.
- There is no effective (finite mechanical) way to tell whether or not they will halt and what they will print out.

The Noncomputability of Kolmogorov Complexity

- One of the consequences of the nonexistence of an algorithm for the halting problem is the noncomputability of Kolmogorov complexity.
- The only way to find the shortest program in general is to try all short programs and see which of them can do the job.
- There is no effective (finite mechanical) way to tell whether or not they will halt and what they will print out.

The Noncomputability of Kolmogorov Complexity

- One of the consequences of the nonexistence of an algorithm for the halting problem is the noncomputability of Kolmogorov complexity.
- The only way to find the shortest program in general is to try all short programs and see which of them can do the job.
- There is no effective (finite mechanical) way to tell whether or not they will halt and what they will print out.

Chaitin's Mystical, Magical Number, Ω

Definition

$$\Omega = \sum_{p:\mathcal{U}(p) halts} 2^{-\ell(p)}. \tag{45}$$

Note that $\Omega = \mathbf{Pr}(\mathcal{U}(p)\text{halts})$, the probability that the given universal computer halts when the input to the computer is a binary string drawn according to a Bernoulli($\frac{1}{2}$) process.

Chaitin's Mystical, Magical Number, Ω

Since the programs that halt are prefix-free, their lengths satisfy the Kraft inequality, and hence the sum above is always between 0 and 1. Let $\Omega_n = .\omega_1\omega_2\ldots\omega_n$ denote the first n bits of Ω.

Chaitin's Mystical, Magical Number, Ω

The properties of Ω are as follows:

1. Ω is noncomputable.
2. Ω is a "philosopher's stone".
3. Ω is algorithmically random.

Chaitin's Mystical, Magical Number, Ω

The properties of Ω are as follows:

1. Ω is noncomputable.
2. Ω is a "philosopher's stone".
3. Ω is algorithmically random.

Chaitin's Mystical, Magical Number, Ω

The properties of Ω are as follows:

1. Ω is noncomputable.
2. Ω is a "philosopher's stone".
3. Ω is algorithmically random.

Ω : noncomputable

There is no effective (finite, mechanical) way to check whether arbitrary programs halt (the halting problem), so there is no effective way to compute Ω.

Ω : a "philosopher's stone"

- Knowing Ω to an accuracy of n bits will enable us to decide the truth of any provable or finitely refutable mathematical theorem that can be written in less than n bits.
- Actually, all that this means is that given n bits of Ω, there is an effective procedure to decide the truth of n-bit theorems; the procedure may take an arbitrarily long (but finite) time.
- Without knowing Ω, it is not possible to check the truth or falsity of every theorem by an effective procedure (Gödel's incompleteness theorem).

Ω : a "philosopher's stone"

- Knowing Ω to an accuracy of n bits will enable us to decide the truth of any provable or finitely refutable mathematical theorem that can be written in less than n bits.
- Actually, all that this means is that given n bits of Ω, there is an effective procedure to decide the truth of n-bit theorems; the procedure may take an arbitrarily long (but finite) time.
- Without knowing Ω, it is not possible to check the truth or falsity of every theorem by an effective procedure (Gödel's incompleteness theorem).

Ω : a "philosopher's stone"

- Knowing Ω to an accuracy of n bits will enable us to decide the truth of any provable or finitely refutable mathematical theorem that can be written in less than n bits.
- Actually, all that this means is that given n bits of Ω, there is an effective procedure to decide the truth of n-bit theorems; the procedure may take an arbitrarily long (but finite) time.
- Without knowing Ω, it is not possible to check the truth or falsity of every theorem by an effective procedure (Gödel's incompleteness theorem).

Ω : a "philosopher's stone"

We run all programs until the sum of the masses $2^{-\ell(p)}$ contributed by programs that halt equals or exceeds $\Omega_n = .\omega_1\omega_2 \ldots \omega_n$. Then, since

$$\Omega - \Omega_n < 2^{-n}, \qquad (46)$$

we know that the sum of all further contributions of the form $2^{-\ell(p)}$ to Ω from programs that halt must also be less than 2^{-n}.This implies that no program of length $\leq n$ that has not yet halted will ever halt, which enables us to decide the halting or nonhalting of all programs of length $\leq n$.

Ω :algorithmically random

Theorem

Ω cannot be compressed by more than a constant; that is, there exists a constant c such that

$$K(\omega_1\omega_2\ldots\omega_n) \geq n - c, \text{ for all } n. \quad (47)$$

Ω :algorithmically random

Proof.

- If we are given n bits of Ω, we can determine whether or not any program of length $\leq n$ halts.
- Using $K(\Omega_n)$ bits, we can calculate n bits of Ω, then we can generate a list of all programs of length $\leq n$ that halt, together with their corresponding outputs.
- We find the first string x_0 that is not on this list. The string x_0 is then the shortest string with Kolmogorov complexity $K(x_0) \geq n$.
- The complexity of this program to print x_0 is $K(\Omega_n) + c$, which must be at least as long as the shortest program for x_0.Consequently, for all n,

$$K(\Omega_n) + c \geq K(x_0) > n.$$

□

Ω :algorithmically random

Proof.

- If we are given n bits of Ω, we can determine whether or not any program of length $\leq n$ halts.
- Using $K(\Omega_n)$ bits, we can calculate n bits of Ω, then we can generate a list of all programs of length $\leq n$ that halt, together with their corresponding outputs.
- We find the first string x_0 that is not on this list. The string x_0 is then the shortest string with Kolmogorov complexity $K(x_0) \geq n$.
- The complexity of this program to print x_0 is $K(\Omega_n) + c$, which must be at least as long as the shortest program for x_0.Consequently, for all n,

$$K(\Omega_n) + c \geq K(x_0) > n.$$

□

Ω :algorithmically random

Proof.

- If we are given n bits of Ω, we can determine whether or not any program of length $\leq n$ halts.
- Using $K(\Omega_n)$ bits, we can calculate n bits of Ω, then we can generate a list of all programs of length $\leq n$ that halt, together with their corresponding outputs.
- We find the first string x_0 that is not on this list. The string x_0 is then the shortest string with Kolmogorov complexity $K(x_0) \geq n$.
- The complexity of this program to print x_0 is $K(\Omega_n) + c$, which must be at least as long as the shortest program for x_0.Consequently, for all n,

$$K(\Omega_n) + c \geq K(x_0) > n.$$

□

Ω :algorithmically random

Proof.

- If we are given n bits of Ω, we can determine whether or not any program of length $\leq n$ halts.
- Using $K(\Omega_n)$ bits, we can calculate n bits of Ω, then we can generate a list of all programs of length $\leq n$ that halt, together with their corresponding outputs.
- We find the first string x_0 that is not on this list. The string x_0 is then the shortest string with Kolmogorov complexity $K(x_0) \geq n$.
- The complexity of this program to print x_0 is $K(\Omega_n) + c$, which must be at least as long as the shortest program for x_0.Consequently, for all n,

$$K(\Omega_n) + c \geq K(x_0) > n.$$

□

Universal Probability

- From our earlier discussions, it is clear that most sequences of length n have complexity close to n.
- Since the probability of an input program p is $2^{-\ell(p)}$,shorter programs are much more probable than longer ones; and when they produce long strings, shorter programs do not produce random strings; they produce strings with simply described structure.
- The probability distribution on the output strings is far from uniform. Under the computer-induced distribution, simple strings are more likely than complicated strings of the same length.

Universal Probability

- From our earlier discussions, it is clear that most sequences of length n have complexity close to n.
- Since the probability of an input program p is $2^{-\ell(p)}$,shorter programs are much more probable than longer ones; and when they produce long strings, shorter programs do not produce random strings; they produce strings with simply described structure.
- The probability distribution on the output strings is far from uniform. Under the computer-induced distribution, simple strings are more likely than complicated strings of the same length.

Universal Probability

- From our earlier discussions, it is clear that most sequences of length n have complexity close to n.
- Since the probability of an input program p is $2^{-\ell(p)}$,shorter programs are much more probable than longer ones; and when they produce long strings, shorter programs do not produce random strings; they produce strings with simply described structure.
- The probability distribution on the output strings is far from uniform. Under the computer-induced distribution, simple strings are more likely than complicated strings of the same length.

Universal Probability

Definition

The `universal probability` *of a string x is*

$$P_{\mathcal{U}}(x) = \sum_{p:\mathcal{U}(p)=x} 2^{-\ell(p)} = \mathbf{Pr}(\mathcal{U}(p) = x), \qquad (48)$$

which is the probability that a program randomly drawn as a sequence of fair coin flips $p_1, p_2, \ldots$ will print out the string x.

Universal Probability

This probability mass function is called universal because of the following theorem.

Theorem

For every computer $\mathcal{A}$,

$$P_{\mathcal{U}}(x) \geq c'_{\mathcal{A}} P_{\mathcal{A}}(x) \tag{49}$$

for every string $x \in \{0,1\}^*$, where the constant $c'_{\mathcal{A}}$ depends only on $\mathcal{U}$ and $\mathcal{A}$.

Universal Probability

Proof. From the discussion of Section 2, we recall that for every program p' for $\mathcal{A}$ that prints x, there exists a program p for $\mathcal{U}$ of length not more than $\ell(p') + c'_{\mathcal{A}}$ produced by prefixing a simulation program for $\mathcal{A}$. Hence,

$$P_{\mathcal{U}}(x) = \sum_{p:\mathcal{U}(p)=x} 2^{-\ell(p)} = \sum_{p':\mathcal{A}(p')=x} 2^{-\ell(p')-c'_{\mathcal{A}}} = c'_{\mathcal{A}} P_{\mathcal{A}}(x),$$

□

Universal Probability

Remark

- *Any sequence drawn according to a computable probability mass function on binary strings can be considered to be produced by some computer $\mathcal{A}$ acting on a random input (via the probability inverse transformation acting on a random input).*
- *Hence, the universal probability distribution includes a mixture of all computable probability distributions.*

Universal Probability

Remark

- *Any sequence drawn according to a computable probability mass function on binary strings can be considered to be produced by some computer $\mathcal{A}$ acting on a random input (via the probability inverse transformation acting on a random input).*
- *Hence, the universal probability distribution includes a mixture of all computable probability distributions.*

Universal Probability

Remark

- *Any sequence drawn according to a computable probability mass function on binary strings can be considered to be produced by some computer $\mathcal{A}$ acting on a random input (via the probability inverse transformation acting on a random input).*
- *Hence, the universal probability distribution includes a mixture of all computable probability distributions.*

Universal Probability

Remark

(Bounded likelihood ratio)

- *In particular, Theorem above guarantees that a likelihood ratio test of the hypothesis that X is drawn according to $\mathcal{U}$ versus the hypothesis that it is drawn according to $\mathcal{A}$ will have bounded likelihood ratio.*
- *If $\mathcal{U}$ and $\mathcal{A}$ are universal, then $P_{\mathcal{U}}(x)/P_{\mathcal{A}}(x)$ is bounded away from 0 and infinity for all x.*
- *Apparently, $\mathcal{U}$, which is a mixture of all computable distributions, can never be rejected completely as the true distribution of any data drawn according to some computable probability distribution.*

Universal Probability

Remark

(Bounded likelihood ratio)

- *In particular, Theorem above guarantees that a likelihood ratio test of the hypothesis that X is drawn according to $\mathcal{U}$ versus the hypothesis that it is drawn according to $\mathcal{A}$ will have bounded likelihood ratio.*
- *If $\mathcal{U}$ and $\mathcal{A}$ are universal, then $P_{\mathcal{U}}(x)/P_{\mathcal{A}}(x)$ is bounded away from 0 and infinity for all x.*
- *Apparently, $\mathcal{U}$, which is a mixture of all computable distributions, can never be rejected completely as the true distribution of any data drawn according to some computable probability distribution.*

Universal Probability

Remark

(Bounded likelihood ratio)

- *In particular, Theorem above guarantees that a likelihood ratio test of the hypothesis that X is drawn according to $\mathcal{U}$ versus the hypothesis that it is drawn according to $\mathcal{A}$ will have bounded likelihood ratio.*
- *If $\mathcal{U}$ and $\mathcal{A}$ are universal, then $P_{\mathcal{U}}(x)/P_{\mathcal{A}}(x)$ is bounded away from 0 and infinity for all x.*
- *Apparently, $\mathcal{U}$, which is a mixture of all computable distributions, can never be rejected completely as the true distribution of any data drawn according to some computable probability distribution.*

Universal Probability

Remark

(Bounded likelihood ratio)

- *In particular, Theorem above guarantees that a likelihood ratio test of the hypothesis that X is drawn according to $\mathcal{U}$ versus the hypothesis that it is drawn according to $\mathcal{A}$ will have bounded likelihood ratio.*
- *If $\mathcal{U}$ and $\mathcal{A}$ are universal, then $P_{\mathcal{U}}(x)/P_{\mathcal{A}}(x)$ is bounded away from 0 and infinity for all x.*
- *Apparently, $\mathcal{U}$, which is a mixture of all computable distributions, can never be rejected completely as the true distribution of any data drawn according to some computable probability distribution.*

Kolmogorov Complexity and Universal Probability

Theorem

(Equivalence of $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$) There exists a constant c, independent of x, such that

$$2^{-K(x)} \leq P_{\mathcal{U}}(x) \leq c2^{-K(x)} \tag{50}$$

for all strings x. Thus, the universal probability of a string x is determined essentially by its Kolmogorov complexity.

Remark

This implies that $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$ have equal status as universal complexity measures, since

$$K(x) - c' \leq \log \frac{1}{P_{\mathcal{U}}(x)} \leq K(x) \tag{51}$$

Remark

- *Notice the striking similarity between the relationship of* $K(x)$ *and* $\log \frac{1}{P_{\mathcal{U}}(x)}$ *in Kolmogorov complexity and the relationship of* $H(x)$ *and* $\log \frac{1}{p(x)}$ *in information theory.*
- *The ideal Shannon code length assignment* $\ell(x) = \log \frac{1}{p(x)}$ *achieves an average description length* $H(x)$*,while in Kolmogorov complexity theory, the ideal description length* $\log \frac{1}{P_{\mathcal{U}}(x)}$ *is almost equal to* $K(x)$*.*
- *Thus,* $\log \frac{1}{p(x)}$ *is the natural notion of descriptive complexity of* x *in algorithmic as well as probabilistic settings.*

Remark

- *Notice the striking similarity between the relationship of $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$ in Kolmogorov complexity and the relationship of $H(x)$ and $\log \frac{1}{p(x)}$ in information theory.*
- *The ideal Shannon code length assignment $\ell(x) = \log \frac{1}{p(x)}$ achieves an average description length $H(x)$, while in Kolmogorov complexity theory, the ideal description length $\log \frac{1}{P_{\mathcal{U}}(x)}$ is almost equal to $K(x)$.*
- *Thus, $\log \frac{1}{p(x)}$ is the natural notion of descriptive complexity of x in algorithmic as well as probabilistic settings.*

Remark

- *Notice the striking similarity between the relationship of* $K(x)$ *and* $\log \frac{1}{P_{\mathcal{U}}(x)}$ *in Kolmogorov complexity and the relationship of* $H(x)$ *and* $\log \frac{1}{p(x)}$ *in information theory.*
- *The ideal Shannon code length assignment* $\ell(x) = \log \frac{1}{p(x)}$ *achieves an average description length* $H(x)$*,while in Kolmogorov complexity theory, the ideal description length* $\log \frac{1}{P_{\mathcal{U}}(x)}$ *is almost equal to* $K(x)$*.*
- *Thus,* $\log \frac{1}{p(x)}$ *is the natural notion of descriptive complexity of* x *in algorithmic as well as probabilistic settings.*

Remark

- *Notice the striking similarity between the relationship of* $K(x)$ *and* $\log \frac{1}{P_{\mathcal{U}}(x)}$ *in Kolmogorov complexity and the relationship of* $H(x)$ *and* $\log \frac{1}{p(x)}$ *in information theory.*
- *The ideal Shannon code length assignment* $\ell(x) = \log \frac{1}{p(x)}$ *achieves an average description length* $H(x)$*,while in Kolmogorov complexity theory, the ideal description length* $\log \frac{1}{P_{\mathcal{U}}(x)}$ *is almost equal to* $K(x)$.
- *Thus,* $\log \frac{1}{p(x)}$ *is the natural notion of descriptive complexity of* x *in algorithmic as well as probabilistic settings.*

Proof. Our objective in the proof is to find a short program to describe the strings that have high $P_{\mathcal{U}}(x)$.

- We want to construct a code tree in such a way that strings with high probability have low depth.
- Since we cannot calculate the probability of a string, we do not know a priori the depth of the string on the tree.
- Instead, we assign x successively to the nodes of the tree, assigning x to nodes closer and closer to the root as our estimate of $P_{\mathcal{U}}(x)$ improves.
- We want the computer to be able to recreate the tree and use the lowest depth node corresponding to the string x to reconstruct the string.

Proof. Our objective in the proof is to find a short program to describe the strings that have high $P_{\mathcal{U}}(x)$.

- We want to construct a code tree in such a way that strings with high probability have low depth.
- Since we cannot calculate the probability of a string, we do not know a priori the depth of the string on the tree.
- Instead, we assign x successively to the nodes of the tree, assigning x to nodes closer and closer to the root as our estimate of $P_{\mathcal{U}}(x)$ improves.
- We want the computer to be able to recreate the tree and use the lowest depth node corresponding to the string x to reconstruct the string.

Proof. Our objective in the proof is to find a short program to describe the strings that have high $P_{\mathcal{U}}(x)$.

- We want to construct a code tree in such a way that strings with high probability have low depth.
- Since we cannot calculate the probability of a string, we do not know a priori the depth of the string on the tree.
- Instead, we assign x successively to the nodes of the tree, assigning x to nodes closer and closer to the root as our estimate of $P_{\mathcal{U}}(x)$ improves.
- We want the computer to be able to recreate the tree and use the lowest depth node corresponding to the string x to reconstruct the string.

Proof. Our objective in the proof is to find a short program to describe the strings that have high $P_{\mathcal{U}}(x)$.

- We want to construct a code tree in such a way that strings with high probability have low depth.
- Since we cannot calculate the probability of a string, we do not know a priori the depth of the string on the tree.
- Instead, we assign x successively to the nodes of the tree, assigning x to nodes closer and closer to the root as our estimate of $P_{\mathcal{U}}(x)$ improves.
- We want the computer to be able to recreate the tree and use the lowest depth node corresponding to the string x to reconstruct the string.

Definition. The Kolmogorov complexity $K_{\mathcal{U}}(x)$ of a string x is

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

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

Universality of Kolmogorov complexity. There exists a universal computer $\mathcal{U}$ such that for any other computer $\mathcal{A}$,

$$K_{\mathcal{U}}(x) \le K_{\mathcal{A}}(x) + c_{\mathcal{A}}.$$

for any string x, where the constant $c_{\mathcal{A}}$ does not depend on x. If $\mathcal{U}$ and $\mathcal{A}$ are universal, $|K_{\mathcal{U}}(x) - K_{\mathcal{A}}(x)| \le c$ for all x.

Upper bound on Kolmogorov complexity.

$$K_{\mathcal{U}}(x|\ell(x)) \le \ell(x) + c.$$

$$K_{\mathcal{U}}(x) \le K_{\mathcal{U}}(x|\ell(x)) + 2\log\ell(x) + c.$$

Kolmogorov complexity and entropy. If $X_1, X_2, \ldots$ are i.i.d. integer valued random variables with entropy H, there exists a constant c such that for all n,

$$H \leq E\frac{1}{n}K(x^n|n) \leq H + \frac{(|\mathcal{X}| - 1)\log n}{n} + \frac{c}{n}$$

Lower bound on Kolmogorov complexity. There are no more than 2^k strings x with complexity $K(x) < k$. If $X_1, X_2, \ldots X_n$ are drawn according to a Bernoulli$\left(\frac{1}{2}\right)$ process,

$$P(K(X_1, X_2, \ldots, X_n|n) < n - k) < 2^{-k}.$$

Definition. A sequence x is said to be incompressible if $\frac{K(x_1, x_2, x_3, \ldots, x_n|n)}{n} \to 1$.

Strong law of large numbers for incompressible sequences.

$$\frac{K(x_1, x_2, x_3, \ldots, x_n|n)}{n} \to 1 \Rightarrow \frac{1}{n}\sum_{i=1}^{n} x_i \to \frac{1}{2}.$$

Definition. $\Omega = \sum_{p:\mathcal{U}(p)\text{halts}} 2^{-\ell(p)} = \mathbf{Pr}(\mathcal{U}(p)\text{halts})$ is the probability that the computer halts when the input p to the computer is a binary string drawn according to a Bernoulli$(\frac{1}{2})$ process.

Properties of Ω.

Ω is noncomputable.

Ω is a "philosopher's stone".

Ω is algorithmically random.

Definition. The universal probability of a string x is

$$P_{\mathcal{U}}(x) = \sum_{p:\mathcal{U}(p)=x} 2^{-\ell(p)} = \mathbf{Pr}(\mathcal{U}(p) = x),$$

Universality of $P_{\mathcal{U}}(x)$. For every computer $\mathcal{A}$,

$$P_{\mathcal{U}}(x) \geq c'_{\mathcal{A}} P_{\mathcal{A}}(x)$$

for every string $x \in \{0,1\}^*$, where the constant $c'_{\mathcal{A}}$ depends only on $\mathcal{U}$ and $\mathcal{A}$.

Equivalence of $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$. There exists a constant c independent of x such that

$$\left| \log \frac{1}{P_{\mathcal{U}}(x)} - K(x) \right| \leq c$$

for all strings x. Thus, the universal probability of a string x is essentially determined by its Kolmogorov complexity.

References I

Cover, T., Thomas, J., et al. (1991).
Elements of information theory.
Wiley Online Library.

Feller, W. (2008).
An introduction to probability theory and its applications, volume 2.
John Wiley & Sons.

Li, M. and Vitányi, P. (2008).
An introduction to Kolmogorov complexity and its applications.
Springer-Verlag New York Inc.