Shanghai Jiao Tong University

概率论期末论文

MS107

Kolmogorov Complexity and Randomness

作者：卢禹杰

学院：致远学院计算机方向

年级：2018 级

学号：518030910111

2020 年 6 月 30 日

摘要

在这篇小论文中，我会首先我们将介绍 Kolmogorov Complexity[1](为了简洁，之后我们写作 K-complexity) 的定义和性质，然后我们会介绍以及它与其他复杂性的联系。

除此之外我们还会提及一些 K-complexity 与概率论中一些知识的联系，最后我们会简单介绍一些 K-complexity 的应用。

为了方便读者理解，正文部分我们会使用中文，而专业术语部分我们会使用英语。

目录

1 Introduction

1.1 K-complexity 定义

假设我们有一个 0,1 字符串 $s \in \{0,1\}^*$，这个字符串可能有许多描述（description），每种描述都只对应一个字符串。我们可以将这些描述中长度最小的那一个视为这个字符串复杂度的一个度量。

关于描述的严格定义，我们必须得十分严谨，否则会引起悖论。最著名的一个例子叫做 Richard–Berry paradox，考虑以下一个描述

> 最小的不能被 20 个字描述的自然数

如果这样的自然数存在，那我们刚刚就用 15 个字描述了它。如果不存在，那么任何自然数都可以被 20 个以下的字去描述，由抽屉原理这显然是不可能的。

一个比较自然的定义是

Definition 1.1 *一个字符串 s 的 K-complexity 是能够输出且仅输出这个字符串的最短计算机或图灵机程序的长度。*

我们要求一个描述对应一个字符串，也就是说存在一个函数 f 将描述 y 映射到唯一的一个字符串 s，我们希望 f 是可以被图灵机计算的，我们称这样的函数为partial recursive functions或者 computable functions。

那么有了 f 之后，我们就有了

Definition 1.2 *f 对应的 K-complexity*

$$C_f(s) = \begin{cases} \min\{|p| : f(p) = s\}, & s \in \mathrm{ran} f \\ \infty, & otherwise \end{cases}.$$

但是 K-complexity 应该只和字符串 s 本身有关。为了扔掉 f，考虑通用图灵机 U 使得 $\forall f, \exists$ program p

$$\forall y, s = U(p, y) = f(y).$$

这样我们就可以构造一个新的可计算函数 g 使得 $g\left(0^{|p|}1py\right) = U(p, y)$，这里 $0^{|p|}$ 的出现是为了把 U 的输入变成一维，最关键的是 g 有相对的优越性，即最小描述相对其他可计算函数更小

Claim 1 $\forall f, \exists c, C_g(s) \leq C_f(s) + 2|p| + 1 = C_f(s) + c$

从而我们可以有一个只和 s 有关的 K-complexity 定义 $C(s) = C_g(s)$。

除此之外我们还可以定义条件复杂度

Definition 1.3 (Conditional Complexity) $C_g(x \mid y) = \min\{|p| : g(p, y) = x\}$

这里的直觉是固定图灵机的输入，找到最短的描述（即图灵机）。

1.2 K-complexity 性质

K-complexity 有许多代数上的性质，这里简单举几例

- $C(x) \leq |x| + c$
- $C(xx) \leq C(x) + c$
- 对任意可计算函数 h, $C(h(x)) \leq C(x) + c_h$

证明并不困难，但我们最关心的一个性质是两个字符串 x, y 叠在一起时 (为了书写方便，我们写作 $\langle x, y\rangle$) 的复杂度会如何变化?

下面这个式子是否成立呢?

$$C(\langle x, y\rangle) \leq C(x) + C(y) + c.$$

实际上我们可以无限逼近这个结果。假设有两个程序 $p.q$ 分别生成 x, y，一个自然的想法是直接把两个程序拼在一起，但有一个问题是，我们不知道程序 p 在哪里结束，所以我们必须知道 p 或者 q 的长度，为了做到这一点我们只需在程序前面加上 $\log|p|$ 的信息，为了区分这个信息和原程序的信息，我们可以把相应的 bit 重复一遍。比如如果 p 长度为 `0b0101`，则我们在前面加上`0b0011001101` ，其中最后的 01 是终止符号。即新的程序长度为，$C(x) + C(y) + 2\log(C(x)) + c$，用相同的方法还可以优化这个界限

$$\begin{aligned} &C(\langle x, y\rangle) \leq C(x) + C(y) + 2\log(C(x)) + c \\ &C(\langle x, y\rangle) \leq C(x) + C(y) + \log(C(x)) + 2\log\log C(x) + c \\ &\qquad \cdots\cdots . \end{aligned}$$

1.3 $C(x)$ 的计算与性质

Definition 1.4 m 的 *length-conditional Kolmogorov complexity* (长度条件 *K-complexity*, 之后我们写作 *lc-K-complexity*) 是 $C(x \mid l(x))$.

以下例子摘自 [6] 的习题

Problem 1.1 π 的 *lc-K-complexity* 是一个常数，*i.e.* $C(\pi_{1:n} \mid n) \leq c$

Proof: 本例的直觉是给出一个有限的程序使得输入 n 可以输出 π 的前 n 位。显然这样的程序是存在的，可以用无穷级数 [2] 逼近

```
import numpy as np
simpi = lambda num: len(filter(lambda pt: pt[0]*pt[0]+pt[1]*pt[1]<1, np.
    reshape(np.random.uniform(size=2*num), [num, 2])))/float(num)*4
print simpi(1000000)

```

□

那么 $C(x)$ 到底该怎么计算呢？实际上它是无法用计算机计算的，也就是说

Theorem 1.1 $C(x)$ 不是一个 *partial computable function*。

计算准确的 $C(x)$ 不是不行，但是可以确定的是这是一个十分困难的问题 [5]。
但是值得高兴的是，$C(x)$ 可以被有效的逼近，详细逼近方法请参考 [6] 的 129 页。
经过计算机模拟，我们可以得到图 1 可以看到 $C(x)$ 在大体上是和 $\log x$ 比较接近的

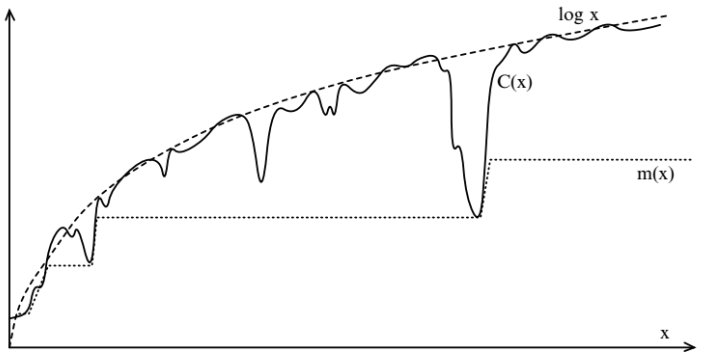

图 1: $C(x)$ 函数的增长趋势

实际上，我们还可以做出 $C(x \mid l(x))$ 的图像，

Problem 1.2 假设 $x \in \{1, \cdots, n\}$ ，每一个元素的概率都是 $\frac{1}{n}$，计算 $\mathbb{E}[C(x)]$.

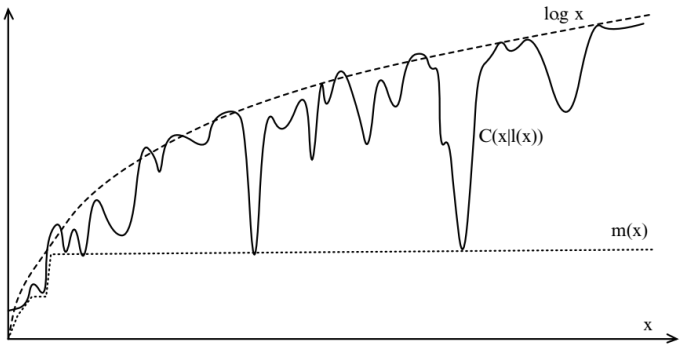

图 2: $C\left(x \mid l\left(x\right)\right)$ 函数的增长趋势

Proof: 熟知

$$\begin{aligned}\mathbb{E}\left[C\left(x\right)\right] &= \sum_{i=1}^{n} \frac{C\left(i\right)}{i} \\ &\geq \sum_{i=1}^{\log n} \frac{1}{2^i} \cdot \left(1 - \frac{i}{\log n}\right) \\ &= \log n + O\left(1\right).\end{aligned}$$

□

上面的例子告诉我们 $C\left(x\right)$ 确实和 $\log x$ 密不可分。

2 Randomness

2.1 定义

这一小节我们将介绍 K-complexity 所定义的随机性。

Definition 2.1 如果字符串 s 满足 $C\left(s\right) \geq |s|$, 则 s 是 *Kolmogorov* 随机的 *(Kolmogorov random).*

直觉告诉我们：随机的数一定有，并且还很多。而事实上我们的确可以证明这一点

Theorem 2.1 $\forall n$, $\exists s$ 使得 $|s| = n$, 且 s 是随机的。

Proof: 我们用反证法来证明。假设 $\exists n$ 使得 $\forall s, |s| = n$ 都有 $C(s) < n$。考虑到每个 s 都存在一个对应的程序 p_s 使得 $g(p_s) = U(p_s, s) = s$ ，且 $|p_s| < n$。并且由定义知对 $s_1 \neq s_2$ ，对应程序 $p_{s_1} \neq p_{s_2}$ ，注意到长度小于 n 的程序最多 $2^n - 1$ 个。而字符串 s 是无限多的，由抽屉原理我们立即得到矛盾。 □

注意到我们证明的事实实际上是比较弱的，所以我们可能可以将上述定理稍加推广，并且还是能用同样的方法证明！

2.2 不可压缩性

首先我们对 random 的定义进行推广：

Definition 2.2 (c-random) 如果存在常数 c 使得字符串 s 满足 $C(s) \geq l(s) - c = |s| - c$。我们称 s 是 *c-random* 的。我们称 s 是 *incompressible* 当且仅当存在 c 使得 s 是 *c-random* 的。

注： 在某些书上，也称 s 为 c-incompressible，random 和 incompressible 的本质是相同的，这从直觉上也是容易理解的。

Theorem 2.2 (Incompressibility Theorem) 给定 y 以及正常数 c，每一个大小为 m 的集合 A 都有至少 $m(1 - 2^{-c}) + 1$ 个元素 x 使得 $C(x \mid y) \geq \log m - c$.

Proof: 同样的，使得 $C(x \mid y) \geq \log m - c$ 长度少于 $\log m - c$ 的程序个数为

$$\sum_{i=1}^{\log m - c - 1} 2^i = 2^{\log m - c} - 1 = m2^{-c} - 1.$$

由抽屉原理我们得到集合 A 中至少 $m - (m2^{-c} - 1)$ 个元素使得 $C(x \mid y) \geq \log m - c$. □

还有一个有趣的事实是

Claim 2 如果 p 是 s 的最小程序即 $C(s) = l(p)$ ，那么 p 一定是 *incompressible* 的。

Proof: 还是用反证法，假设 $\forall c$ 都 $\exists s$ 以及一个最小程序 q 可以生成 p_s 使得

$$l(q) \leq l(p) - c.$$

大概的思路是用 q 来构造一个更小的可以生成 s 的程序，从而导出矛盾。 □

incompressible 还可以继续被推广，这次我们考虑将常数 c 推广到更加一般的函数

Definition 2.3 假设 $g:\mathbb{N}\to\mathbb{N}$ 是无界的，如果长度为 n 的 s 满足 $C(s)\geq n-g(n)$，那么我们称 s 是 *g-incompressible* 的。

一个很自然的问题是，这样的字符串是否还是有非常多？事实上的缺如此，当 n 趋于无穷的时候，g-incompressible 的字符串出现的概率会无限接近于 1:

Theorem 2.3 假设 $I(n)$ 是长度不超过 n 的 *g-incompressible* 字符串的个数，则

$$\lim_{n\to\infty}\frac{I(n)}{2^{n+1}}=1.$$

Proof: 还是用相同的思路。注意到

$$I(n)=\sum_{i=1}^{n}\sum_{\substack{l(s)=i\\ c(s)\geq i-g(i)}}1.$$

我们考虑长度小于 $i-g(i)$ 的程序的总数，这是不超过 $2^{i-g(i)}-1$ 的，也就是说还剩至少 $2^i-(2^{i-g(i)}-1)$ 个字符串满足 $c(s)\geq i-g(i)$ ，所以第二个求和的值至少为:

$$2^i-2^{i-g(i)}+1.$$

进而

$$I(n)\geq 2^{n+1}-2-\sum_{i=1}^{n}2^{i-g(i)}+n.$$

由于 g 是 unbounded 的，$\exists N,\forall n>N,g(i)\geq 2$，命题得证。 □

2.3 Random s 的最长全 0 子列

假设长度为 n 的字符串 s 是随机的，一个有趣的问题是 s 中最长的一段全为 0 的子列能有多长呢，显然这个子列不能是 $O(n)$ 级别长度的，否则这个字符串可以被轻松的描述。我们可以证明这个长度是 $\log n$ 级别的。

Theorem 2.4 长度 n 的字符串 s 是随机的，那么最长的全 *0* 子列长度不超过 $\log n$.

Proof: 假设 s 可以被写作 $u0^{2\log n}v$，那么我们可以用 u, v 来描述 s，由之前的定理知

$$\begin{aligned} C(s) &\leq l(u) + l(v) + \log l(u) + 2\log\log l(u) + c \\ &\leq n - \log n + 2\log\log n + c. \end{aligned}$$

这与我们之前的假设矛盾! □

类似的，我们还可以证明随机的 s 必定有相对较长的全 0 子列。

3 Algorithmic Information Theory

3.1 Algorithmic Information

Definition 3.1 *y 包含 x 的算法信息 (algorithmic information) 为*

$$I_C(x:y) = C(y) - C(y \mid x)$$

容易发现 $C(x) = I_C(x:x)$，即 $C(x)$ 可以看做 x 包含自己的 algorithmic information。

3.2 Symmetry of information

下面我们介绍一个 K-Complexity 极为重要的定理，假设 $n = \max\{l(x), l(y)\}$. ，首先显然我们有

$$C(\langle x, y\rangle) \leq C(y \mid x) + C(x) + O(\log n).$$

因为想生成 $\langle x, y\rangle$ 我们可以先用一个程序生成 x，再用一个输入 x 输出 y 的程序来输出 y.

惊人的是，这个不等式在反方向也是成立的

Theorem 3.1 *$C(y \mid x) + C(x) \leq O(\log n) + C(\langle x, y\rangle)$ ，其中 $n = \max\{l(x), l(y)\}$.*

Proof: 考虑集合 $A = \{\langle u, v\rangle \mid C(\langle u, v\rangle) \leq C(\langle x, y\rangle)\}$ 以及

$$A_u = \{u \mid \langle u, v\rangle \in A\}.$$

显然 A, A_u 都是有限集，假设 $e = \log |A_x|$，从而

$$C(y \mid x) \leq \log |A_x| + O(1) = e + O(1).$$

考虑 $B = \{u \mid |A_u| \geq 2^e\}$, 则我们有

$$|B| \leq \frac{|A|}{2^e} \leq \frac{2C(\langle x, y\rangle)}{2^e}.$$

所以

$$\begin{aligned} C(x) &\leq e + 2\log e + \log \frac{2^{C(\langle x,y\rangle)}}{2^e} \\ &\leq C(\langle x, y\rangle) - e + O(\log n). \end{aligned}$$

整理后得到

$$C(y \mid x) + C(x) \leq O(\log n) + C(\langle x, y\rangle).$$

□

根据这个定理我们可以得到在 $O(\log n)$ 的误差下，我们有

$$\begin{aligned} &C(x) - C(x \mid y) = C(y) - C(y \mid x) \\ \Rightarrow &I_C(x : y) = I_C(y : x) \pm O(\log n). \end{aligned}$$

4 Applications

4.1 The Incompressiblity Method

Incompressiblity Method 是一种证明技巧，他可以被用来证明许多有趣的事情，其大概的思路是想要证明存在某个实例满足某种性质，一般这种实例很难被构造出来，但是我们可以用 random 的物体来代替这种实例，[6] 有专门的一章 (Chapter 6) 讲述这个方法的使用，并且有大量例子，比如作者用这个方法证明了 Hastad's switching lemma，有兴趣的读者可以自行阅读。

5 Acknowledgement and Conclusion

讲这个 Kolmogorov Complexity 主要还是因为看到吴老师在微信群发的一个 pdf:What is Kolmogorov Complexity，我看了之后十分感兴趣。然后为了写这篇小论文，我主要阅读了 Kolmogorov Complexity 的一部经典著作 [6] 的第一，二，四章，谈了谈自己的理解，并且解决了其中一些练习题。

经过了解粗浅的了解后，我发现 Kolmogorov Complexity 可以用来解决很多有趣的问题，它与理论计算机和概率论都有着十分紧密的联系 [3]，尽管它很早就被提出和研究 [4] 了，但是就在最近的理论计算机顶会 STOC 2020 也有相关的论文发表 [5]，主要研究计算 K-complexity，也就是 $C(x)$ 的困难性。

最后，感谢吴耀琨老师和祝隐峰学长的辛苦付出，感谢同学们的无私分享。

参考文献

[1] Komogorov complexity. https://en.wikipedia.org/wiki/Kolmogorov_complexity. Accessed June 28, 2020.

[2] 用计算机算圆周率，是个怎样的过程. https://www.zhihu.com/question/20756479. Accessed June 28, 2020.

[3] Lance Fortnow. Kolmogorov complexity and computational complexity. *Complexity of Computations and Proofs. Quaderni di Matematica*, 13, 2004.

[4] Juris Hartmanis. Generalized kolmogorov complexity and the structure of feasible computations. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 439–445. IEEE, 1983.

[5] Shuichi Hirahara. Unexpected hardness results for kolmogorov complexity under uniform reductions. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1038–1051, 2020.

[6] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.