Codificación aritmética

Fernando Martínez
fernando.martinez@upc.edu

Departament de Matemàtiques • Universitat Politècnica de Catalunya

22 de febrero de 2022

Capítulo 4 de Introduction to Data Compression **Sayood, Khalid** Morgan Kaufmann, 2012, 4th ed.

Algoritmo

Ejemplo

Consideremos la fuente $\{(a, 0{,}4), (b, 0{,}3), (c, 0{,}2), (d, 0{,}1)\}$ y queremos codificar `ccda`.

Si quisiéramos usar Huffman lo mejor sería una 4-extensión de la fuente, calcular las palabras del código asociado y elegir la correspondiente a `ccda`. En este caso sería necesario un árbol con $4^4 = 256$ hojas.[a]

La forma ideal sería poder hallar la palabra del código sin necesidad de construir todo el árbol.

Podemos pensar que cada recorrido desde la raiz a las hojas es la representación binaria de un número en el intervalo $[0, 1)$.[b]

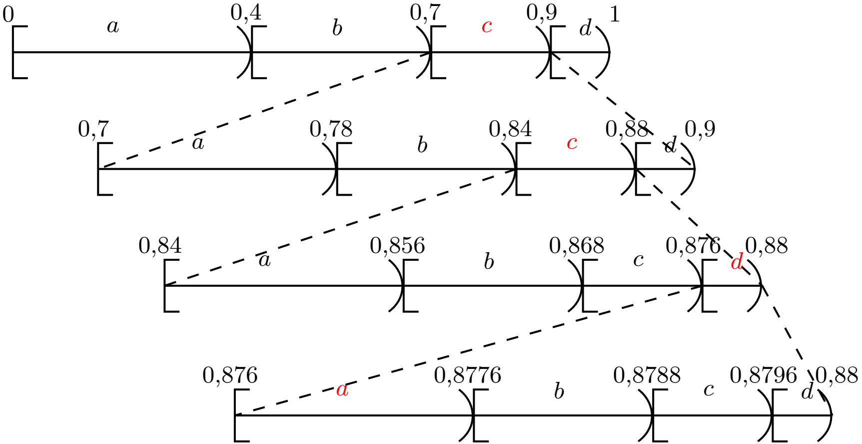
Esta es la idea de la codificación aritmética, codificar el mensaje con un número del intervalo $[0, 1)$.

[a] En general s^n hojas con s # de símbolos y n la longitud del mensaje, ¡impracticable!

[b] Por ejemplo, $0100110 = 0\frac{1}{2^1} + 1\frac{1}{2^2} + 0\frac{1}{2^3} + 0\frac{1}{2^4} + 1\frac{1}{2^5} + 1\frac{1}{2^6} + 0\frac{1}{2^7} = \frac{19}{64}$.

Algoritmo

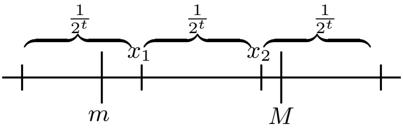
Ejemplo $((a, 0{,}4), (b, 0{,}3), (c, 0{,}2), (d, 0{,}1))$



El mensaje estará representado por un real del intervalo $[0{,}876, 0{,}8776) \equiv [m, M)$.

Cualquier valor del intervalo sirve para representar el mensaje.

Método 1 Hallar el menor entero t tal que $\frac{1}{2^t} \leq M - m$. A continuación hallar x tal que $m \leq \frac{x}{2^t} < M$ ó equivalentemente $m2^t \leq x < M2^t$; si hay dos valores posibles, se elige el valor par.



Ejemplo anterior $[0{,}876, 0{,}8776)$

$\frac{1}{2^t} \leq 0{,}8776 - 0{,}876 = 0{,}0016$, $t = 10$, $2^t = 1024$. $897{,}024 \leq x < 898{,}662$. El valor sería

$$\begin{aligned}\frac{898}{1024} =& \frac{449}{512} = \frac{256 + 128 + 64 + 0 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1}{512} \\ =& \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{0}{16} + \frac{0}{32} + \frac{0}{64} + \frac{0}{128} + \frac{0}{256} + \frac{1}{512} = (0{,}111000001)_2\end{aligned}$$

El código sería 111000001 (449 en binario con 9 bits $512 = 2^9$).

Si hubiéramos tenido $x = 7$, el código sería 0000000111 (10 bits $1024 = 2^{10}$).

Método 2 Hallar la expresión binaria de m y M:

$$m = 0.a_1a_2...a_r0...$$
$$M = 0.a_1a_2...a_r1...$$
$$\uparrow \text{ primer bit en que difieren}$$

En la mayoría de los casos se enviará $a_1a_2...a_r1$.
Excepciones:

1. si $m = 0.a_1a_2...a_r$ se envía $a_1a_2...a_r$
2. si $M = 0.a_1a_2...a_r1$:
   - si la expresión de m es finita se envía m,
   - si no lo es, se envía $a_1a_2...a_r0a_{r+2}...a_s1$ siendo $a_{r+2} = ... = a_s = 1,\ a_{s+1} = 0$ en la representación de m; o sea se envían se envían los 1's que vienen y el primer 0 se transforma en 1.*

Ejemplo

$$m = 0{,}10101|0|11001...$$
$$M = 0{,}10101|1| \quad \updownarrow$$
$$\text{se envía} \quad 10101\ 0\ 11\ 1$$

*No pueden ser todos 1 porque entonces $m = M$ $(0.\bar{9} = 1)$.

Volviendo al ejemplo anterior $[0{,}876, 0{,}8776)$

$$m = 0{,}876 = 0{,}11100000|0|1000$$
$$M = 0{,}8776 = 0{,}11100000|1|0101$$

El código que se envía es 111000001, 9 bits. (Notemos que $\log \frac{1}{p} = \log \frac{1}{0{,}2\,0{,}2\,0{,}1\,0{,}4} \approx 9{,}3$ bits.)

Observaciones:

- Con el primer método hay que esperar a tener todo el mensaje para empezar a codificarlo.
- Con el segundo método se puede empezar a codificar en cuanto haya bits que coincidan en la representación binaria de m y M.

Algoritmo

- *Prerrequisito:* Dados $s_1, ..., s_n$ con probabilidades $p_1 \geq p_2 \geq ... \geq p_n > 0$ definimos la función de distribución

$$F(i) = \sum_{k=1}^{i} p_k$$

En el caso del ejemplo 2 si $a \leftrightarrow 1$, $b \leftrightarrow 2$, $c \leftrightarrow 3$, $4 \leftrightarrow 4$, entonces:

$F(0) = 0, \quad F(1) = 0{,}4, \quad F(2) = 0{,}7, \quad F(3) = 0{,}9, \quad F(4) = 1.$

- *Codificar:* Dada la secuencia $s_{i_1}s_{i_2}s_{i_3}...s_{i_r}$

$$m^{(0)} = 0 \qquad M^{(0)} = 1$$

Para $k = 1, ..., r$

$$d_{k-1} = M^{(k-1)} - m^{(k-1)}$$
$$m^{(k)} = m^{(k-1)} + F(s_{i_k} - 1)d_{k-1}$$
$$M^{(k)} = m^{(k-1)} + F(s_{i_k})d_{k-1}$$

- *Descodificar:* Dado x_1 y la longitud del mensaje.

  Para $i = 1, ...,$ longitud del mensaje:

  - Si $F(k-1) \leq x_i < F(k)$ devolver el símbolo s_k.
  - $x_{i+1} = \dfrac{x_i - F(k-1)}{F(k) - F(k-1)}$.

Ejemplo: Si hemos recibido 111000001

- $x_1 = 0{,}876953125 \longmapsto c$
- $x_2 = 0{,}884765625 \longmapsto c$
- $x_3 = 0{,}923828125 \longmapsto d$
- $x_4 = 0{,}23828125 \longmapsto a$
- $x_5 = 0{,}595703125 \longmapsto b$
- $x_6 = 0{,}65234375 \longmapsto b$
- ...

Podríamos seguir indefinidamente, por eso es necesario dar la longitud del mensaje o incluir un símbolo `EOF`.

Observaciones:

1. El número que se devuelve al codificar se puede construir cómo se quiera, lo importante es que esté confinado en el intervalo semiabierto.
2. Es necesario dar la longitud del mensaje o incluir un símbolo `EOF`.

Ver `03.1_Aritmetica.py`

Teorema

La longitud media de los códigos aritméticos generados para mensajes de longitud n es menor que $n\,H(\mathcal{S})+1$.

Demostración.

Igual que el teorema de la codificación sin ruido (y de la extensión de la fuente), teniendo en cuenta que existe $\frac{1}{2^t} \leq M - m < \frac{1}{2^{t-1}}$, por lo tanto necesitamos como máximo t bits,

$t < \log \dfrac{1}{p_{i_1} p_{i_2} \cdots p_{i_n}} + 1$ $(*)$.

$$
\begin{aligned}
\bar{l} = & \sum_{i_1} \cdots \sum_{i_n} p_{i_1} \cdots p_{i_n} l_{i_1 \ldots i_n} \overset{(*)}{<} \sum_{i_1} \cdots \sum_{i_n} p_{i_1} \cdots p_{i_n} \left[\log \frac{1}{p_{i_1} p_{i_2} \cdots p_{i_n}} + 1 \right] \\
= & \sum_{i_1} \cdots \sum_{i_n} p_{i_1} \cdots p_{i_n} \left(\sum_{k=1}^{n} \log \frac{1}{p_{i_k}} \right) + \sum_{i_1} \cdots \sum_{i_n} p_{i_1} \cdots p_{i_n} \\
= & \sum_{i_1} \cdots \sum_{i_n} p_{i_1} \cdots p_{i_n} \log \frac{1}{p_{i_1}} + \cdots + \sum_{i_1} \cdots \sum_{i_n} p_{i_1} \cdots p_{i_n} \log \frac{1}{p_{i_n}} + 1 \\
= & nH(\mathcal{S}) + 1
\end{aligned}
$$

□

Detalles de la implementación

¡Necesitamos precisión infinita!

Posibles soluciones, no excluyentes:

- Reescalado.
- Trabajar con enteros en el intervalo $[0, R)$ en vez de reales de $[0, 1)$. En este caso en vez de probabilidades trabajamos con frecuencias $f_i \in \mathbb{N}$.

Reescalado (I)

- $E_1(x) : [0, \frac{1}{2}) \to [0, 1)$, $E_1(x) = 2x$, sale: 0.
- $E_2(x) : [\frac{1}{2}, 1) \to [0, 1)$, $E_2(x) = 2x - 1$, sale: 1.
- $E_3(x) : [\frac{1}{4}, \frac{3}{4}) \to [0, 1)$, $E_3(x) = 2x - \frac{1}{2}$, sale: hay que esperar $(*)$.

$E_1(x)$, $E_2(x)$ equivalente a $\ll$. $E_3(x)$ equivalente a $\ll$ y $\oplus\,$0x80$\ldots$0.

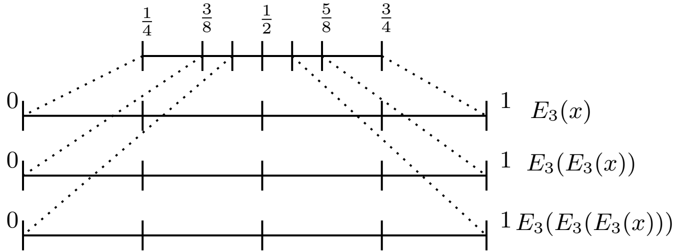
$(*)$ Si hay que esperar N veces y después se cae en:

- E_1 sale $0\,\overbrace{1\ldots1}^{N}$,
- E_2 sale $1\,\overbrace{0\ldots0}^{N}$.

Reescalado (II)

(∗) Si hay que esperar N veces y después se cae en:

- E_1 sale $0\,\overbrace{1\ldots1}^{N}$, • E_2 sale $1\,\overbrace{0\ldots0}^{N}$.



$E_3(x)$ $\begin{cases}\text{si llegamos a } [0,\ \frac{1}{2}) \text{ veníamos de } [\frac{1}{4},\ \frac{1}{2})\ [0.01\ldots,\ 0.01\ldots) \\ \text{si llegamos a } [\frac{1}{2},1) \text{ veníamos de } [\frac{1}{2},\ \frac{3}{4})\ [0.10\ldots,\ 0.10\ldots)\end{cases}$

$E_3(E_3(x))$ $\begin{cases}\text{si llegamos a } [0,\ \frac{1}{2}) \text{ veníamos de } [\frac{3}{8},\ \frac{1}{2})\ [0.011\ldots,\ 0.011\ldots) \\ \text{si llegamos a } [\frac{1}{2},1) \text{ veníamos de } [\frac{1}{2},\ \frac{5}{8})\ [0.100\ldots,\ 0.100\ldots)\end{cases}$

$E_3(E_3(E_3(x)))$ $\begin{cases}\text{si llegamos a } [0,\ \frac{1}{2}) \text{ veníamos de } [\frac{7}{16},\ \frac{1}{2})\ [0.0111\ldots,0.0111\ldots) \\ \text{si llegamos a } [\frac{1}{2},1) \text{ veníamos de } [\frac{1}{2},\ \frac{9}{16})\ [0.1000\ldots,0.1000\ldots)\end{cases}$

Trabajar con enteros $[0, R)$

Si los símblos aparecen con frecuencia $f_i \in \mathbb{N}$, sea $T = \sum_i f_i$.

Definimos R tal que $R > 4T$ (porque el intervalo $[n, m)$ puede llegar a ser como máximo $\frac{1}{4}$ del intervalo total antes de ser reescalado; en este cuarto del intervalo ha de caber T para poder discernir entre los distintos símbolos).

Ver `03_Aritmetica_entera.py` para ilustrar el algoritmo.