Arithmetic Codes

Visualization using "language model" of controller presses (X,Y,A,B)

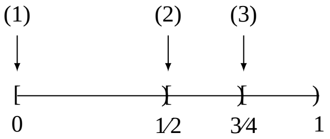
In this section, we study a different kind of code that can handle context-dependent distributions, unlike the Huffman code. Binary representations of numbers in the interval $[0, 1)$ as studied in the previous exercise give rise to a very elegant code:

Definition: Arithmetic code

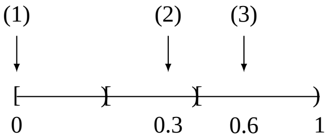
Given a source P_X with $\mathcal{X} = \{x_1, .., x_m\}$, construct the arithmetic code as follows. Divide the interval $[0, 1)$ into disjoint subintervals $I_{x_j} = [a_j, a_{j+1})$, where $a_1, \dots, a_{m+1}$ are defined such that $a_{j+1} - a_j = P_X(x_j)$, and $a_1 = 0, a_{m+1} = 1$. The encoding $AC(x_j)$ of the element x_j is the (shortest possible) standard binary representation of some number in the interval I_{x_j}.

Example

Let X be a random variable with $\mathcal{X} = \{1, 2, 3\}$ and $P_X(1) = \frac{1}{2}, P_X(2) = P_X(3) = \frac{1}{4}$. The arithmetic code is constructed by first determining the intervals:



This image results in the arithmetic code $\mathcal{C}$ with $\mathcal{C}(1) = 0$ (the representation of 0), $\mathcal{C}(2) = 1$ (the representation of $\frac{1}{2}$), $\mathcal{C}(2) = 11$ (the representation of $\frac{3}{4}$). For P_X, the codewords happen to fall exactly *on* the boundaries of the intervals. This is not always the case, however. The same code would have resulted from this procedure if we started with the random variable Y with $\mathcal{Y} = \{1, 2, 3\}$ and $P_Y(1) = P_Y(2) = 0.3$ and $P_Y(3) = 0.4$:



Proposition

For any (X, P_X), the arithmetic code has average length $\ell_{AC}(P_X) \leq H(X) + 1$.

Proof

Let $x \in \mathcal{X}$, and define $\ell_x := \lceil \log(1/P_X(x)) \rceil$ to be the rounded surprisal value of x. Then

$$2^{-\ell_x} = 2^{-\lceil \log(1/P_X(x)) \rceil} \leq 2^{-\log(1/P_X(x))} = 2^{\log P_X(x)} = P_X(x).$$

Therefore, since the size of the interval I_x is $P_X(x)$, there must exist an integer $0 \leq s_x < 2^{\ell_x}$ such that $s_x \cdot 2^{-\ell_x}$ lies in the interval I_x. This number $s_x \cdot 2^{-\ell_x}$ has a binary representation of length $\ell_x \leq -\log P_X(x) + 1$. Repeating this argument for every x, we obtain

$$\ell_{AC}(P_X) = \mathbb{E}[\ell(AC(X))] = \sum_x P_X(x)\ell(AC(x)) \leq \sum_x P_X(x)(-\log P_X(x) + 1) = H(X) + 1.$$

As we can see from the example above, this construction for arithmetic codes does not necessarily yield prefix-free codes. However, at the expense of one extra bit of code (on average), the construction can be adapted into a prefix-free code. One example of this is the **Shannon-Fano-Elias code** (see

Cover/Thomas, Section 5.9 or Wikipedia), which provides a more sophisticated way of selecting a number within each interval than simply selecting the number with the shortest binary representation. This alternative selection procedure ensures prefix-freeness. Another option is to select *binary intervals* within each interval:

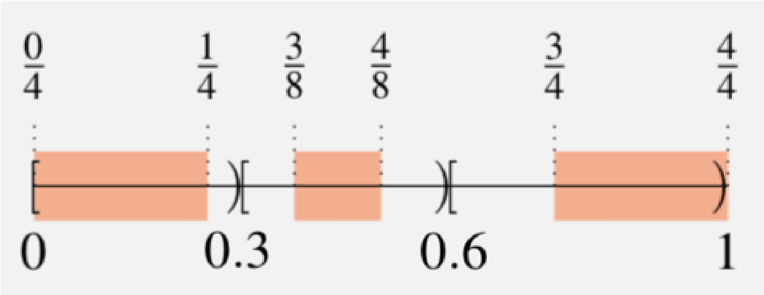
Definition: Arithmetic code (prefix-free version)

A prefix-free arithmetic code is identical to the definition above, except that the encoding $AC^{pf}(x_j)$ of the element x_j is now the name of a largest binary interval that fits entirely in I_{x_j}.

Similarly to the proposition above, it can be shown that for any source P_X, $\ell_{AC^{pf}}(P_X) \leq H(X) + 2$. Note that we get prefix-freeness only at the expense of an extra bit on average.

Example

Let Y be the random variable as in the example above, that is, $P_Y(1) = P_Y(2) = 0.3$ and $P_Y(3) = 0.4$. The prefix-free code for Y is constructed as follows:



This results in the codewords $\mathcal{C}(1) = 00$, $\mathcal{C}(2) = 011$, and $\mathcal{C}(3) = 11$.

The arithmetic code is slightly less efficient than the Huffman code in terms of average codeword length. A big advantage is the way it is able to adapt to changing distributions, such as when we are encoding a stream of English text. Suppose we are given the (not necessarily i.i.d.) random variables $X_1, X_2, \ldots, X_n$, and we want to encode the source $P_{X_1 X_2 \cdots X_n}$. We start by dividing the interval $[0,1)$ into subintervals according to P_{X_1}. If, for example, the event $X_1 = b$ happens, we zoom into the interval corresponding to b, and subdivide *that* interval according to $P_{X_2|X_1}$, so that the sizes of these intervals add up to $P_{X_1}(b)$.

The concept of arithmetic coding is exploited as an accessibility tool in the keyboard alternative Dasher, invented by the group of David MacKay at Cambridge University, UK. Try to download and play with it, it's great fun!