Symbol Probabilities for Arithmetic Coding

Symbol probabilities are often derived from *counts* of how often symbols occurred previously. We'll design an arithmetic coder assuming this.

Suppose the counts for symbols $s_1, \ldots, s_q$ are $f_1, \ldots, f_q$ times (with all $f_i > 0$). Then we'll use estimated probabilities of

$$p_i \;=\; f_i \,/ \sum_{j=1}^{q} f_j$$

For arithmetic coding, it's convenient to pre-compute the *cumulative frequencies*

$$F_i \;=\; \sum_{j=1}^{i} f_j$$

We define $F_0 = 0$, and use T for the total count, F_q. We will assume that $T < 2^h$, so counts fit in h bits.

Precision of the Coding Interval

The ends of the coding interval will be represented by m-bit integers.

The integer bounds l and u represent the interval

$$\big[\,l \times 2^{-m},\;\; (u{+}1) \times 2^{-m}\big)$$

(The addition of 1 to u allows the upper bound to be 1 without the need to use $m+1$ bits for u.)

The received message, t, will also be represented as an m-bit integer.

With these representations, the arithmetic performed will never produce a result bigger than $m+h$ bits.

Encoding Using Integer Arithmetic

```
l ← 0,  u ← 2^m − 1
c ← 0

For each source symbol, s_i, in turn:

  r ← u − l + 1
  u ← l + ⌊(r ∗ F_i) / T⌋ − 1
  l ← l + ⌊(r ∗ F_{i−1}) / T⌋

  While l ≥ 2^m/2 or u < 2^m/2 or l ≥ 2^m/4 and u < 2^m ∗ 3/4:
    If l ≥ 2^m/2:
      Transmit a 1 bit followed by c 0 bits
      c ← 0
      l ← 2 ∗ (l − 2^m/2),  u ← 2 ∗ (u − 2^m/2) + 1
    If u < 2^m/2:
      Transmit a 0 bit followed by c 1 bits
      c ← 0
      l ← 2 ∗ l,  u ← 2 ∗ u + 1
    If l ≥ 2^m/4 and u < 2^m ∗ 3/4:
      c ← c + 1
      l ← 2 ∗ (l − 2^m/4),  u ← 2 ∗ (u − 2^m/4) + 1

Transmit two final bits to specify a point in the interval
If l < 2^m/4:
  Transmit a 0 bit followed by c 1 bits
  Transmit a 1 bit
Else
  Transmit a 1 bit followed by c 0 bits
  Transmit a 0 bit
```

Precision Required

For this procedure to work properly, the loop that expands the interval must terminate. This requires that the interval never shrink to nothing — ie, we must always have $u \geq l$.

This will be guaranteed as long as

$$\lfloor (r * F_i)\,/\,T \rfloor \;>\; \lfloor (r * F_{i-1})\,/\,T \rfloor$$

This will be so as long as $f_i \geq 1$ (and hence $F_i \geq F_{i-1} + 1$) and $r \geq T$.

The expansion of the interval guarantees that $r \geq 2^m/4 + 1$.

So the procedure will work as long as $T \leq 2^m/4 + 1$. If our symbol counts are bigger than this, we have to scale them down (or use more precise arithmetic, with a bigger m).

However, to obtain near-optimal coding, T should be a fair amount less than $2^m/4 + 1$.

Decoding Using Integer Arithmetic

$l \leftarrow 0,\ u \leftarrow 2^m - 1$
$t \leftarrow$ first m bits of the received message

Until last symbol decoded:

$r \leftarrow u - l + 1$
$v \leftarrow \lfloor ((t - l + 1) * T - 1) / r \rfloor$

Find i such that $F_{i-1} \leq v < F_i$
Output s_i as the next decoded symbol

$u \leftarrow l + \lfloor (r * F_i) / T \rfloor - 1$
$l \leftarrow l + \lfloor (r * F_{i-1}) / T \rfloor$

While $l \geq 2^m/2$ or $u < 2^m/2$ or $l \geq 2^m/4$ and $u < 2^m * 3/4$:

If $l \geq 2^m/2$:
$l \leftarrow 2 * (l - 2^m/2),\ u \leftarrow 2 * (u - 2^m/2) + 1$
$t \leftarrow 2 * (t - 2^m/2)\ +$ next message bit

If $u < 2^m/2$:
$l \leftarrow 2 * l,\ u \leftarrow 2 * u + 1$
$t \leftarrow 2 * t\ +$ next message bit

If $l \geq 2^m/4$ and $u < 2^m * 3/4$:
$l \leftarrow 2 * (l - 2^m/4),\ u \leftarrow 2 * (u - 2^m/4) + 1$
$t \leftarrow 2 * (t - 2^m/4)\ +$ next message bit

Proving That the Decoder Finds the Right Symbol

To show this, we need to show that if

$$F_{i-1} \leq \lfloor ((t - l + 1) * T - 1) / r \rfloor < F_i$$

then

$$l + \lfloor (r * F_{i-1}) / T \rfloor \leq t \leq l + \lfloor (r * F_i) / T \rfloor - 1$$

This can be proved as follows:

$F_{i-1} \leq \lfloor ((t-l+1) * T - 1) / r \rfloor \leq ((t-l+1) * T - 1) / r$
$\Rightarrow r * F_{i-1} / T \leq t - l + 1 - 1/T$
$\Rightarrow l + \lfloor (r * F_{i-1}) / T \rfloor \leq l + (t - l) = t$

$F_i > \lfloor ((t - l + 1) * T - 1) / r \rfloor$
$\Rightarrow F_i \geq \lfloor ((t - l + 1) * T - 1) / r \rfloor + 1$
$\Rightarrow F_i \geq ((t - l + 1) * T - 1) / r - (r-1)/r + 1$
$\Rightarrow r * F_i / T \geq t - l + 1 - 1/T - (r-1)/T + r/T$
$\Rightarrow r * F_i / T \geq t - l + 1$
$\Rightarrow l + \lfloor (r * F_i) / T \rfloor - 1 \geq t$

Summary

- Arithmetic coding provides a practical way of encoding a source in a very nearly optimal way.
- Faster arithmetic coding methods that avoid multiplies and divides have been devised.
- **However:** It's not necessarily the best solution to *every* problem. Sometimes Huffman coding is faster and almost as good. Other codes may also be useful (see Sayood, Sections 3.5 to 3.7).
- Arithmetic coding is particularly useful for *adaptive* codes, in which probabilities constantly change. We just update the table of cumulative frequencies as we go.

History of Arithmetic Coding

- Elias — around 1960.
  Seen as a mathematical curiosity.
- Pasco, Rissanen – 1976.
  The beginnings of practicality.
- Rissanen, Langdon, Rubin, Jones – 1979.
  Fully practical methods.
- Langdon, Witten/Neal/Cleary — 1980's.
  Popularization.
- Many more... (eg, Moffat/Neal/Witten)
  Further refinements to the method.