Arithmetic coding

Information Theory (APMA 1710), Fall 2011

(1) Implement the arithmetic coding algorithm

Implement the finite-precision version of the algorithm, using code alphabet $\mathcal{A} = \{0, 1\}$. Your implementation should include:

(a) (Encoder) A function that takes a (finite) source alphabet $\mathcal{X} = \{EOF, x_1, \ldots, x_n\}$, a vector of probabilities $p = (p_0, \ldots, p_n)$, and a string to be encoded, and outputs the encoded sequence.

(b) (Decoder) A function that takes an encoded sequence produced by (a) (along with $\mathcal{X}$ and p) and decodes it back into the original sequence of source symbols.

Notes: For the rest of the homework assignment, you will only need the Encoder. So, if you have difficulty implementing the Decoder you may want to come back to this after finishing the rest of the exercises.

Use whatever level of precision you find convenient. This determines the range of integers you will use. (A typical choice is 16-bit precision, but you will need more than this for the Extra Credit part below. I'm using 32-bit precision.)

Although you are free to implement the algorithm any way you like, you may find it useful to first implement the "infinite-precision" version using floating-point arithmetic (without rescaling) and test it on short input strings, and only then modify your implementation to use integers and rescaling to be able to handle arbitrary length strings. Remember: The key to making the Decoder work is to do the rescaling at exactly the same steps during decoding as during encoding.

(2) Verify your implementation

Run your implementation on the following short test cases to make sure everything is working. For each test case, you are given $\mathcal{X}$, p, the input string x, and the correct encoded string β. The EOF symbol is 0.

(1) $\mathcal{X} = (0, 1, 2)$, $p = (.2, .4, .4)$, $x = 210$, $\beta = 1011$
(2) $\mathcal{X} = (0, 1, 2)$, $p = (.2, .4, .4)$, $x = 1221120$, $\beta = 100011111$
(3) $\mathcal{X} = (0, 1, 2, 3)$, $p = (.05, .05, .5, .4)$, $x = 2320$, $\beta = 011011$
(4) $\mathcal{X} = (0, 1, 2, 3)$, $p = (.05, .05, .5, .4)$, $x = 3231220$, $\beta = 1100001111$

(3) Compare lengths

In this exercise, we compare arithmetic coding with Huffman coding and with the entropy. In the previous homework assignment, you implemented a function to take a (finite) source alphabet $\{1, \ldots, d\}$ and a PMF $q = (q_1, \ldots, q_d)$, and generate a sequence of n i.i.d. samples $X_1, \ldots, X_n$ from q. To do arithmetic coding, we need to have an EOF symbol, so we (A) augment the original source alphabet to contain 0, so that $\mathcal{X} = \{0, 1, \ldots, d\}$, and (B) modify q to obtain p by choosing some value of p_0 (the probability of EOF) and setting

$$p_i = (1 - p_0)q_i$$

for $i = 1, \ldots, d$. Consider the following values of q:

(1) $q = (.5, .25, .125, .125)$
(2) $q = (.99, .01)$
(3) $q = (.001, .002, .003, .004, .99)$

For each of these values of q:

(1) Generate a sequence of length $n = 1000$,
(2) Encode the sequence using arithmetic coding with p obtained from q as above, choosing $p_0 = 1/n$,
(3) Compare the following quantities:
   (a) the encoded length using arithmetic coding,
   (b) the encoded length using Huffman coding,
   (c) the "ideal" encoded length: $nH_2(q)$,
   (d) the "original" length, converted to bits: $n \log_2 d$,

   where d is the number of elements in the original source alphabet $\{1, \ldots, d\}$, before including the EOF symbol 0.

(EXTRA CREDIT)

Get within a few bits of the entropy on a real-world data set

Included along with these instructions is the same text file `calgary.txt` that was included in the Huffman assignment.

Estimate the probability of each symbol that appears in the file, by counting how many times it occurs. Since the character 0 may occur in the file, we can't use it as the EOF symbol. Instead, you can use a non-printable character that doesn't occur in the file, such as the ASCII character with code 00, sometimes denoted NUL. Using your estimated probabilities q, compute the corresponding p as in exercise (3), using $p_{EOF} = 1/n$ where n is the number of characters in the file.

Encode the file using your arithmetic coding implementation. How does your encoded length compare with the "ideal" length, $nH_2(q)$? How does this compare with the encoded length using Huffman coding?

(CHALLENGE)

If you're really up for a challenge, you could take this to the next level by implementing a non-i.i.d. statistical model such as a kth order Markov model, and coupling this with arithmetic coding. If you come up with a good model, maybe you can beat the current world records!