EE276: Homework #3 Solutions

Due on Friday Feb 2, 5pm

1. **Bad codes.**
   Which of these codes cannot be a Huffman code for any probability assignment? Justify your answers.

   (a) $\{0, 10, 11\}$.

   (b) $\{00, 01, 10, 110\}$.

   (c) $\{01, 10\}$.

   Solution:

   (a) $\{0, 10, 11\}$ is a Huffman code for the distribution $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$.

   (b) The code $\{00, 01, 10, 110\}$ can be shortened to $\{00, 01, 10, 11\}$ without losing its prefix-free property, and therefore is not optimal, so it cannot be a Huffman code. Alternatively, it is not a Huffman code because there is a unique longest codeword.

   (c) The code $\{01, 10\}$ can be shortened to $\{0, 1\}$ without losing its prefix-free property, and therefore is not optimal and not a Huffman code.

2. **Coin Toss Experiment and Golomb Codes**

   Kedar, Mikel and Naroa have been instructed to record the outcomes of a coin toss experiment. Consider the coin toss experiment $X_1, X_2, X_3, ...$ where X_i are i.i.d. $Bern(p)$ (probability of a H (head) is p), $p = 15/16$.

   (a) Kedar decides to use Huffman coding to represent the outcome of each coin toss separately. What is the resulting scheme? What compression rate does it achieve?

   (b) Mikel suggests he can do a better job by applying Huffman coding on blocks of r tosses. Construct the Huffman code for $r = 2$.

   (c) Will Mikel's scheme approach the optimum expected number of bits per description of source symbol (coin toss outcome) with increasing r? How does the space required to represent the codebook increase as we increase r?

   (d) Naroa suggests that, as the occurrence of T is so rare, we should just record the number of tosses it takes for each T to occur.
   To be precise, if Y_k represents the number of trials until the k^{th} T occurred (inclusive), then Naroa records the sequence:

   $$Z_k = Y_k - Y_{k-1}, \ k \geq 1, \tag{1}$$

   where $Y_0 = 0$

i. What is the distribution of Z_k, i.e., $P(Z_k = j), j = 1, 2, 3, ...$?

ii. Compute the entropy and expectation of Z_k.

iii. How does the ratio between the entropy and the expectation of Z_k compare to the entropy of X_k? Give an intuitive interpretation.

(e) Consider the following scheme for encoding Z_k, which is a specific case of Golomb Coding. We are showing the first 10 codewords.

Z	Quotient	Remainder	Code
1	0	1	1 01
2	0	2	1 10
3	0	3	1 11
4	1	0	0 1 00
5	1	1	0 1 01
6	1	2	0 1 10
7	1	3	0 1 11
8	2	0	00 1 00
9	2	1	00 1 01
10	2	2	00 1 10

i. Can you guess the general coding scheme? Compute the expected codelength of this code.

ii. What is the decoding rule for this Golomb code?

iii. How can you efficiently encode and decode with this codebook?

Solution:

(a) As we have just 2 source symbols (H & T), the optimal Huffman code is $\{0, 1\}$ for $\{H, T\}$. The compression rate (expected codelength) achieved is 1.

(b) One possible Huffman code for $r = 2$ is HH - 0, HT - 10, TH - 110, TT - 111. This has average codelength of $\frac{303}{512}$ bits/source symbol.

(c) We know that, using Shannon codes for extended codewords of length r, the expected average codelength has bounds:

$$H(X) \leq \ell_{avg} \leq H(X) + \frac{1}{r} \tag{2}$$

As Huffman coding is the optimal prefix code, it will always perform at least as well as the Shannon codes. Thus, as r increases, we will achieve optimal average codelength of $H(X) = H_b(1/16)$. As r increases, the number of codewords increases as 2^r. Also, the average size of the codewords, increases as $rH(X)$. Thus, effectively, the amount of space required to store the codebook increases exponentially in r.

(d) First of all, let us convince ourselves that if $Y_k = n$, then $\{Z_1, ..., Z_k\}$ represents the same information as $\{X_1, ..., X_n\}$. Also, Z_k random variables are independent of each other (since the coin tosses are independent) and have identical distributions.

i. Z_k has geometric distribution. Specifically $P(Z_k = j) = p^{j-1}(1-p), j \in 1, 2, 3, ...$ To see this, lets consider Z_1 for ease first. For $Z_1 = j$, we should have the first $j-1$ tosses to be H, while the j^{th} toss should be a T. This results in: $P(Z_1 = j) = p^{j-1}(1-p)$. Due to Z_k being i.i.d., we can generalize the results to Z_k.

ii. $H(Z_k) = \frac{H_b(p)}{(1-p)}$. As Z_k has geometric distribution, $E[Z_k] = \frac{1}{(1-p)}$.

iii. We observe that, $H(X) = \frac{H(Z_k)}{E[Z_k]}$. Intuitively, we can explain this as follows: Z_k represents the same amount of information as $X_{Y_{k-1}+1}, ..., X_{Y_{k-1}}$. As the X_i are i.i.d., the information content of Z_k is the same as the information content of $E[Z_k]$ i.i.d random variables X_i. Thus, $H(Z_k) = E[Z_k]H(X)$

(e) As an aside, the aim of the question was to introduce Golomb Codes. Golomb codes are infact optimal for sequences with geometric distribution, even though having a very simple construction. We have presented a specific case of Golomb codes in our example.

i. We are considering quotient of Z with respect to 4 in the unary form, and remainder in the binary form and concatenating them using a separator bit (the bit 1 in this case).
To calculate the expected codelength, observe that the length of codeword for $Z = 4k + m$ (where $0 \leq m \leq 3$), is $3 + k$. Thus, adding terms in groups of 4 and subtracting one term corresponding to $Z = 0$, we get,

$$\begin{aligned}
\bar{l} &= \sum_{k=0}^{\infty}(1-p)p^{4k-1}(1+p+p^2+p^3)(3+k) - 3\frac{1-p}{p} \\
&= (1-p)(1+p+p^2+p^3)\sum_{k=0}^{\infty}p^{4k-1}(3+k) - 3\frac{1-p}{p} \\
&= (1-p)(1+p+p^2+p^3)\frac{3-2p^4}{p(1-p^4)^2} - 3\frac{1-p}{p} \\
&= \frac{3-2p^4}{p(1-p^4)} - 3\frac{1-p}{p}
\end{aligned}$$

For the current source, this evaluates to 6.62 bits, as compared to the entropy $H(Z) = 5.397$ bits.

ii. It is easy to show that the code is prefix-free. Thus, the decoding can proceed by using the standard techniques of constructing a prefix-tree and using the same for decoding. However, note that in this case, the codebook size itself is unbounded, thus we might need to truncate the tree to some maximum depth.
There, is however a better and a simpler way, which does not require us to store the codebook explicitly. We can parse the symbol stream until we get the bit 1, this gives us the quotient q. Now, we read the next 2 bits, which gives us the remainder r. Using q and r, we can decode $Z = 4 * q + r$.

iii. The good thing about Golomb codes is that, even though the codebook size is unbounded, we do not need to explicitly store the codebook, but can have a simple code which generate the codes, and also to decode the codes.

3. **Arithmetic Coding.**

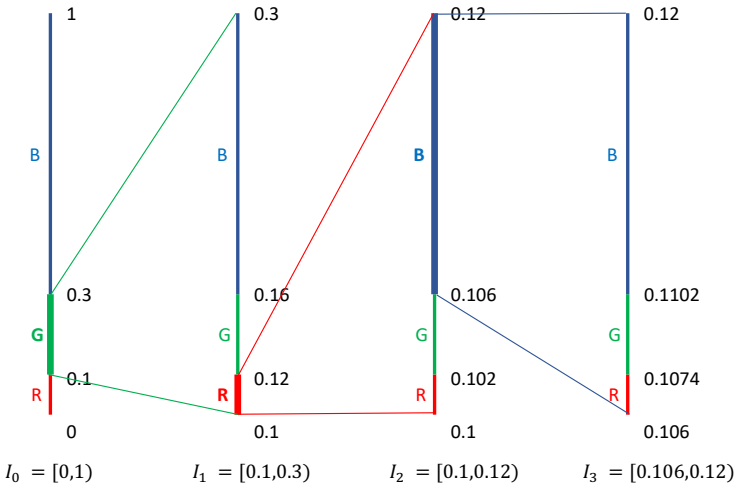

Figure 1: Illustration of arithmetic coding.

Note: Throughout this problem, we will work with digits rather than bits for simplicity. So the logarithms will be base 10 and the compressor will output digits $\{0, 1, \ldots, 9\}$.

This problem introduces a simplified version of arithmetic coding, which is itself based on Shannon-Fano-Elias coding. Arithmetic coding takes as input a sequence $x^n \in \mathcal{X}^n$ and a distribution q over $\mathcal{X}$. The encoder maintains an interval which is transformed at each step as follows:

- Start with $I_0 = [0, 1)$.
- For $i = 1, \ldots, n$:
  - Divide I_{i-1} into $|\mathcal{X}|$ half-open subintervals $\{I_{i-1}^{(x)},\ x \in \mathcal{X}\}$ with length of $I_{i-1}^{(x)}$ proportional to $q(x)$, i.e., $\left|I_{i-1}^{(x)}\right| = q(x)\,|I_{i-1}|$ for $x \in \mathcal{X}$.
  - Set $I_i = I_{i-1}^{(x_i)}$

Figure 1 shows an example of this for $\mathcal{X} = \{R, G, B\}$, $(q(R), q(G), q(B)) = (0.1, 0.2, 0.7)$ and $x^3 = GRB$. At the end of this process, the encoder selects a number in the interval I_n and outputs the digits after the decimal point for that number. In the example shown, the encoder can output 11, which corresponds to $0.11 \in [0.106, 0.12)$. While 1103 (corresponding to 0.1103) is also a valid output, the encoder tries to output the shortest possible valid sequence. The YouTube video `https://youtu.be/FdMoL3PzmSA` might be helpful for understanding this process even better.

(a) Briefly explain how the decoding might work in a sequential manner. You can assume that the decoder knows the alphabet, the distribution q and the length of the source sequence n.

(b) What is the length of interval I_n in terms of q and x^n?

(c) For the following intervals I_n obtained by following the above-described process for some x^n, find the length of the shortest output sequence (in digits):

i. $[0.095, 0.105)$
ii. $[0.11, 0.12)$
iii. $[0.1011, 0.102)$

In general, if the interval length is l_n, then the shortest output sequence has at most $\left\lceil \log \frac{1}{l_n} \right\rceil$ digits.

(d) Show that the length $l(x^n)$ for the arithmetic encoding output satisfies

$$l(x^n) \leq \log \frac{1}{q(x_1) \dots q(x_n)} + 1$$

(e) Suppose that X^n is i.i.d. with each X_i following distribution P, and we use arithmetic coding with $q = P$. Then show that

$$\frac{1}{n} E[l(X^n)] \leq H(X) + \frac{1}{n}$$

Compare this to Huffman coding over blocks of length n with respect to compression rate and computational complexity.

(f) Suppose both the encoder and the decoder have a prediction algorithm (say a neural network) that provides probabilities $q_i(x|x^{i-1})$ for all i's and all $x \in \mathcal{X}$. How would you modify the scheme such that you achieve

$$l(x^n) \leq \log \frac{1}{q_1(x_1) q_2(x_2|x_1) \dots q_n(x_n|x^{n-1})} + 1$$

Thus, if you have a prediction model for your data, you can apply arithmetic coding on it - good prediction translating to high probability, in turn translating to short compressed representations.

Solution:

(a) We first locate the interval $I_0^{(x)}$ containing the output number and decode x_1 as the corresponding x. Then we set $I_1 = I_0^{(x_1)}$ and locate the interval $I_1^{(x)}$ containing the output number to decode x_2. Repeating this n times, we get back x^n.

(b) Length of I_n is $q(x_1) \times q(x_2) \times \cdots \times q(x_n)$.

(c) i. Length 1, output sequence 1 (corresponding to 0.1)
ii. Length 2, output sequence 11 (corresponding to 0.11)
iii. Length 4, output sequence 1011 (corresponding to 0.1011) or 1012, etc.

(d) From part (b), the length of I_n is $q(x_1) \times q(x_2) \times \cdots \times q(x_n)$. Using part (c), we get

$$l(x^n) \leq \left\lceil \log \frac{1}{q(x_1) \dots q(x_n)} \right\rceil$$

The result follows using the fact that $\lceil x \rceil \leq x + 1$.

(e) Using result from part (d),

$$\begin{aligned} l(X^n) &\leq \log \frac{1}{P(X_1)\dots P(X_n)} + 1 \\ &= \sum_{i=1}^{n} \log \frac{1}{P(X_i)} \end{aligned}$$

Taking expectation and dividing by n,

$$\begin{aligned} \frac{1}{n} E[l(X^n)] &\leq \frac{1}{n} \sum_{i=1}^{n} E\left[\log \frac{1}{P(X_i)}\right] + \frac{1}{n} \\ &= \frac{1}{n} \sum_{i=1}^{n} H(P) + \frac{1}{n} \\ &= H(P) + \frac{1}{n} \end{aligned}$$

As n becomes large, arithmetic coding approaches entropy, which is the optimal compression rate. Huffman codes also approach the same limit, but are also optimal for any given n (although the gap becomes pretty small as n increases) (note that we can show the same $1/n$ upper bound for Huffman codes as seen in HW 2). Arithmetic coding has linear complexity in n, but Huffman codes have exponential complexity in n (block length) for storing the codebook.

(f) At step i, instead of dividing I_{i-1} into subintervals with lengths proportional to $q(x)$, we divide into subintervals with lengths proportional to $q_i(x|x^{i-1})$. Then the length of I_n is $q_1(x_1) \times q_2(x_2|x_1) \times \cdots \times q_n(x_n|x^{n-1})$ and the length satisfies the desired bound (using same proof as part d).

4. **Entropy Rate.**
Consider the Markov process from class taking values in $\{H, T\}$ with the joint probability distribution given as

$$P(X_1 = x_1, \dots, X_n = x_n) = P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_{i-1} = x_{i-1})$$

where $P(X_1 = H) = \frac{1}{2}$, $P(X_i = H|X_{i-1} = H) = \frac{3}{4}$ and $P(X_i = T|X_{i-1} = T) = \frac{3}{4}$ for all $i > 1$.

(a) Directly compute $P(X_2 = H)$ and extend that result to show that the process is stationary (we are only looking for the main idea, no need to write a long proof).

(b) Compute $H(X_n|X_{n-1}, \dots, X_1)$ as a function of n and find the limit as $n \to \infty$.

(c) Compute $\frac{1}{n} H(X_1, \dots, X_n)$ as a function of n and find the limit as $n \to \infty$. How does this relate to the result in part (b)?

Solution:

(a)

$$\begin{aligned} P(X_2 = H) &= P(X_1 = T, X_2 = H) + P(X_1 = H, X_2 = H) \\ &= P(X_1 = T)P(X_2 = H|X_1 = T) + P(X_1 = H)P(X_2 = H|X_1 = H) \\ &= \frac{1}{2} \times \frac{1}{4} + \frac{1}{2} \times \frac{3}{4} \\ &= \frac{1}{2} \end{aligned}$$

Thus, $P(X_2 = H) = \frac{1}{2}$. Now by repeating this exercise, it can be shown that $P(X_n = H) = \frac{1}{2}$ for all n. To show that the process is stationary one needs to show that for any n, k and $(x_1, \ldots, x_k) \in \{H, T\}^k$, the following holds:

$$P(X_1 = x_1, \ldots, X_k = x_k) = P(X_{n+1} = x_1, \ldots, X_{n+k} = x_k)$$

By the definition of the Markov process, we have

$$P(X_1 = x_1, \ldots, X_k = x_k) = P(X_1 = x_1) \prod_{i=2}^{k} P(X_i = x_i|X_{i-1} = x_{i-1})$$

Similarly by considering $P(X_1, \ldots, X_{n+k})$ and marginalizing over $X_1, \ldots, X_n$, we obtain

$$P(X_{n+1} = x_1, \ldots, X_{n+k} = x_k) = P(X_{n+1} = x_1) \prod_{i=n+2}^{n+k} P(X_i = x_i|X_{i-1} = x_{i-1})$$

The stationarity follows by observing that two products match termwise.

(b) Note that by the Markov property, X_n is independent of $X_1, \ldots, X_{n-2}$ given X_{n-1}. Thus,

$$H(X_n|X_{n-1}, \ldots, X_1) = H(X_n|X_{n-1})$$

By stationarity, this is same as $H(X_2|X_1)$. The joint distribution of (X_1, X_2) is

$$P(X_1 = H, X_2 = H) = \frac{1}{2} \times \frac{3}{4}$$

$$P(X_1 = T, X_2 = H) = \frac{1}{2} \times \frac{1}{4}$$

$$P(X_1 = H, X_2 = T) = \frac{1}{2} \times \frac{1}{4}$$

$$P(X_1 = T, X_2 = T) = \frac{1}{2} \times \frac{3}{4}$$

and the entropy $H(X_2|X_1)$ is just $h_2(\frac{3}{4})$ by direct computation. Thus $H(X_n|X_{n-1}, \ldots, X_1) = h_2(3/4)$. Since this does not depend on n, the limit is also $h_2(\frac{3}{4})$.

(c) Using chain rule for entropy

$$\frac{1}{n}H(X_1,\ldots,X_n) = \frac{1}{n}H(X_1) + \frac{1}{n}\sum_{i=2}^{n} H(X_i|X_{i-1},\ldots,X_1)$$

Using the result from part (b), we get

$$\frac{1}{n}H(X_1,\ldots,X_n) = \frac{1}{n} + \frac{(n-1)h_2(3/4)}{n}$$

The limit is $h_2(3/4)$ which matches the limit from part (b). Both parts compute the entropy rate of this process, and the proof of equality in the general case is given in the book C&T Theorem 4.2.1.

5. **Individual Sequences and a Universal Compressor.**
Note: Ignore integer constraints on codeword lengths throughout this problem.
Notation: $h_2(p) = -p\log_2 p - (1-p)\log_2(1-p)$ (= binary entropy function).

Let x^n be a given arbitrary binary sequence, with n_0 0's and n_1 1's ($n_1 = n - n_0$). You are also provided a compressor C_q which takes in any arbitrary distribution q on $\{0,1\}$ as a parameter, and encodes x^n using:

$$\bar{L}_q(x^n) = \frac{1}{n}\log\frac{1}{q(x^n)}$$

bits per symbol where $q(x^n) := \prod_{i=1}^{n} q(x_i)$.

(a) Given the sequence x^n, what binary distribution $q(x)$ will you choose as a parameter to the compressor C_q, so that $\bar{L}_q(x^n)$ is minimized. Your answer (values of $q(0)$ and $q(1)$) will be expressible in terms of n, n_0 and n_1.

(b) When compressing any given individual sequence x^n, we also need to store the parameter distribution $q(x)$ (required for decoding). Show that you can represent the optimal parameter distribution $q(x)$ from part (a) using $\log(n+1)$ bits. You can assume that the decoder knows the length of the source sequence n.

(c) Show that the effective compression rate for compressing x^n (in bits per source symbol) with the distribution q from part (a) is $h_2(n_1/n) + \log(n+1)/n$.

(d) Now suppose that we apply the scheme above to X^n sampled from an i.i.d. $Ber(p)$ distribution. Show that the expected compression rate approaches $h_2(p)$ as $n \to \infty$, i.e., the scheme is a *universal* compressor for i.i.d. sources.

Solution:

(a) For $q(0) = 1-q, q(1) = q$, we have

$$\bar{L}_q = \frac{1}{n}\log\frac{1}{(1-q)^{n_0}q^{n_1}} = -\frac{n_0}{n}\log(1-q) - \frac{n_1}{n}\log(q).$$

We see that $\bar{L}_q$ is convex in q, and taking derivative w.r.t q gives $q^* = \frac{n_1}{n}$.

(b) By the previous part, it suffices to store $n_1 \in \{0, 1, \cdots, n\}$ for full knowledge of $q(x)$. Hence, $\log(n+1)$ bits are enough (ignoring integer constraints).

(c) Simply substitute $q = q^*$ in the $\bar{L}_q$ expression in part (a) solution above, and add the contribution from part (b) (normalized by n).

$$\bar{L}_q + \frac{\log(n+1)}{n} = h_2\left(\frac{n_1}{n}\right) + \frac{\log(n+1)}{n}.$$

(d) Let N_1 denote the number of 1's in X^n. Then we get that the expected compression rate (call it R_n) is

$$R_n = \mathcal{E}\left[h_2\left(\frac{N_1}{n}\right)\right] + \frac{\log(n+1)}{n}$$

For the first term, we can use Jensen's inequality and the concavity of entropy to get

$$R_n \leq h_2\left(\mathcal{E}\left[\frac{N_1}{n}\right]\right) + \frac{\log(n+1)}{n}$$

Now, $\mathcal{E}\left[\frac{N_1}{n}\right] = p$ which gives us

$$R_n \leq h_2(p) + \frac{\log(n+1)}{n}$$

Taking the limit,

$$\lim_{n\to\infty} R_n \leq h_2(p)$$

But since the entropy $h_2(p)$ is also a lower bound on any compression scheme, we must have

$$\lim_{n\to\infty} R_n = h_2(p)$$

Thus the scheme is universal for i.i.d. sources.

6. **Prefix-free Codes for Integers.**
We saw in class that LZ77 compression requires storing of integers representing the match position and length. In this problem we consider binary prefix codes over the set of natural numbers $\mathbb{N} = \{1, 2, 3, \dots\}$, where a codeword d_j is associated with the natural number j. *Notation*: 0^j denotes a sequence of j zeros.

(a) Consider the *unary* code $u_j = 0^j 1$ with length $l_{u_j} = j+1$. Is this code prefix-free?

(b) Consider now the code b_j, which is the binary representation of j (e.g., $b_1 = 1$, $b_5 = 101$). Note that the codelength of b_j is given by: $l_{b_j} = \lfloor \log_2 j \rfloor + 1$. Give examples to show that this code is not prefix-free.

(c) We combine the ideas in parts (a) and (b) to design a shorter prefix-free code than that in part (a). We first encode the length of b_j in unary, followed by b_j itself, i.e., $c_j = 0^{\lfloor \log_2 j \rfloor + 1} 1 b_j$ with length $l_{c_j} = 2\lfloor \log_2 j \rfloor + 3$. Show that this code is prefix-free.

(d) **[Bonus]** Can you further improve the idea in part (c) to get a prefix-free code with length below $\log_2 j + 2\log_2\log_2 j + C$ where C is some constant (ignore $j = 1$ for simplicity)?

Solution:

(a) This is a prefix-free code. Different codes u_j have different number of zeros before 1.

(b) This code is not prefix-free. For example, $b_1 = 1$ is a prefix of $b_3 = 11$.

(c) Assume for the sake of contradiction that c_j is a prefix of $c_{j'}$ for $j \neq j'$. Comparing the number of zeros in the front, we must have $\lfloor \log_2 j \rfloor = \lfloor \log_2 j' \rfloor$. Hence, b_j and $b_{j'}$ must have the same length, and the prefix assumption implies $b_j = b_{j'}$. Since b_j is the binary representation of j, we then have $j = j'$, a contradiction!

(d) You can first represent $\lfloor \log_2 j \rfloor + 1$ (the length of b_j) using the code in part (c) and then encode j as b_j. The codeword would look like $c_{\lfloor \log_2 j \rfloor + 1} b_j$ or more explicitly

$$0^{\lfloor \log(\lfloor \log_2 j \rfloor + 1) \rfloor + 1} 1 b_{\lfloor \log_2 j \rfloor + 1} b_j$$

It is easy to see how one can decode this instantaneously, implying that this is prefix-free. The length is

$$\lfloor \log_2 j \rfloor + 2\lfloor \log(\lfloor \log_2 j \rfloor + 1) \rfloor + 4$$

which satisfies the condition in the problem for some C large enough.