COMP2610/COMP6261
Tutorial 8 Sample Solutions

Tutorial 8: Source Coding

Young Lee and Bob Williamson
Tutors: Debashish Chakraborty and Zakaria Mhammedi

Week 10 (9th – 13th Oct), Semester 2, 2017

1. (a) We have

$$\begin{aligned}H(X) &= -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - \frac{31}{128}\log\frac{31}{128} - \frac{1}{128}\log\frac{1}{128}\\ &= 1.55.\end{aligned}$$

(b) The expected code length is

$$L(C, X) = \frac{1}{2}\times 1 + \frac{1}{4}\times 2 + \frac{31}{128}\times 3 + \frac{1}{128}\times 3 = \frac{7}{4}.$$

(c) The code lengths for X are

$$\lceil \log_2 \frac{1}{1/2}\rceil = 1,\ \lceil \log_2 \frac{1}{1/4}\rceil = 2,\ \lceil \log_2 \frac{31}{1/128}\rceil = 3,\ \text{and}\ \lceil \log_2 \frac{1}{1/128}\rceil = 7.$$

An example of a prefix Shannon code for X would be:

$$C_S = \{0, 10, 110, 1110001\}$$

The expected code length would be

$$L(C_S, X) = \frac{1}{2}\times 1 + \frac{1}{4}\times 2 + \frac{31}{128}\times 3 + \frac{1}{128}\times 7 = 1.78125.$$

(d) We have

$$q_1 = 2^{-1} = \frac{1}{2},\ q_2 = 2^{-2} = \frac{1}{4},\ q_3 = 2^{-3} = \frac{1}{8},\ q_4 = 2^{-3} = \frac{1}{8},$$

(And $Z = 1$)

(e) By the definition of $D(\mathbf{p}||\mathbf{q})$ we have

$$\begin{aligned}D(\mathbf{p}||\mathbf{q}) &= \sum_{i=1}^{4} p_i \log\frac{p_i}{q_i}\\ &= \frac{1}{2}\times\log_2\frac{1/2}{1/2} + \frac{1}{4}\times\log_2\frac{1/4}{1/4} + \frac{31}{128}\times\log_2\frac{31/128}{1/8} + \frac{1}{128}\times\log_2\frac{1/128}{1/8}\\ &= \frac{31}{128}\times\log_2\frac{31}{16} + \frac{1}{128}\times 4\\ &= 0.200.\end{aligned}$$

So we have $D(\mathbf{p}||\mathbf{q}) = L(C, X) - H(X)$ as we would expect. We also note that $L(C_S, X)$ is greater (i.e. the code is worse) than C.

(f) The steps of Huffman coding would be:

- from set of symbols $\{x_1, x_2, x_3, x_4\}$ with probabilities $\{1/2, 1/4, 31/128, 1/128\}$, merge the two least likely symbols x_3 and x_4. The new meta-symbol x_3x_4 has probability $1/4$.
- from set of symbols $\{x_1, x_2, x_3x_4\}$ with probabilities $\{1/2, 1/4, 1/4\}$, merge the two least likely symbols x_2 and x_3x_4. The new meta-symbol $x_2x_3x_4$ has probability $1/2$.
- from set of symbols $\{x_1, x_2x_3x_4\}$ with probabilities $\{1/2, 1/2\}$, merge the two least likely symbols x_1 and $x_2x_3x_4$. The new meta-symbol $x_1x_2x_3x_4$ has probability 1, so we stop.

We then assign a bit for each merge step above. This is summarised below. We then read off the resulting codes by tracing the path from the final meta-symbol to each original symbol. This gives the code $C = \{0, 10, 110, 111\}$. (Note, we could equally derive $C = C_H$ depending on how we labelled the penultimate merge operation.)

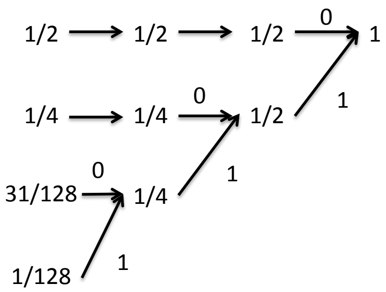

Figure 1: Huffman code.

2. (a) We know $4 = 2^2$ and $0.25 = 2^{-2}$, so $4.25_{10} = \mathtt{10.01}_2$.

   (b) One way to proceed would be to first deal with the part before the decimal point: this is $8 = 2^3$. For the part after the decimal point, we multiply $0.1 \cdot 2 = 0.2, 0.2 \cdot 2 = 0.4, 0.4 \cdot 2 = 0.8, 0.8 \cdot 2 = 1.6, 0.6 \cdot 2 = 0.2, \ldots$, where in each step we multiply using the decimal part of the previous step. If the result of the multiplication is greater than 1, there is a bit of 1 in the corresponding power of 2. In this case we note that we end up in an infinite loop as we get back to multiplying $0.2 \cdot 2$. So we conclude the representation is $\mathtt{1000.0}\overline{\mathtt{0011}}_2$.

   Here is another way to show it. We know $8 = 2^3$. Now, $\log_2 0.1 = -3.3219 < -3$, so the first three bits past the decimal point are zero. At this stage we can reduce the problem to finding the binary expansion for $0.1 - 2^{-4}$. Now, $\log_2(0.1 - 2^{-4}) = -4.7370 < -4$, so the fifth bit is the next to be activated. Repeating, we have $\log_2(0.1 - 2^{-4} - 2^{-5}) = -7.3219 < -7$, so the next bit to be active is the 8th. In fact, we can observe that $0.1 - 2^{-4} - 2^{-5} = 0.0063 = \frac{0.1}{16}$, so that we are effectively recomputing the binary expansion of 0.1, with digits shifted by $\log_2 16 = 4$ places. Thus, the representation will be $\mathtt{1000}.0\overline{0011}_2$. To verify this, note that

$$\sum_{k=1}^{\infty} \frac{1}{2^{4k}} + \frac{1}{2^{4k+1}} = \sum_{k=1}^{\infty} \frac{1}{2^{4k}} \cdot \frac{3}{2} = \sum_{k=1}^{\infty} \frac{1}{16^k} \cdot \frac{3}{2} = \frac{3}{2 \cdot 15} = \frac{1}{10}.$$

3. Suppose $\mathcal{A}_X = \{x_1, \ldots, x_4\}$ where $x_1 = \mathtt{a}$ and so on. We need to compute the cumulative probabilities $F(x)$, the modified probabilities $\bar{F}(x)$, and truncate the binary expansions of the latter to the first $\ell(x) = \lceil \log_2 \frac{1}{p(x)} \rceil + 1$ bits. These are summarised in the table below.

   Evidently, removing the last bit from every codeword means the result will no longer be a prefix code (since we have e.g. `1` and `110` as two codewords, with the first a prefix of the second).

   To decode `10001`, we compute the codeword intervals starting from the first bit:

   - `1` has interval $[0.1, 1.0)_2 = [0.5, 1)_{10}$. This overlaps with the intervals for x_2, x_3, x_4, so we can't conclude anything.

i	$p(x_i)$	$F(x_i)$	$[F(x_{i-1}, F(x_i))$	$\bar{F}(x_i)$	$\bar{F}(x_i)_2$	$\ell(x_i)$	Codeword
1	0.25	0.25	[0, 0.25)	0.125	0.001	3	001
2	0.5	0.75	[0.25, 0.75)	0.5	0.10	2	10
3	0.125	0.875	[0.75, 0.875)	0.8125	0.1101	4	1101
4	0.125	1.0	[0.875, 1.0)	0.9375	0.1111	4	1111

Table 1: Shannon-Fano-Elias code.

- `10` has interval $[0.10, 0.11)_2 = [0.5, 0.75)_{10}$. This is contained in the interval for x_2 (viz. $[0.25, 0.75)$), so we can conclude the first symbol is x_2. At this stage we can forget about the first two bits, since the SFE code for a sequence is just based on the extension i.e. we just concatenate the codewords for the individual outcomes.
- `0` has interval $[0.0, 0.1)_2 = [0, 0.5)_{10}$. This overlaps with the intervals for x_1, x_2, so we can't conclude anything.
- `00` has interval $[0.00, 0.01)_2 = [0, 0.25)_{10}$. This is exactly the interval for x_1, so we can conclude that the second symbol is x_1. We can compute the length of the interval for x_1 on the fly (though we already know it is 3), and conclude that there is one redundant bit we can skip over.

4. (a) We assume fixed probabilities.
   - We start with the symbol intervals as computed in the previous question:
   $$[0.0000, 0.2500), [0.2500, 0.7500), [0.7500, 0.8750), [0.8750, 1.0000).$$
   - The first symbol is a `c`. So, we slice up the interval $[0.7500, 0.8750)$. Since we are using the same probabilities in every iteration, we end up with:
   $$[0.7500, 0.7812), [0.7812, 0.8438), [0.8438, 0.8594), [0.8594, 0.8750).$$
   - This is the end of the stream. So, we end up in the final interval, viz. $[0.8594, 0.8750)$. The midpoint of this interval is 0.86718750. This has binary representation `0.1101111`. The probability of `c□` is $(1/8)(1/8) \approx 0.0156$. The number of bits to output is $\lceil \log_2 1/0.00156 \rceil + 1 = 7$. So, the codeword is `1101111`.

   (b) We assume fixed probabilities.
   - We start with the symbol intervals as computed in the previous part:
   $$[0.0000, 0.2500), [0.2500, 0.7500), [0.7500, 0.8750), [0.8750, 1.0000).$$
   - The first symbol is a `c`. So, we slice up the interval $[0.7500, 0.8750)$. Since we are using the same probabilities in every iteration, as per the previous part, we end up with:
   $$[0.7500, 0.7812), [0.7812, 0.8438), [0.8438, 0.8594), [0.8594, 0.8750).$$
   - The second symbol is a `a`. So, we slice up the interval $[0.7500, 0.7812)$. Since we are using the same probabilities in every iteration, we end up with:
   $$[0.7500, 0.7578), [0.7578, 0.7734), [0.7734, 0.7773), [0.7773, 0.7812).$$
   - This is the end of the stream. So, we end up in the final interval, viz. $[0.7773, 0.7812)$. The midpoint of this interval is 0.77929688. This has binary representation 0.1100011110. The probability of `ca□` is $(1/4)(1/8)(1/8) \approx 0.0039$. The number of bits to output is $\lceil \log_2 1/0.0039 \rceil + 1 = 9$. So, the codeword is `110001111`.

   The codeword for `c` isn't a prefix for that for `ca`. So, the code isn't just computing a codeword for each symbol and concatenating them (as we have done for Huffman and SFE codes). Arithmetic coding implicitly associates every sequence with its own codeword.

(c) We assume adaptive probabilities.

- We know $p(\square) = 0.25$. From the virtual counts, we have $p(\cdot|\epsilon) = \frac{0+1}{0+3} \cdot (1 - p(\square)) = 0.25$, since at this stage we have not observed anything. So, we start off with the intervals

$$[0.0000, 0.2500), [0.2500, 0.5000), [0.5000, 0.7500), [0.7500, 1.0000).$$

- The first symbol is c. So, we slice up the interval $[0.5, 0.75)$. From the virtual counts, we have $p(\cdot|\mathtt{c}) = (\frac{0+1}{1+3}, \frac{0+1}{1+3}, \frac{1+1}{1+3}) \cdot (1 - p(\square)) = (1/4, 1/4, 1/2) \cdot (3/4) = (3/16, 3/16, 3/8)$. So, we have the intervals

$$[0.5000, 0.5469), [0.5469, 0.5938), [0.5938, 0.6875), [0.6875, 0.7500)$$

  remembering that we need to scale the probabilities by the length of the interval, viz. $0.75 - 0.5 = 0.25$.
- The next symbol is a. So, we slice up the interval $[0.5, 0.5469)$. From the virtual counts, we have $p(\cdot|\mathtt{ca}) = (\frac{1+1}{2+3}, \frac{0+1}{2+3}, \frac{1+1}{2+3}) \cdot (1 - p(\square)) = (2/5, 1/5, 2/5) \cdot (3/4) = (3/10, 3/20, 3/10)$. So, we have the intervals

$$[0.5000, 0.5141), [0.5141, 0.5211), [0.5211, 0.5352), [0.5352, 0.5469)$$

  remembering that we need to scale the probabilities by the length of the interval, viz. $0.5469 - 0.5352 \approx 0.0117$.
- This is the end of the stream. So, we end up with the last interval, $[0.5352, 0.5469)$. The midpoint is 0.54101562. This has representation 0.100010101. The probability of $\mathtt{ca}\square$ is $(1/4)(3/16)(1/4) \approx 0.0177$. The number of bits to output is $\lceil \log_2(1/0.0177) \rceil + 1 = 8$. So, the codeword is `10001010`.