INFORMATION THEORY
University of Amsterdam, 2018
Team Name: Latin Square

Homework set 3

Contents

Question 1

(a) We sort x based on its probabilities (high to low) and combine the two lowest probability each step until it reaches 1:

x	$p_x(x)$		**code**
1	$\frac{7}{8}$	\0	0
0	$\frac{1}{8}$	/1	1

The average code word length can then be computed by summing over the probabilities multiplied by their codeword length: $\frac{7}{8} * 1 + \frac{1}{8} * 1 = 1$.

(b) We start by combining the two lowest probabilities. Since 10 and 01 have the same probability, we simply pick the one that is ranked the lowest. So in step 1 we combine 00 and 01. After this we combine the result of step 1 and 10. In the final step we combine the result of the last step and 11, leading to:

xy	$p_{xy}(xy)$				**code**
11	$\frac{49}{64}$	...	...	\0	0
10	$\frac{7}{64}$	...	\0	/1	10
01	$\frac{7}{64}$	\0	/1		110
00	$\frac{1}{64}$	/1			111

This gives us the following average codeword length: $\frac{49}{64} * 1 + \frac{7}{64} * 2 + \frac{7}{64} * 3 + \frac{1}{64} * 3 = 1\frac{23}{64}$

(c) We first sort xyz from the highest to the lowest probability. We then combine 000 and 100 since they 000 has the lowest probability and 100 has one of the lowest probabilities and happens to be ranked the lowest. In step 2 we combine 001 and 010. We then combine the results of step 1 and step 2 in step 3. In step 4 we combine the result of step 3 with 110. In step 5 this we take the 010 and 101. In the sixth step we combine step 4 and 5. And in the final step we take the result of step 6 with 111, resulting into:

xyz	$p_{xyz}(xyz)$								**code**
111	$\frac{343}{512}$	...	...	...	...	...	...	\0	0
010	$\frac{49}{512}$	...	...	...	...	\0		—	100
101	$\frac{49}{512}$	...	...	...	...	/1	\0	—	101
110	$\frac{49}{512}$	...	...	...	\0		—	/1	110
001	$\frac{7}{512}$	...	\0		—	...	/1		11100
010	$\frac{7}{512}$	...	/1	\0	/1				11101
100	$\frac{7}{512}$	\0	...	/1					11110
000	$\frac{1}{512}$	/1							11111

The average codeword length therefore turns out to be: $\frac{343}{512} * 1 + \frac{49}{512} * 3 + \frac{49}{512} * 3 + \frac{49}{512} * 3 + \frac{7}{512} * 5 + \frac{7}{512} * 5 + \frac{7}{512} * 5 + \frac{1}{512} * 5 = 1\frac{382}{512}$. Note that this code is not optimal, since $8 = k(3-1)+1$ has no solution for $k \in \mathbb{N}$.

(d)
- For $n = 1$
  - The average codeword length divided by n is equal to 1
  - The entropy is equal to $\frac{7}{8} \cdot \log \frac{8}{7} + \frac{1}{8} \cdot \log 8 \approx 0.544$.
- For $n = 2$
  - The average codeword length divided by n is equal to $\frac{1\frac{23}{64}}{2} = \frac{87}{128}$.
  - The entropy is equal to $\frac{49}{64} \cdot \log \frac{64}{49} + \frac{7}{64} \cdot \log \frac{64}{7} + \frac{7}{64} \cdot \log \frac{64}{7} + \frac{1}{64} \cdot \log 64 \approx 1.087$.
- For $n = 3$
  - The average codeword length divided by n is equal to $\frac{447}{512}$.
  - The entropy is equal to $\frac{343}{512} \cdot \log \frac{512}{343} + \frac{49}{512} \cdot \log \frac{512}{49} + \frac{49}{512} \cdot \log \frac{512}{49} + \frac{49}{512} \cdot \log \frac{512}{49} + \frac{7}{512} \cdot \log \frac{512}{7} + \frac{7}{512} \cdot \log \frac{512}{7} + \frac{7}{512} \cdot \log \frac{512}{7} + \frac{1}{512} \cdot \log 512 \approx 1.631$

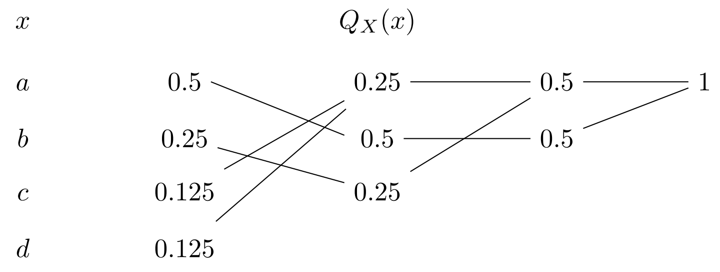
We can see the entropy increasing with N. The average codeword length divided by N however stays below 1.

(e) For $n = 100$ we would have k^n possible outcomes, with k being the length of the alphabet. This would lead to $k^n - 1$ steps for constructing the binary Huffman, which has a high complexity.

(f)

z	**pz(z)**				**code**
2	$\frac{3}{10}$	...	...	\0	0
3	$\frac{2}{10}$	...	\0	—	10
4	$\frac{2}{10}$	...	-1	/1	11
1	$\frac{1}{10}$	\0			120
5	$\frac{1}{10}$	-1	/2		121
6	$\frac{1}{10}$	/2			122

Question 2



(a) For the distribution Q_X, we have:

$$\begin{aligned} Pr(X=a) &= \frac{1}{2}, \text{ codeword } C(a)=1 \\ Pr(X=b) &= \frac{1}{4}, \text{ codeword } C(b)=01 \\ Pr(X=c) &= \frac{1}{8}, \text{ codeword } C(c)=001 \\ Pr(X=d) &= \frac{1}{8}, \text{ codeword } C(d)=000 \end{aligned}$$

We know this prefix-free code is optimal as it has been generated using Huffman's algorithm.

(b) We know that expected code length $\mathbb{E}[l(C(X))] = \sum_{x\in\chi} Q_X(x)l(C(x))$. In our case this is 1.75 bits.

(c) If we use the same $l(C(X))$ in the formula for the P_X distribution, that is by using the wrong code: $\mathbb{E}[l(C(X))] = \sum_{x\in\chi} P_X(x)l(C(x))$. We get as a result an expected code length of 2.25 bits.

(d) We know using the fact that $l(x) = \lceil -\log_2 Q_X(x)\rceil = \left\lceil \log_2 \frac{1}{Q_X(x)} \right\rceil$ for all $x \in \chi$ that:

$$\begin{aligned} \mathbb{E}[l(C(X))] &= \sum_{x\in\chi} P_X(x) \left\lceil \log_2 \frac{1}{Q_X(x)} \right\rceil \\ &\leq \sum_{x\in\chi} P_X(x) \left(\log_2 \frac{1}{Q_X(x)} + 1 \right) \\ &\leq \sum_{x\in\chi} P_X(x) \left(\log_2 \frac{1}{Q_X(x)} \right) + \sum_{x\in\chi} P_X(x) \\ &= \sum_{x\in\chi} P_X(x) \log_2 \frac{P_X(x)}{Q_X(x)} \frac{1}{P_X(x)} + 1 \\ &= \sum_{x\in\chi} P_X(x) \log_2 \frac{P_X(x)}{Q_X(x)} + \sum_{x\in\chi} P_X(x) \log_2 \frac{1}{P_X(x)} + 1 \\ &= D(P_x||Q_x) + H(P_x) + 1 \end{aligned}$$

To prove the lower bound it is enough to notice that:

$$\begin{aligned} \mathbb{E}[l(C(X))] &\geq \sum_{x\in\chi} P_X(x) \left\lceil \log_2 \frac{1}{Q_X(x)} \right\rceil \\ &= D(P_x||Q_x) + H(P_x) \end{aligned}$$

Question 3

(a)

k	$P_X(k)$	F_k	code length $\lceil -\log P_X(k) \rceil$	code C(k)
1	0.5	0.0	1	0
2	0.25	0.5	2	10
3	0.125	0.75	3	110
4	0.125	0.875	3	111

(b)

k	$P_X(k)$	F_k	code length $\lceil -\log P_X(k) \rceil$	code C(k)
1	$\frac{1}{3}$	0.0	2	00
2	$\frac{1}{3}$	$\frac{1}{3}$	2	01
3	$\frac{1}{3}$	$\frac{2}{3}$	2	10

(c) Denoting C(i) the coding of the i^{th} word, we want to see the condition for which the code of the next word $C(i+1)$ is a prefix of $C(i)$. This can happen only if $F_{i+1} - F_i < 2^{-l(C(i))}$, as otherwise the least significant bit would change and $C(i)$ would not be a prefix of $C(i+1)$ anymore. For the Shannon code, we have

$$\begin{aligned} l_S(C(i)) &= \lceil -\log P_X(i) \rceil \\ l_S(C(i)) &\geq -\log P_X(i) \\ 2^{l_S(C(i))} &\geq \frac{1}{P_X(i)} \\ P_X(i) &\geq 2^{-l_S(C(i))} \end{aligned}$$

Furthermore, in the shannon code, $F_{i+1} - F_i = P_X(i)$. We have shown that $P_x(i) \geq 2^{-l(C(i))}$, and as $C(i)$ is prefix of $C(i+1)$ only if $F_{i+1} - F_i < 2^{-l(C(i))}$, this can never happen.

Also, $F_j - F_i > P_X(i)$ for all $j > i$ so they cannot be prefix of $C(i)$ either. Finally, for all $j < i$, as all the F values are increasing, $F_j < F_i$, so $C(i)$ cannot be prefix of $C(j)$. Thus, the Shannon code if prefix free.

(d)

$$l_S(P_X) = \sum_{i=1}^{m} P_X(i) l(C(i)) \sum_{i=1}^{m} P_X(i) \lceil -\log P_X(i) \rceil \geq -\sum_{i=1}^{m} P_X(i) \log P_X(i) = H(X)$$

$$\begin{aligned} l_S(P_X) &= \sum_{i=1}^{m} P_X(i) l(C(i)) \sum_{i=1}^{m} P_X(i) \lceil -\log P_X(i) \rceil \leq -\sum_{i=1}^{m} P_X(i)(\log P_X(i) + 1) \\ &= -\sum_{i=1}^{m} P_X(i) \log P_X(i) - \sum_{i=1}^{m} P_X(i) = H(X) + 1 \end{aligned}$$

Question 4

(a) The strategy to sample from Z$_1$ involves tossing a coin either once or twice. Since the coin we use is fair, the probability of heads and tails is $1/2$. Accordingly, we ascribe to event a heads, and in case of tails, we repeat the coin toss, and ascribe to event b heads, and to c tails. Given the above strategy, the average number of coin tosses is $1/2 \times 1 + 1/2 \times 2 = 1.5$ on average. As such, the expected number of coin tosses is equivalent to the entropy of the distribution.

(b) The binary representation of 1/3 is 0.010101..., and the representation of 2/3 is 0.101010.... These numbers have no finite binary representation. The *atoms* for the binary expansion of $1/3$ are all numbers of the form $2^{k+1 \mod 2}$, while the atoms of 2/3 are the numbers of the form $2^{k \mod 2}$.

(c) The number 2^{-k} can be represented in binary by means of a number of 0s followed by a 1. As such, the representations of these numbers are prefix-free, since the number of 0s preceeding the 1 is unique. If we interpret these representations to be a code, we may apply Kraft's inequality and conclude since the total sum of these numbers 2^{-k} is smaller than or equal to 1, the can construct a binary tree from which we can sample.

(d) The sampling tree of Z$_2$ is such that if one takes the binary expansion denoting $1/3$, one may consider a sample to of event a whenever a 0 occurs in the series instead of a 1, and to be of event b if a 1 occurs instead of a 0. Otherwise, flip the coin again.

(e) Since $P_{Y_X}(y) = 2^{d(y)}$, the entropy of the distribution is as follows.

$$H(Y_X) = \sum_{y \in Y} 2^{-d(y)} \log \frac{1}{2^{-d(y)}}$$

The outcome of a number of sampling random bits may be represented as a sequence of 1s and 0s. The probability of any sequence of length n of bits may be expressed as follows: $1/2^n = 2^{-n}$. The number of coin required to describe a probability (P_X) of the form $2^{-n(x)}$ is n, leaving us with the following equation for the expected number of coin tosses.

$$\begin{aligned} ET(X) &= \sum_{x \in X} 2^{-n(x)}(x) \log \frac{1}{2^{-n(x)}} \\ &= H(Y_X) \end{aligned}$$

(f)

(g) The first inequality follows from subexercise e, which shows that equality holds if $P_{Y_X} = 2^{-d(y)}$. The second part is shown as follows.

Question 5

(a) At the beginning we realised that entropy of X is $H(X) = 8.079313589591118.10^{-2}$. Moreover, we noticed that encoding one bit using Huffman tree is not efficient, because each bit will be encoded to one bit. This is very inefficient and will results in no compression at all. We can easily solve this using words of length n (We used $n = 20$, because " 10000 mod 20 = 0" and for a larger n, we don't have enough Ram.). More zeros are in the word then the word is more probable. The probability of word can be calculated as follows:

$$P(w) = 0.99^k * 0.01^{k-1}$$

where k is number of zeros in that word.

Using this, we construct Huffman tree from words of length n. We can see that number of leafs is 2^n, which grows exponentially.

We ran our compressor over 100 files and got (in average) 934 bites length of compressed file, with $n = 20$. The entropy of compressed file is around 0.8770624

(b) If we define compress ratio as an average ratio between uncompressed data length and compressed data length, we can calculate that our compress ratio is around: $10000/934$. That is around 10.7. This tell us that in average we compress our data 10 times.

Calculating the entropy of the source, we get $H(X) = 8.079313589591118.10^{-2}$ compare to the entropy of compress data $= 0.8770624$. The entropy of compress data is around 10 times bigger then the entropy of the source.

(Maybe we can define the compress ratio as ratio between entropy of compress data to the entropy of the source. In the end we get very similar number).

Comparison to Zip(LZMA) Using we zip program, we get in the average around 420 bytes length, which is at least 2 times better then we can do. (So, there is still place for improvement.)

Code description *Tree construction:* We created a priority queue, where elements are sub-trees of Huffman tree. The probability is the measure of priority (lower, then higher is the sub-tree in the queue). Then we take 2 elements from the priority queue and join them together plus insert this new sub-tree to the queue with new probability, which is sum of the sub-trees's probabilities. We loop until we have one element in the queue, which is whole Tree.

From Tree we construct Hashmap for encoding (key=input string, value=corresponding c), which we use from encoding. (Map each word of length n there exist value in the Hasmap, which will replace the corresponding word.

To decode the string, we just start from the root and go down to the corresponding Leaf using read bits, which show us direction, where should we go in the tree. After we reached the leaf (we found the uncompressed word), we replace the word with the Leaf word and continue from the root again.

To check covertness of our code, we use this idea : decode(encode(word)) = word.