Communication Systems Laboratory
Lab 4: Source Coding

(Report Due: 23:59, November 18, 2020)

1 Overview

After the journey of quantum information processing, now we are back to the fully classical (digital) zone (home sweet home?). In this lab, we are going to implement coding for discrete sources. Here, the sources are classical information, which means that one can designate every possible outcome of such sources (e.g. it could be something like '010', ♠, or ☺, etc.) with certain probability value[1]. Your goal in this lab is to implement a classical date (noiseless) compression scheme by e.g. Matlab to see that we can indeed remove the redundancy hidden in the discrete sources.

2 Experiments

We are going to explore Huffman coding from three perspectives. First, beginning with a toy example in Table 1, we analyze the entropy, the Huffman tree, the Huffman dictionary, and the average codeword length by hand. Next we implement three main blocks for Huffman coding: `huffman_dict` for building the Huffman dictionary from the probability distribution, `huffman_enc` for Huffman encoding, and `huffman_dec` for Huffman decoding. With these functions, we study the average codeword length of Huffman coding through Monte-Carlo simulations.

1. **(35 points) Information theory and Huffman coding:** First let us consider a random symbol X whose outcomes and the associated probabilities been given in Table 1. Show your answers to the following problems like what we did in the lecture.

   (a) **(4 points)** Calculate the entropy of X, i.e. $H[X]$.

   (b) **(10 points)** Construct the Huffman tree and the Huffman dictionary for X.

   (c) **(4 points)** Verify that the codewords constructed by your Huffman tree satisfy the Kraft inequality or not.

   (d) **(4 points)** Find the average codeword length $\overline{L}$ for the dictionaries in Problem 1b. Do they satisfy the source-coding theorem?

   (e) **(4 points)** Encode the sequence of symbols in (1) using the Huffman tree in Problem 1b.

   $$\{\mathrm{g}, \mathrm{a}, \mathrm{c}, \mathrm{a}, \mathrm{b}\}. \tag{1}$$

   (f) **(4 points)** Decode the bitstream in Problem 1e using the Huffman tree in Problem 1b.

[1] At this point, I believe you couldn't help but think about the scenarios that each outcome of the sources could be something like $|010\rangle$ (endowed with certain probability value). Yes, underlying source is now *quantum information*. If you really love it so much, please go ahead to see *Schumacher's noiseless compression of quantum bit*.

(g) (**5 points**) Let T_ε^n denote the typical set of X with $\varepsilon = 0.1$ and $n = 10$. Find 10 members in the set T_ε^n.

2. (**35 points**) **Implementation of Huffman coding:** In what follows, we implement three main blocks of Huffman coding using Matlab.

(a) (**15 points**) Implement a Matlab function to construct Huffman dictionary with following arguments:

```
dict = huffman_dict(symbols, prob)
```

where `symbols` is a vector consisting of the names of the symbols. The second argument `prob` is a vector containing the probabilities of these symbols. The output `dict` is a ***cell array*** with the following specifications

- Each row in `dict` corresponds to a node in the Huffman tree. The index of this node is row number.
- The first column of `dict` is the name of a node in the Huffman tree.
- The second column of `dict` corresponds to its probability.
- The third and the fourth columns of `dict` are the **indices** for the left child and the right child of this node.
- The fifth column of `dict` has the bits.

For the Huffman tree example given in class, the following sample codes are for your reference.

```
>> symbols = {   's0',   's1',   's2',   's3',   's4'   };
>> prob    = [   0.26,   0.25,   0.20,   0.15,   0.14   ];
>> dict = huffman_dict( symbols, prob )

dict =

```

Table 1: A table of symbols in $\mathcal{X}$ and their probabilities.

Symbol	Probability
a	0.28
b	0.21
c	0.2
d	0.12
e	0.075
f	0.06
g	0.05
h	0.005

```
7    9×5 cell array
8
9      {'s0'        }    {[0.2600]}    {0×0 double}    {0×0 double}    {'01'    }
10     {'s1'        }    {[0.2500]}    {0×0 double}    {0×0 double}    {'10'    }
11     {'s2'        }    {[0.2000]}    {0×0 double}    {0×0 double}    {'11'    }
12     {'s3'        }    {[0.1500]}    {0×0 double}    {0×0 double}    {'000'   }
13     {'s4'        }    {[0.1400]}    {0×0 double}    {0×0 double}    {'001'   }
14     {'s3s4'      }    {[0.2900]}    {[       4]}    {[       5]}    {'00'    }
15     {'s1s2'      }    {[0.4500]}    {[       2]}    {[       3]}    {'1'     }
16     {'s3s4s0'    }    {[0.5500]}    {[       6]}    {[       1]}    {'0'     }
17     {'s3s4s0s1s2'}    {[     1]}    {[       8]}    {[       7]}    {0×0 char}
```

For example, the 8th row of `dict` (Line 16 in the above block) represents a node with name `'s3s4s0'`, probability 0.55, left child with index 6 (node name `'s3s4'`), right child with index 1 (node name `'s0'`), and the bits `'0'`.

Print the Hufman dictionary `dict` associated with the symbols and the probabilities in Table 1.

(b) **(10 points)** Implement a Huffman encoder function which has the following arguments:

```
bin_seq = huffman_enc(sym_seq, dict);
```

where `sym_seq` is a sequence of symbols and `dict` is a Huffman dictionary with the same format in Problem 2a.

Verify your results with Problem 1e.

(c) **(10 points)** Implement a Huffman decoder function which has the following arguments:

```
sym_seq = huffman_dec(bin_seq, dict);
```

where `bin_seq` is a sequence of bits after Huffman encoding, and `sym_seq` represents the decoded symbols.

Verify your results with Problem 1f.

Note: In addition to the verification in Problems 2a, 2b, and 2c, **we will verify your implementation with extra test cases, which are not available to you**.

3. **(30 points) The average codeword length of Huffman coding:** In this problem, we study the average codeword length of Huffman codes through Monte-Carlo simulation. We consider the alphabet $\mathcal{X} = \{a, b, \ldots, h\}$ in Table 1.

***Warning*: We suggest that you use `huffman_dict` and `huffman_enc` in Problem 2. If you use the built-in** Matlab **functions `huffmandict` and `huffmanenco` for Huffman coding, then the grade becomes $\times 50\%$ for Problem 3.**

(a) **(5 points)** Generate a sequence of $n = 10$ symbols according to the probability in Table 1. List the symbol sequence and Huffman-encoded binary data, and find the length of the data (in bits) in your report.

(b) (**5 points**) Repeat the experiment in Problem 3a multiple times (say $R = 200$ times). Each experiment randomly generates a sequence of symbols. Assume that the length of the encoded binary data in the r-th experiment is $L_n^{(r)}$. **Plot the histogram** of $L_n^{(1)}, L_n^{(2)}, \ldots, L_n^{(R)}$ and **indicate the mean** (denoted by $L_n(R)$) in the title according to

$$L_n(R) := \frac{1}{R} \sum_{r=1}^{R} L_n^{(r)}. \tag{2}$$

You may use the Matlab command `histogram` for this problem. The parameter R is also known as the number of Monte-Carlo simulations.

(c) (**10 points**) Based on R experiments, the average codeword length $\overline{L}_n(R)$ is defined as

$$\overline{L}_n(R) := \frac{1}{n} L_n(R). \tag{3}$$

Next we generate a **plot with the following specifications**.

- The horizontal axis corresponds to the number of Monte-Carlo runs with

  $$R = 10,\ 20,\ 50,\ 100,\ 200,\ 500,\ 1000,$$

  in the logarithm scale.
- Two horizontal lines show the entropy $H[X]$ and the average codeword length $\overline{L}$ in Problems 1a and 1d.
- Three curves illustrate the *experimental* average codeword length $\overline{L}_n(R)$ for $n = 10$, 50, and 100.
- You can use Matlab commands `semilogx`, `xlabel`, `ylabel`, `title`, `grid`, `legend` for your plot. We suggest that you present these five curves with proper markers, line styles, and line colors.

(d) (**10 points**) Comment on the plot in Problem 3c as much as possible.

4. (**Bonus 15 points**) Consider the *identical and independently-distributed* extension of the symbols X to $X_1 \times X_2 \times X_3$. Now do Problems 2 and 3 again 😱. Note that you have to normalize the average length $\bar{L}$ and the entropy $H[X_1X_2X_3]$ by factor 3. What do you observe?

3 Lab Report

There is no format/typesetting requirements for your lab report, but you have to make your report decent and looking nice. In the report, you should address the results of the exercises mentioned above. You should also include your simulation program in the appendix of the report. Include whatever discussions about the new findings during the lab exercise, or the problems encountered and how are those solved. Please properly cite the literature if you referred to. Do not limit yourself to the exercises specified here. You are highly encouraged to play around with your simulation program on self-initiated extra lab exercises/discussions.

It is highly recommended to use Matlab to implement the Huffman coding. However, you are also free to use other software programming languages. For more references and instructions of Matlab, there are a bunch of online references such as:

- https://www.mccormick.northwestern.edu/documents/students/undergraduate/introduction-to-matlab.pdf
- https://blogs.mathworks.com/loren/2007/12/11/making-pretty-graphs/
- https://blogs.mathworks.com/loren/2007/12/11/making-pretty-graphs/
- https://web.eecs.umich.edu/~aey/eecs451/matlab.pdf
- http://staff.www.ltu.se/~damvar/matlab.html