2 Source Coding

In this chapter, we look at the "source encoder" part of the system. This part removes redundancy from the message stream or sequence. We will focus only on binary source coding.

2.1. The material in this chapter is based on [C & T Ch 2, 4, and 5].

2.1 General Concepts

Example 2.2. Suppose your message is a paragraph of (written natural) text in English.

- Approximately 100 possibilities for characters (symbols).
    - For example, a character-encoding scheme called **ASCII** (American Standard Code for Information Interchange) originally[1] had 128 specified characters – the numbers 0–9, the letters a–z and A–Z, some basic punctuation symbols[2], and a blank space.
- Do we need 7 bits per character?

2.3. A sentence of English–or of any other language–always has more information than you need to decipher it. The meaning of a message can remain unchanged even though parts of it are removed.

Example 2.4.

- "J-st tr- t- r–d th-s s-nt-nc-." [3]
- "Thanks to the redundancy of language, yxx cxn xndxrstxnd whxt x xm wrxtxng xvxn xf x rxplxcx xll thx vxwxls wxth xn 'x' (t gts lttl hrdr f y dn't vn kn whr th vwls r)." [4]

[1] Being American, it didn't originally support accented letters, nor any currency symbols other than the dollar. More advanced Unicode system was established in 1991.

[2] There are also some control codes that originated with Teletype machines. In fact, among the 128 characters, 33 are non-printing control characters (many now obsolete) that affect how text and space are processed and 95 printable characters, including the space.

[3] Charles Seife, Decoding the Universe. Penguin, 2007

[4] Steven Pinker, The Language Instinct: How the Mind Creates Language. William Morrow, 1994

2.5. It is estimated that we may only need about 1 bits per character in English text.

Definition 2.6. Discrete Memoryless Sources (DMS): Let us be more specific about the information source.

- The message that the information source produces can be represented by a **vector** of symbols (characters) $X_1, X_2, \ldots, X_n$.
  - A perpetual message source would produce a never-ending **sequence** of symbols $X_1, X_2, \ldots$.
- These X_k's are random variables (at least from the perspective of the decoder; otherwise, these is no need for communication).
- For simplicity, we will assume our source to be discrete and memoryless.
  - Assuming a **discrete** source means that the random variables are all discrete; that is, they have supports which are countable.
    - Recall that "countable" means "finite" or "countably infinite".
    - We will further assume that they all share the same support and that the support is finite.
      - This support is called the **source alphabet**.
      - See Example 2.7 for some examples.
  - Assuming a **memoryless** source means that there is no dependency among the symbols in the sequence.
    - More specifically,

$$p_{X_1,X_2,\ldots,X_n}(x_1, x_2, \ldots, x_n) = p_{X_1}(x_1) \times p_{X_2}(x_2) \times \cdots \times p_{X_n}(x_n). \tag{1}$$

    - Practical sources would *not* be memoryless; there are some amount of dependence (structure) among the symbols. For English text, this is demonstrated in Example 2.4.
      - Simple DMS model provides a good starting point to study.
      - We can take advantage of such dependency.

* We will further assume that all of the random variables share the same probability mass function (pmf)[5]. We denote this shared pmf by $p_X(x)$.

  In which case, (1) becomes

$$p_{X_1,X_2,\ldots,X_n}(x_1, x_2, \ldots, x_n) = p_X(x_1) \times p_X(x_2) \times \cdots \times p_X(x_n). \quad (2)$$

  - We will also assume that the pmf $p_X(x)$ is known. In practice, there is an extra step of estimating this $p_X(x)$.
  - To save space, we may see the pmf $p_X(x)$ written simply as $p(x)$, i.e. without the subscript part.

* The shared support of X which is usually denoted by S_X becomes the source alphabet. Note that we also often see the use of $\mathcal{X}$ to denote the support of X.

- Summary: A DMS produces a sequence (symbol by symbol) of i.i.d. RVs $X_1, X_2, \ldots$ all of which share the same pmf $p_X(x)$ whose support is called the source alphabet.

- Because our simplified source code can be characterized by a random variable X, we only need to specify its pmf $p_X(x)$.

Example 2.7. Examples of (finite) source alphabets

(a) Collection of 95 symbols for English text.

(b) Collection of 128 symbols for string of ASCII symbols.

(c) Collection of four symbols {Yes, No, OK, Thank You} for crude conversation with Farang.

(d) Collection of four symbols {A, B, C, D} for answers of multiple-choice test.

[5]We often use the term "distribution" interchangably with pmf and pdf; that is, instead of saying "pmf of X", we may say "distribution of X".

Definition 2.8. An **encoder** $c(\cdot)$ is a function that maps each of the symbol in the source alphabet into a (binary) codeword.

- The codeword corresponding to a source symbol x is denoted by $c(x)$.

- Each codeword is constructed from a **code alphabet**.
  - A binary codeword is constructed from a two-symbol alphabet, wherein the two symbols are usually taken as 0 and 1.
  - It is possible to consider non-binary codeword. Morse code discussed in Example 2.13 is one such example.
- Mathematically, we write

  $$\text{Encoder } c : S_X \to \{0,1\}^*$$

  where

  $$\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\}$$

  is the set of all finite-length binary strings.

- The length of the codeword associated with source symbol x is denoted by $\ell(x)$.
  - In fact, writing this as $\ell\left(c(x)\right)$ may be clearer because we can see that the length depends on the choice of the encoder. However, we shall follow the notation above[6].

Example 2.9. $c(\text{red}) = 00$, $c(\text{blue}) = 11$ is an example of a source code for the source alphabet $S_X = \{\text{red}, \text{blue}\}$.

[6] which is used by the standard textbooks in information theory.

Example 2.10. Suppose the message is a sequence of basic English words which happen according to the probabilities provided in the table below.

x	$p(x)$	Codeword $c(x)$	$\ell(x)$
Yes	4%		
No	3%		
OK	90%		
Thank You	3%		

Definition 2.11. The **expected length** of a code $c(\cdot)$ for (a DMS source which is characterized by) a random variable X with probability mass function $p_X(x)$ is given by

$$\mathbb{E}\left[\ell(X)\right] = \sum_{x \in S_X} p_X(x)\ell(x).$$

Example 2.12. Back to Example 2.10. Consider a new encoder:

x	$p(x)$	Codeword $c(x)$	$\ell(x)$
Yes	4%	01	
No	3%	001	
OK	90%	1	
Thank You	3%	0001	

Observe the following:

- Data compression can be achieved by assigning short descriptions to the most frequent outcomes of the data source, and necessarily longer descriptions to the less frequent outcomes.
- When we calculate the expected length, we don't really use the fact that the source alphabet is the set $\{\text{Yes}, \text{No}, \text{OK}, \text{Thank You}\}$. We would get the same answer if it is replaced by the set $\{1, 2, 3, 4\}$, or the set $\{a, b, c, d\}$. All that matters is that the alphabet size is 4, and the corresponding probabilities are $\{0.04, 0.03, 0.9, 0.03\}$.

  Therefore, for brevity, we often find DMS source defined only by its alphabet size and the list of probabilities.

Example 2.13. The Morse code is a reasonably efficient code for the English alphabet using an alphabet of four symbols: a dot, a dash, a letter space, and a word space. [See Slides]

- Short sequences represent frequent letters (e.g., a single dot represents E) and long sequences represent infrequent letters (e.g., Q is represented by "dash,dash,dot,dash").

Example 2.14. Thought experiment: Let's consider the following code

x	$p(x)$	Codeword $c(x)$	$\ell(x)$
1	4%	0	
2	3%	1	
3	90%	0	
4	3%	1	

This code is bad because we have ambiguity at the decoder. When a codeword "0" is received, we don't know whether to decode it as the source symbol "1" or the source symbol "3". If we want to have lossless source coding, this ambiguity is not allowed.

Definition 2.15. A code is **nonsingular** if every source symbol in the source alphabet has different codeword.

As seen from Example 2.14, nonsingularity is an important concept. However, it turns out that this property is not enough.

Example 2.16. Another thought experiment: Let's consider the following code

x	$p(x)$	Codeword $c(x)$	$\ell(x)$
1	4%	01	
2	3%	010	
3	90%	0	
4	3%	10	

2.17. We usually wish to convey a sequence (string) of source symbols. So, we will need to consider **concatenation** of codewords; that is, if our source string is

$$X_1, X_2, X_3, \ldots$$

then the corresponding encoded string is

$$c(X_1)c(X_2)c(X_3)\cdots .$$

In such cases, to ensure decodability, we may

(a) use fixed-length code (as in Example 2.10), or

(b) use variable-length code and

(i) add a special symbol (a "comma" or a "space") between any two codewords

or

(ii) use uniquely decodable codes.

Definition 2.18. A code is called **uniquely decodable** (UD) if any encoded string has only one possible source string producing it.

Example 2.19. The code used in Example 2.16 is not uniquely decodable because source string "2", source string "34", and source string "13" share the same code string "010".

2.20. It may not be easy to check unique decodability of a code. (See Example 2.28.) Also, even when a code is uniquely decodable, one may have to look at the entire string to determine even the first symbol in the corresponding source string. Therefore, we focus on a subset of uniquely decodable codes called prefix code.

Definition 2.21. A code is called a **prefix code** if no codeword is a prefix[7] of any other codeword.

- Equivalently, a code is called a **prefix code** if you can put all the codewords into a binary tree where all of them are leaves.
- A more appropriate name would be "**prefix-free**" code.
- The codeword corresponding to a symbol is the string of labels on the path from the root to the corresponding leaf.

Example 2.22.

x	Codeword $c(x)$
1	10
2	110
3	0
4	111

[7]String s_1 is a prefix of string s_2 if there exist a string s_3, possibly empty, such that $s_2 = s_1 s_3$.

Example 2.23. The code used in Example 2.12 is a prefix code.

x	Codeword $c(x)$
1	01
2	001
3	1
4	0001

2.24. Any prefix code is uniquely decodable.

- The end of a codeword is immediately recognizable.
- Each source symbol can be decoded as soon as we come to the end of the codeword corresponding to it. In particular, we need not wait to see the codewords that come later.
- Therefore, another name for "prefix code" is **instantaneous code**.

Example 2.25. The codes used in Example 2.12 (Example 2.23) and Example 2.22 are prefix codes and hence they are uniquely decodable.

2.26. The nesting relationship among all the types of source codes is shown in Figure 2.

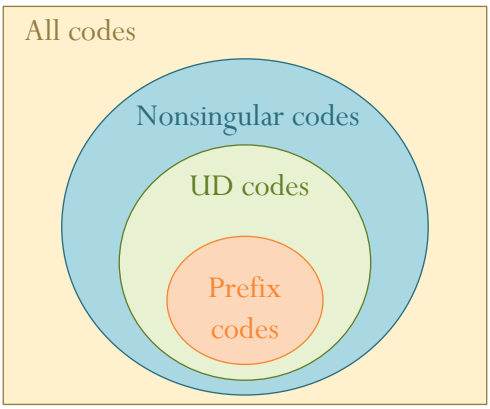

Figure 2: Classes of codes

Example 2.27.

x	Codeword $c(x)$
1	1
2	10
3	100
4	1000

Try to decode 100100011101001 11

Example 2.28. [5, p 106–107]

x	Codeword $c(x)$
1	10
2	00
3	11
4	110

This code is not a prefix code because codeword "11" is a prefix of codeword "110".

This code is uniquely decodable. To see that it is uniquely decodable, take any code string and start from the beginning.

- If the first two bits are 00 or 10, they can be decoded immediately.
- If the first two bits are 11, we must look at the following bit(s).
    - If the next bit is a 1, the first source symbol is a 3.
    - If the next bit is a 0, we need to count how many 0s are there before 1 shows up again.
    - If the length of the string of 0's immediately following the 11 is even, the first source symbol is a 3.
    - If the length of the string of 0's immediately following the 11 is odd, the first codeword must be 110 and the first source symbol must be 4.

By repeating this argument, we can see that this code is uniquely decodable.