Information Theory (5XSE0)
Ch.2: Data Compression

Hamdi Joudeh

TU/e (Q3 2020-2021)

Reading

- Cover & Thomas, Ch. 5 (excluding 5.8, 5.9, 5.10).
- Supplementary: Gallager, Ch. 3 (3.2, 3.3, 3.4); MacKay, Ch. 5.

Before we start

- Recall change of base for entropy: $H_2(X) = (\log_2 D) H_D(X)$.

Consider the problem of representing a random variable X in bits. Each element x in $\mathcal{X}$ is assigned a unique binary string $C(x)$, known as a codeword. The code should be uniquely decodable: upon receiving a stream of bits $C(x_1)C(x_2)\cdots C(x_j)$ (with no punctuation), we should be able to retrieve $x_1, x_2, \ldots, x_j$. We wish to make $C(X)$ as short as possible on average. How short can it be?

1 Source Codes

Suppose that we have a random variable X that takes values over the finite alphabet $\mathcal{X}$. We wish to find a description of X in terms of a D-ary *code alphabet* $\mathcal{D} = \{0, 1, \ldots, D-1\}$. Such a description is known as a *source code*. It is often the case that a source code maps each value in $\mathcal{X}$ to a string of symbols from $\mathcal{D}$. An example is the ASCII code, which assigns a binary string to each character. The set of finite-length strings of symbols from $\mathcal{D}$ is denoted by $\mathcal{D}^*$, which is referred to as the extension of $\mathcal{D}$.

Definition 1. A D-ary *source code* C of a random variable X is a mapping from $\mathcal{X}$ to $\mathcal{D}^*$. The string assigned to x is denoted by $C(x)$, and is known as a *codeword*. The length of $C(x)$ in D-ary symbols is denoted by $l(x)$. We occasionally use the notation C_i and l_i to denote codeword i and its length.

Example 1. Consider $\mathcal{X} = \{0, 1, 2, 3\}$ and $\mathcal{D} = \{0, 1\}$. A source code that maps each element in $\mathcal{X}$ to its binary representation is given by: $C(0) = 00$, $C(1) = 01$, $C(2) = 10$, $C(3) = 11$. More generally, for any $\mathcal{X}$, elements can be uniquely represented using binary codewords of length no more than $\lceil \log |\mathcal{X}| \rceil$, i.e. X can be represented using $\lceil \log |\mathcal{X}| \rceil$ bits on average. This is sometimes called the *raw bit* content.

Example 2. $C(\mathsf{r}) = 0$, $C(\mathsf{g}) = 10$, $C(\mathsf{b}) = 11$, where $\mathcal{X} = \{\mathsf{r}, \mathsf{g}, \mathsf{b}\}$ and $\mathcal{D} = \{0, 1\}$. Note that in this example, not all codewords have the same length.

Definition 2. The *expected length* $L(C)$ of a code C for a random variable X with pmf $p(x)$ is given by

$$L(C) \equiv \sum_{x \in \mathcal{X}} p(x) l(x) = \mathbb{E} l(X).$$

1.1 Non-singular, uniquely decodable and prefix codes

The goal of this chapter is to find a description of X, i.e. a source code, that has a minimal expected length. In principle, we can map all elements in $\mathcal{X}$ to a single digit and achieve an expected length of 1. This code, however, is useless due to its singularity. Therefore, our code design is not without constraints: we are interested in a family of codes where such confusions do not arise (lossless codes).

Definition 3. A code is *non-singular* if every element in $\mathcal{X}$ maps into a different string in $\mathcal{D}^*$, that is

$$x \neq x' \implies C(x) \neq C(x').$$

Non-singularity is sufficient to describe a single random variable X with no ambiguity. However, imagine the case where we wish to describe a sequence of random variables $X_1, X_2, \ldots, X_n$, drawn from the alphabet $\mathcal{X}^n$. This scenario is very common in practice: X may describe a character, while we often send or store a text which can be described by a sequence of characters $X_1, X_2, \ldots, X_n$.

Definition 4. The *extension* C^* of a source code C is a mapping from $\mathcal{X}^*$ to $\mathcal{D}^*$ such that

$$C^*(x_1 x_2 \cdots x_n) = C(x_1)C(x_2)\cdots C(x_n).$$

We will often drop the asterisk and denote an extension of codewords by $C(x_1 x_2 \cdots x_n)$.

Definition 5. A code C is called *uniquely decodable* if its extension C^* is non-singular.

A uniquely decodable code guarantees that a stream of codewords $C(x_1)C(x_2)\cdots C(x_n)$ can be mapped uniquely to the source sequence $x_1, x_2, \ldots, x_n$. However, we may have to observe the entire stream $C(x_1 x_2 \cdots x_n)$ before we are able to make a decision. This delay is mitigated by prefix codes.

Definition 6. A code C is called a *prefix code* if no codeword is a prefix of any other codeword. That is, if $C(x)$ is a codeword, then any string starting with $C(x)$ cannot be a codeword.

An example of different codes is shown in the table below.

X	singular	non-singular	uniquely decodable	prefix
1	0	0	10	0
2	0	1	00	10
3	0	10	11	110
4	0	01	110	111

The non-singular code in the above table is not uniquely decodable. For instance, the source sequences $(2,1,1,2)$, $(3,4)$, $(2,1,4)$, $(3,1,2)$ all map to same code sequence 1001, and hence observing 1001 will cause an ambiguity in decoding. The code in the fourth column is uniquely decodable: we can always retrieve the correct source sequence from any code sequence (check this!). However, it is not a prefix code as the third codeword 11 is a prefix of the fourth codeword 110. As a result, if we observe 11, we will not be able to decode this string immediately and we will have to look at the following bits. If 11 is followed by an odd number of 0's, then this 11 is part of the codeword 110 which corresponds to the symbol 4. However, if the length of the string of 0's following 11 is even (including no 0's), then we know that 11 corresponds to 3. Therefore, decoding some codewords cannot be done instantaneously and will depend on observing the following codeword(s). This is not the case for prefix codes, as seen in the fifth column. Here no codeword is a prefix of another, and hence as soon as we recieve a codeword, we can decode it with no ambiguity and without waiting for the following codeword (try this!). This is why prefix codes are sometimes called *instantaneous* codes (or *self-punctuating* codes).

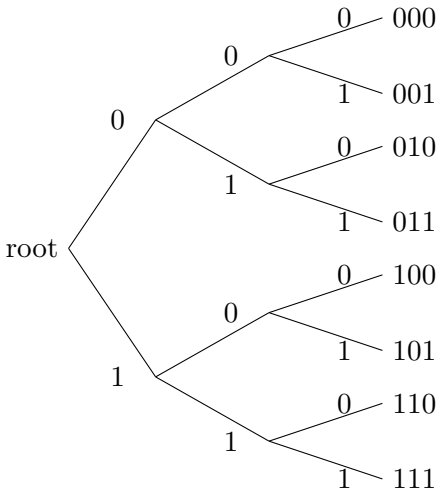
Fixing a code alphabet D, we have the following relationship between classes of source codes:

$$\{\text{Prefix codes}\} \subset \{\text{Uniquely decodable codes}\} \subset \{\text{Non-singular codes}\} \subset \{\text{All codes}\}.$$

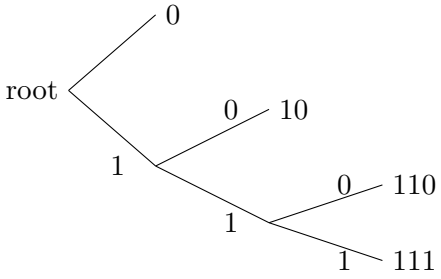
That is, prefix codes are uniquely decodable, and uniquely decodable codes are non-singular (verify this!).

1.2 Code tree

It is useful to represent codes using trees. A D-ary tree of depth $l_{\max}$ is constructed as follows. We start with a root node at level (or depth) zero and branch out into D *children* nodes at level 1, and we assign a distinct symbol from the code alphabet $\mathcal{D}$ to each of the D branches (or edges). From each child node at level 1, we branch out into D children nodes at level 2, and again we assign distinct code symbols to branches stemming out from the same node. This is repeated until we reach nodes at level $l_{\max}$. Each node at level $l \leq l_{\max}$ has D^{a-l} *descendants* at level $a \geq l$, which follows from the fact that each child node will have D children of its own, and so on. We allow $a = l$ in the above statement, i.e. a node is considered to be the only descendent of itself at its own level $a = l$. A binary code tree is shown below.



Each branch in the tree represents a D-ary symbol, and each node represents a D-ary string formed by tracing out the path of branches leading to it from the root node. A node is hence a possible codeword in a D-ary code, with length equal to the node's depth. For a node to be a codeword in a prefix code, its descendants cannot be codewords. For example, in a binary tree, picking 0 to be a codeword in a prefix code eliminates half the binary strings of lengths greater than one from being potential codewords. The binary code tree for the prefix code in the previous table is shown below.



One may notice from the above prefix code tree that making some codewords short in a prefix code necessarily requires making other codewords long: there is a limited supply of short codewords which are not a prefix for other codewords. This fact is quantified by Kraft inequality.

2 Kraft Inequality

In a prefix code, not all codewords can be made short. This is captured by the following inequality.

Theorem 1. (Kraft inequality). *For any D-ary prefix code with m codewords, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy the inequality*

$$\sum_{i=1}^{m} D^{-l_i} \leq 1. \tag{1}$$

Moreover, for any set of integers $l_1, l_2, \ldots, l_m$ that satisfy the inequality (1), there exists a D-ary prefix code consisting of m codewords with $l_1, l_2, \ldots, l_m$ as their lengths.

Intuition. Think of the right-hand-side of (1) as the total allowed budget for codewords in a prefix code, and each term D^{-l_i} on the left-hand-side as the cost of selecting a codeword of length l_i. A short codeword has a high cost which depletes a large portion of the total budget, while a long codeword has a lower cost. For instance, an extremely short codeword of length 1 in a binary code consumes half the budget, leaving us with no choice but to select longer codewords if we wish to have two or more additional codewords.

Proof. Suppose that we have a prefix code with codewords $C_1, C_2, \ldots, C_m$ of lengths $l_1, l_2, \ldots, l_m$, respectively. We define $l_{\max}$ to be the maximum of such lengths. As we saw earlier, each codeword C_i of length l_i is represented by a node at level l_i on a D-ary tree. In what follows, C_i will represent both the codeword and its corresponding node on the D-ary tree. Recall that in a prefix code, each codeword eliminates all its descendants on the D-ary tree from being possible codewords.

Now let us focus on nodes at the deepest level $l_{\max}$. We have a total of $D^{l_{max}}$ nodes at this level, and each node is either a descendent of a codeword of length less than $l_{\max}$, a codeword of length $l_{\max}$, or neither. For any codeword C_i, we define $\mathcal{D}_i$ to be the set of its descendants at the deepest level $l_{\max}$. Note that if $l_i = l_{\max}$, then $\mathcal{D}_i$ is the codeword C_i itself. As codewords are distinct and no codeword is a descendent of another, the sets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m$ are disjoint. Moreover, the union of these sets is a subset of all nodes at level $l_{\max}$, as some nodes may be neither a codeword nor a descendent of one. From these observations, we have

$$\sum_{i=1}^{m} |\mathcal{D}_i| \leq D^{l_{max}}. \tag{2}$$

We know from earlier that each codeword C_i has D^{a-l_i} descendants at level $a \geq l$. Therefore, we have $|\mathcal{D}_i| = D^{l_{\max}-l_i}$, which is equal to 1 whenever $l_i = l_{\max}$. Plugging the values of $|\mathcal{D}_i|$ into (2), we obtain

$$\sum_{i=1}^{m} D^{l_{\max}-l_i} \leq D^{l_{max}} \implies \sum_{i=1}^{m} D^{-l_i} \leq 1$$

which proves the inequality in (1).

To complete the proof of Theorem 1, it remains to show that given a set of lengths $l_1, l_2, \ldots, l_m$ that satisfy (1), we can find a set of D-ary codewords $C_1, C_2, \ldots, C_m$ with these lengths such that no codeword is a prefix of the other, or equivalently, nodes at levels $l_1, l_2, \ldots, l_m$ such that no node is a descendent of the other. For this, we assume without loss of generality that

$$l_1 \leq l_2 \leq \cdots \leq l_m = l_{\max}. \tag{3}$$

Staring with the shallowest level l_1, we pick a node at this level and make it the first codeword C_1. This will automatically eliminate the descendants of C_1 at level l_2 from being possible codewords. We have $D^{l_2-l_1}$ descendants of C_1 at level l_2, constituting a fraction $D^{-l_1} < 1$ of all nodes at this level. Recall that if $l_2 = l_1$, C_1 is considered to be its own descendant, hence eliminating itself from level $l_2 = l_1$.

Moving on to level l_2, we pick a node from the remaining fraction $1 - D^{-l_1} > 0$ of nodes and make it the second codeword C_2. Codewords C_1 and C_2 eliminate their descendants at level l_3 from being possible codewords, which together constitute a fraction $D^{-l_1} + D^{-l_2} < 1$ of nodes at this level. This fraction is strictly less than 1 because $D^{-l_1} + D^{-l_2} + \cdots + D^{-l_m} \leq 1$ holds. Therefore, we have a nonzero fraction $1 - (D^{-l_1} + D^{-l_2}) > 0$ of possible codeword nodes at level l_3, from which we pick the third codeword C_3.

We proceed in a similar fashion while noting that in each iteration $i < m$, after selecting a codeword C_i at level l_i, we are always left with a nonzero fraction $1 - (D^{-l_1} + D^{-l_2} + \cdots + D^{-l_i}) > 0$ of possible codeword nodes at level l_{i+1}. This always enables us to select a codeword C_{i+1} in the following iteration, hence guaranteeing that we could construct a prefix code with codeword lengths that satisfy (1). □

It is worthwhile noting that Theorem 1 does not say that any code whose lengths satisfy the Kraft inequality is a prefix code. For example, the set of binary codewords $\{0, 00, 11\}$ satisfies (1), but is not a prefix code. Theorem 1 says that *some* prefix code exists with those lengths, e.g. $\{0, 10, 11\}$.

2.1 Kraft inequality for uniquely decodable codes

We know that the class of prefix codes is included in the class of uniquely decodable codes, i.e. a prefix code is uniquely decodable but a uniquely decodable code is not necessarily a prefix code. This suggests that we may be able to find uniquely decodable codes with shorter codeword lengths compared to prefix codes, which would ultimately lead to shorter descriptions of random variables. Surprisingly, this turns out to be not the case: the class of uniquely decodable codes does not offer any further possibilities to reducing codeword lengths compared to the class of prefix codes, as shown in the following theorem.

Theorem 2. (McMillan). *For any D-ary uniquely decodable code with m codewords, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy the Kraft inequality*

$$\sum_{i=1}^{m} D^{-l_i} \leq 1. \tag{4}$$

Moreover, for any set of integers $l_1, l_2, \ldots, l_m$ that satisfy the Kraft inequality, there exists a D-ary uniquely decodable code consisting of m codewords with $l_1, l_2, \ldots, l_m$ as their lengths.

Intuition. Theorem 2 states that the Kraft inequality characterizes the set of admissible lengths for uniquely decodable codes. This is the exact same set of admissible lengths for prefix codes, as shown in Theorem 1. Note that this does not mean that the *class* of uniquely decodable codes is equal to the *class* of prefix codes. What it says is that given a uniquely decodable code with codeword lengths $l_1, l_2, \ldots, l_m$, we can always find a prefix code with the same codeword lengths. Therefore, we may just focus on prefix codes from now on and safely forget about uniquely decodable codes (after proving the theorem!).

Proof. The second part of the theorem follows from the proof of Theorem 1. In particular, given a set lengths $l_1, l_2, \ldots, l_m$ that satisfy the Kraft inequality, we can construct a prefix code, which is also a uniquely decodable code. Therefore, we focus on proving the first part of Theorem 2.

Let C be a uniquely decodable code that maps every symbol $x \in \mathcal{X}$ to a D-ary string $C(x)$ of length $l(x)$. The k-th extension C^k maps every sequence of symbols $(x_1, x_2, \ldots, x_k) \in \mathcal{X}^k$ to an extended codeword $C^k(x_1, x_2, \ldots, x_k) = C(x_1)C(x_2)\cdots C(x_k)$ of length

$$l(x_1, x_2, \ldots, x_k) = \sum_{i=1}^{k} l(x_i)$$

Recall from Definition 5 that the extension C^k of a uniquely decodable code is non-singular (for any k). We wish to show that $\sum_{x\in\mathcal{X}} D^{-l(x)} \leq 1$. The trick is to take the k-th power of $\sum_{x\in\mathcal{X}} D^{-l(x)}$ as follows:

$$\begin{aligned}
\left(\sum_{x\in\mathcal{X}} D^{-l(x)}\right)^k &= \left(\sum_{x_1\in\mathcal{X}} D^{-l(x_1)}\right)\left(\sum_{x_2\in\mathcal{X}} D^{-l(x_2)}\right)\cdots\left(\sum_{x_k\in\mathcal{X}} D^{-l(x_k)}\right) \\
&= \sum_{x_1\in\mathcal{X}}\sum_{x_2\in\mathcal{X}}\cdots\sum_{x_k\in\mathcal{X}} D^{-l(x_1)}D^{-l(x_2)}\cdots D^{-l(x_k)} \\
&= \sum_{x^k\in\mathcal{X}^k} D^{-l(x^k)}
\end{aligned} \tag{5}$$

where in the above, x^k is a shorthand notation for the sequence $(x_1, x_2, \ldots, x_k)$. Let $a(n)$ be the number of extended codewords of length n, where n takes values between 1 to $kl_{\max}$. Then we may rewrite (5) as

$$\left(\sum_{x\in\mathcal{X}} D^{-l(x)}\right)^k = \sum_{x^k\in\mathcal{X}^k} D^{-l(x^k)} = \sum_{n=1}^{kl_{\max}} a(n)D^{-n}. \tag{6}$$

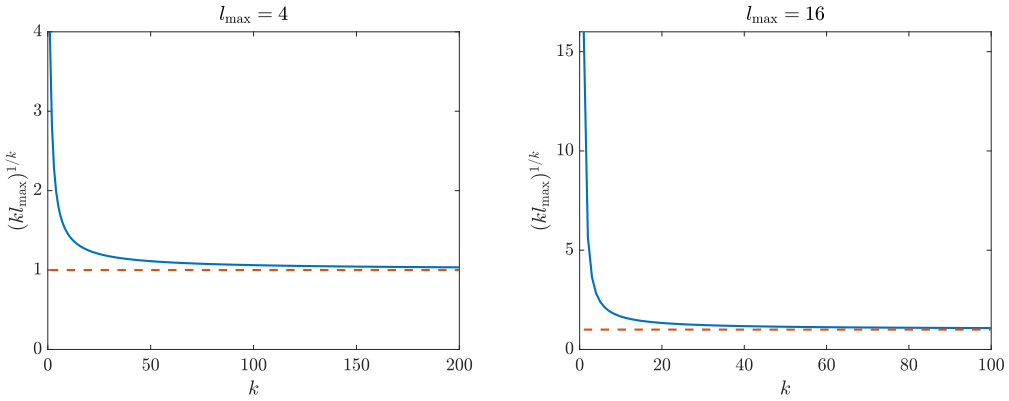
A simple upper bound for $a(n)$ is given by $a(n) \leq D^n$, as there are at most D^n unique code sequences of length n (no repetitions are allowed). Plugging this upper bound into (6), we obtain

$$\left(\sum_{x\in\mathcal{X}} D^{-l(x)}\right)^k \leq \sum_{n=1}^{kl_{\max}} D^n D^{-n} = kl_{\max}. \tag{7}$$

By taking the k-th root of both sides, we have

$$\sum_{x\in\mathcal{X}} D^{-l(x)} \leq (kl_{\max})^{1/k} \tag{8}$$

which must hold for all admissible values of k (integers no less than 1). Note that for each k, we obtain a different upper bound for $\sum_{x\in\mathcal{X}} D^{-l(x)}$ in (8). The tightest upper bound is obtained by choosing k such that $f(k) = (kl_{\max})^{1/k}$ is minimized. For any positive $l_{\max}$, the function $f(k)$ is monotonically decreasing in k and has a limit of 1 (verify this!). Examples are shown in the figures below.



Combining this property of $f(k)$ with (8), we obtain

$$\sum_{x\in\mathcal{X}} D^{-l(x)} \leq \lim_{k\to\infty} (kl_{\max})^{1/k} = 1 \tag{9}$$

which completes the proof of Theorem 2. □

Exercise 1. Show that $\lim_{k\to\infty}(kl_{\max})^{1/k} = 1$.

3 Optimal Prefix Codes

The Kraft inequality in Theorem 1 gives us a necessary and sufficient conditions on the codeword lengths for a prefix code: such code exists if and only if $\boldsymbol{l} \equiv (l_1, l_2, \ldots, l_m)$ satisfies (1). This reduces the problem of designing a prefix code to a problem of finding adequate sequence of lengths $\boldsymbol{l}$. Once we have found the *right* lengths, we can construct a corresponding prefix code by following the procedure described in the proof of Theorem 1. The question hence becomes: what are the *right* lengths?

This question is answered by relating the problem of designing a code C to the problem of describing a random variable X, as mentioned at the beginning of the chapter The *right* codeword lengths of C are those that yield a minimum expected description length for X. To make this statement more concrete, let us define a random variable X over an alphabet $\mathcal{X} = \{1, 2, \ldots, m\}$, and let $p_1, p_2, \ldots, p_m$ be the corresponding probabilities (we use p_x to denote $p(x)$). A D-ary code C for X is specified by its set of codeword lengths $\boldsymbol{l}$, and the expected length of such code as a function of $\boldsymbol{l}$ is given by

$$L(\boldsymbol{l}) = \sum_{i=1}^{m} p_i l_i$$

Finding a prefix code for X with minimum expected length is equivalent to finding an optimal length sequence $\boldsymbol{l}^\star \equiv (l_1^\star, l_2^\star, \ldots, l_m^\star)$ that has the following properties:

- The lengths in $\boldsymbol{l}^\star$ must satisfy the Kraft inequality (prefix code condition).
- We must have $L(\boldsymbol{l}^\star) \leq L(\boldsymbol{l})$ for all $\boldsymbol{l}$ that satisfy the Kraft inequality (shortest expected length).

3.1 Bounds on the expected length of optimal codes

Before delving further into the problem of designing optimal codes, it is useful to identify the target that we wish to hit, i.e. the minimum expected length $L^\star \equiv L(\boldsymbol{l}^\star)$. As we show in the next theorem, it turns out that $L^\star$ is in fact very close to the D-ary entropy $H_D(X) \equiv -\sum_i p_i \log_D p_i$. Note that $H_D(X)$ measures information in "D-its" (or D-ary digits), and can be easily converted back to the entropy in bits as: $H(X) = (\log_2 D) H_D(X)$.

Theorem 3. *The expected length $L^\star$ of an optimal D-ary prefix code for X satisfies:*

$$H_D(X) \leq L^\star < H_D(X) + 1. \tag{10}$$

The above theorem states that *any* D-ary description of X may never be shorter than $H_D(X)$ D-its on average. On the other hand, the optimal description is strictly shorter than $H_D(X)+1$ D-its on average. This theorem establishes the entropy as a fundamental measure for data compression.

In the next two subsections, we will prove the lower and upper bounds in (10) respectively. It will become apparent why the characterization of $L^\star$ is in terms of lower and upper bounds instead of an exact quantity. In short, this is due to the fact that lengths $l_1, l_2, \ldots, l_m$ are integers, which makes it difficult to characterize $L^\star$ exactly in general. Before we proceed, we will also see how to sharpen the characterization in (10) by considering the problem of describing a sequence of i.i.d. random variables.

Instead of describing a single random variable X, consider the problem of designing a source code for the sequence $X^n \equiv (X_1, X_2, \ldots, X_n)$ with n i.i.d. elements (or symbols), each with the same distribution as X. We use $L_n^\star$ to denote the minimum expected code length per source symbol, i.e. an optimal source code for X^n will have an expected length of $nL_n^\star$. By treating the sequence X^n as a single random variable and applying Theorem 3, we obtain the following bounds

$$H_D(X) \leq L_n^\star < H_D(X) + \frac{1}{n}. \tag{11}$$

Therefore as n grows large, i.e. bundling together more i.i.d. source symbols, we approach the ultimate compression rate of $H_D(X)$ D-ary code digits per source symbol.

Exercise 2. Show that (11) holds.

3.1.1 Lower bound on the minimum expected length

We start by finding a lower bound on the minimum expected length in the next theorem . In particular, we show that for any $\boldsymbol{l}$ that satisfies the Kraft inequality, the corresponding expected length $L(\boldsymbol{l})$ can never be less than the D-ary entropy $H_D(X) \equiv -\sum_i p_i \log_D p_i$. Note that $H_D(X)$ measures information in "Dits". Recall that this can be easily converted back to the entropy in bits as: $H(X) = (\log_2 D) H_D(X)$.

Theorem 4. *The expected length for any D-ary prefix code for a random variable X is bounded below as:*

$$L(\boldsymbol{l}) \geq H_D(X)$$

with equality if and only if probabilities are related to lengths as: $p_i = D^{-l_i}$ for all $i \in \{1, 2 \ldots, m\}$.

Proof. We start by defining $r_1, r_2, \ldots, r_m$ such that

$$r_i = \frac{D^{-l_i}}{c} \tag{12}$$

where $c \equiv \sum_{j=1}^{m} D^{-l_j}$. It is clear that (12) is a valid pmf, since $r_i \geq 0$ for all i and $\sum_{i=1}^{m} r_i = 1$. Next, we express the expected length $L(\boldsymbol{l})$ in terms of the pmf in (12) as follows:

$$\begin{aligned}
L(\boldsymbol{l}) &= \sum_{i=1}^{m} p_i l_i \\
&= \sum_{i=1}^{m} p_i \log_D D^{l_i} \\
&= -\sum_{i=1}^{m} p_i \log_D D^{-l_i} \\
&= -\sum_{i=1}^{m} p_i \left(\log_D c + \log_D \frac{D^{-l_i}}{c} \right) \\
&= -\log_D c - \sum_{i=1}^{m} p_i \log_D \frac{D^{-l_i}}{c} \\
&= -\log_D c - \sum_{i=1}^{m} p_i \log_D r_i.
\end{aligned}$$

We we proceed to bound the difference $L(\boldsymbol{l}) - H_D(X)$ as follows:

$$\begin{aligned}
L(\boldsymbol{l}) - H_D(X) &= \log_D \frac{1}{c} + \sum_{i=1}^{m} p_i \log_D \frac{1}{r_i} + \sum_{i=1}^{m} p_i \log_D p_i \\
&= \log_D \frac{1}{c} + \sum_{i=1}^{m} p_i \log_D \frac{p_i}{r_i} \\
&= \log_D \frac{1}{c} + \sum_{i=1}^{m} p_i \log_D \frac{p_i}{r_i} \\
&= \log_D \frac{1}{c} + D_D(p\|r) \\
&\geq 0
\end{aligned} \tag{13}$$

where $D_D(p\|r) \equiv \sum_i p_i \log_D(p_i/r_i)$ is the relative entropy in Dits between the distributions $p_1, p_2, \ldots, p_m$ and $r_1, r_2, \ldots, r_m$. The inequality in (13) holds due to:

- $\log_D(1/c) \geq 0$, which follows from the Kraft inequality $c = \sum_{j=1}^{m} D^{-l_j} \leq 1$.
- $D_D(p\|r) \geq 0$, which is the information inequality.

Note that equality in (13) holds if and only if $\sum_{j=1}^{m} D^{-l_j} = 1$ and $p_i = r_i$ for all i, that is:

$$p_i = r_i = D^{-l_i} \text{ for all } i.$$

For this to hold, $l_i = -\log_D p_i$ must be an integer for all i. This completes the proof. □

Theorem 4 introduces a family of distributions for which there exists D-ary prefix codes with expected lengths equal to the entropy. These are known as D-adic distributions, and are defined as follows.

Definition 7. A probability distribution (or pmf) with mass points $p_1, p_2, \ldots, p_m$ is called D-adic if there exists integers $l_1, l_2, \ldots, l_m$ such that $p_i = D^{-l_i}$ for all $i \in \{1, 2, \ldots, m\}$.

3.1.2 Upper bound on the minimum expected length

From the proof for the lower bound, we have

$$L(\boldsymbol{l}) = H_D(X) + D_D(p\|r) - \log_D c \tag{14}$$
$$\geq H_D(X) \tag{15}$$

where equality holds if and only if $p_i = D^{-l_i}$ or, equivalently, $l_i = -\log_D p_i$. If p_i is not D-adic, then the quantity $l_i = -\log_D p_i$ is not an integer. A straightforward choice of lengths in this case would be to choose l_i as the closest integer which is no smaller than $-\log_D p_i$, that is

$$l_i = \left\lceil \log_D \frac{1}{p_i} \right\rceil. \tag{16}$$

Does the set of length $\boldsymbol{l}$ chosen according to the above give us a valid prefix code? The answer is yes, because these lengths satisfy the Kraft inequality. In particular, we have

$$\begin{aligned}\sum_{i=1}^{m} D^{-l_i} &= \sum_{i=1}^{m} D^{-\lceil \log_D(1/p_i) \rceil} \\ &\leq \sum_{i=1}^{m} D^{-\log_D(1/p_i)} = \sum_{i=1}^{m} p_i = 1.\end{aligned}$$

We call a code with length assignment as in (16) a *roof* code (also called a Shannon code). What is the expected length of such code? since $l_i < \log_D(1/p_i) + 1$, we have

$$L(\boldsymbol{l}) = \sum_{i=1}^{m} p_i l_i < \sum_{i=1}^{m} p_i \log_D(1/p_i) + 1 = H_D(X) + 1.$$

Since an optimal code with lengths $\boldsymbol{l}^\star$ can only be better, in the sense that it must give an expected length $L(\boldsymbol{l}^\star)$ which is no greater than $L(\boldsymbol{l})$, we must have

$$L(\boldsymbol{l}^\star) < H_D(X) + 1 \tag{17}$$

which proves the upper bound in Theorem 3.

3.2 Using the wrong code

Now let's look at the case where we have a random variable $X \sim p(x)$ that we describe using a D-ary prefix code designed for a distribution $q(x)$. This occurs if we, for instance, do not know the *true* distribution $p(x)$, but instead only have access to an *estimate* $q(x)$, which we believe to be the true distribution. Our description of X in this case will not be most efficient due to the mismatch between the true distribution and the estimate. But what is the cost of this mismatch? This will turn out to be $D_D(p\|q)$.

To show this, we consider a prefix code with a length assignment of

$$l(x) = \left\lceil \log_D \frac{1}{q(x)} \right\rceil$$

that is, a roof code. From the Kraft inequality, we know that such code exists. If we use this code to describe X, which has a pmf of $p(x)$, we achieve an expected description length of

$$\mathbb{E}_p\left[l(X)\right] \equiv \sum_{x \in \mathcal{X}} p(x) l(x).$$

The above expected length is bounded as follows.

Theorem 5. *The expected length with respect to a true distribution $p(x)$ of a D-ary prefix code with length assignment $l(x) = \left\lceil \log_D \frac{1}{q(x)} \right\rceil$ satisfies:*

$$H_D(p) + D_D(p\|q) \leq \mathbb{E}_p\left[l(X)\right] < H_D(p) + D_D(p\|q) + 1.$$

Proof.

$$\begin{aligned}
\mathbb{E}_p\left[l(X)\right] &= \sum_{x\in\mathcal{X}} p(x) \left\lceil \log_D \frac{1}{q(x)} \right\rceil \\
&< \sum_{x\in\mathcal{X}} p(x) \left(1 + \log_D \frac{1}{q(x)}\right) \\
&= 1 + \sum_{x\in\mathcal{X}} p(x) \log_D \left(\frac{p(x)}{p(x)} \cdot \frac{1}{q(x)}\right) \\
&= 1 + \sum_{x\in\mathcal{X}} p(x) \log_D \frac{1}{p(x)} + \sum_{x\in\mathcal{X}} p(x) \log_D \frac{p(x)}{q(x)} \\
&= 1 + H_D(p) + D_D(p\|q).
\end{aligned}$$

The lower bound follows by similar steps, while replacing the first couple of lines with:

$$\mathbb{E}_p\left[l(X)\right] = \sum_{x\in\mathcal{X}} p(x) \left\lceil \log_D \frac{1}{q(x)} \right\rceil \geq \sum_{x\in\mathcal{X}} p(x) \log_D \frac{1}{q(x)}.$$

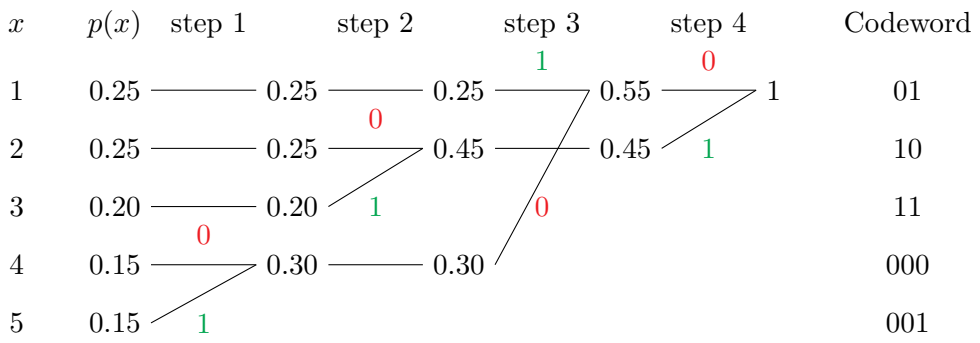
□

Intuition. The inefficiency of describing $X \sim p(x)$ using a D-ary prefix code for $q(x)$ is given by the relative entropy $D_D(p\|q)$. This is intuitive: the *closer* $q(x)$ is to the true distribution $p(x)$, the less the penalty is. This also gives the relative entropy a concrete operational interpretation.

4 Huffman Codes

Huffman codes are optimal prefix codes that can be constructed using a simple and elegant algorithm. The main idea of the algorithm is to construct the code backwards, i.e. starting from the leaves of the D-ary code tree. This is demonstrated through the following example.

Example 3. Consider a random variable X taking values in $\{1, 2, 3, 4, 5\}$ with a probability assignment of $0.25, 0.25, 0.2, 0.15, 0.15$, respectively. We wish to construct a binary Huffman code for X. This is carried out as follows:



The above algorithm start with the two least probable symbols, i.e. 4 and 5. These symbols must be assigned the longest codewords, as they have the lowest probabilities; and the two assigned codewords must have equal lengths, as otherwise we may delete the last digit of the longer codeword and still have a prefix code with a shorter expected length (why?). In general, the two least probable symbols can be assigned codewords that differ only in the last digit, which is revealed after step 1 as seen in the above diagram. Symbols 4 and 5 are combined into a single symbol which has a probability of 0.3.

For step 2, we have four symbols for which a similar procedure is carried out: we combine the two least probable symbols while assigning them different binary digits. This is repeated until we have only one symbol. Each codeword is then constructed by tracing the path back to its corresponding symbol and concatenating the binary digits appearing along the way. Note that the most significant digit is the first digit appearing on the way back, i.e. the last digit revealed during the combining procedure. In this example, we have $H(X) = 2.2855$ bits, while the expected length of the code is $L = 2.30$.

Exercise 3. Recall the roof code from earlier, which has a codeword length assignment of $l_i = \lceil \log(1/p_i) \rceil$. Find the roof code length assignment for the pmf in the above example. Compare the expected lengths of the roof code and the Huffman code. Which one does better?

Exercise 4. Consider X which takes values in $\{1, 2\}$ with probabilities $p_1 = 0.9999$ and $p_1 = 0.0001$. Design a binary Huffman code based on the procedure in Example 3. Find the roof code length assignment and compare it to the length assignment of the Huffman code. Compare their expected lengths.

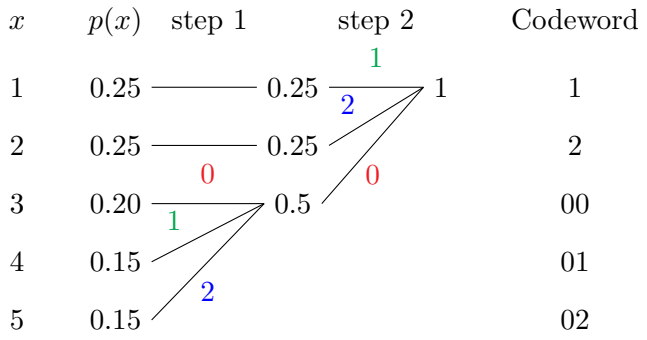
4.1 General algorithm and more examples

For the general D-ary case, the algorithm is described as follows:

1. Take the D least probably symbols in the alphabet. These symbols will be assigned the longest codewords, which will have equal lengths, and differ only in the last digit.

2. Combine these D symbols into a single symbol, and the repeat the procedure.

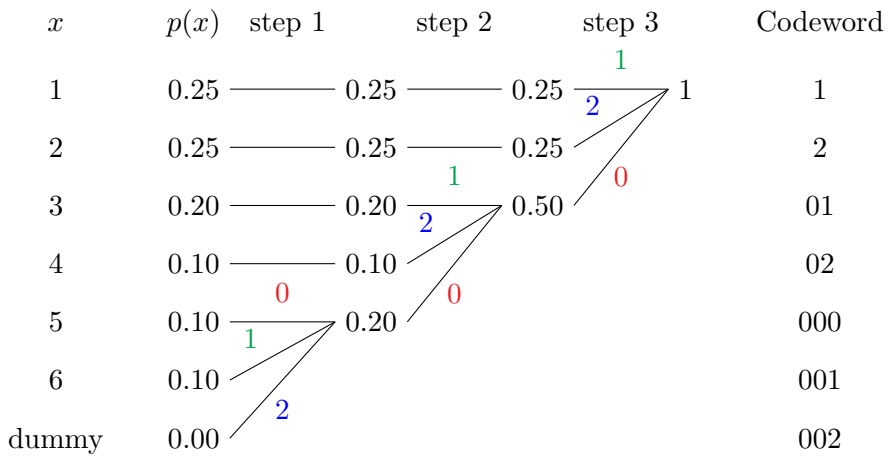
Example 4. We now revisit Example 3 but with ternary code ($D = 3$). The code is constructed as:



Here in each step we combine the three least probable symbols. Note that the ternary entropy of X is given by $H_3(X) = 1.44197$, while the expected length of the code here is 1.5 ternary digits.

Whenever $D \geq 3$, we need to make sure that the number of symbols in the alphabet takes the form $1 + k(D-1)$ for some integer k. This is to insure that in each step we have at least D symbols to combine. This is demonstrated through the following example.

Example 5. Consider the following ternary code construction:



Here we added a dummy symbol so that the total number of symbols take the form $1+k(D-1)$. Calculate an compare the ternary entropy and the expected code length.

Note that in all above examples, we employed the following rules when assigning D-ary digits to symbols after each combining step: we assigned 0 to the most probable symbol, 1 to the second probable symbol, and so on; in case we had more than one symbol with the same probability, we assigned a smaller code digit to the symbol which is listed first and so on. There is nothing special about these rules. This symbol assignment can be arbitrary, and hence the codeword assignment is not unique. Even codeword lengths are not unique, as seen through the following exercise.

Exercise 5. Consider a random variable X with a probability assignment of $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{12}\right)$. Construct two different binary Huffman codes for this random variable: the first with codeword lengths of $(2, 2, 2, 2)$, and the second with codeword lengths of $(1, 2, 3, 3)$. Note that both codes have the same expected length.

4.2 Optimality of Huffman codes*

Theorem 6. *A Huffman code is an optimal prefix code, i.e. it has a minimal expected length.*

The rest of this part is dedicated to proving the above theorem. We focus on the binary case, however the result also extends to D-ary codes. Throughout the following discussion, we consider a pmf $\boldsymbol{p} = (p_1, p_2, \ldots, p_m)$ and we assume without loss of generality that $p_1 \geq p_2 \geq \cdots \geq p_m$.

As we saw through Exercise 5, Huffman codes (and optimal codes in general) are not unique. We present a class of optimal instantaneous codes which we call *canonical* codes.

Lemma 1. *For any pmf $\boldsymbol{p}$ as defined above, there exists an optimal instantaneous code with lengths $\boldsymbol{l} = (l_1, l_2, \ldots, l_m)$ that satisfies the following properties:*

1. *Codeword lengths are ordered inversely with probabilities, i.e. $p_j > p_k \implies l_j \leq l_k$.*
2. *The two longest codewords are assigned to the two least probable symbols and have the same length.*
3. *The codewords assigned to the two least probable symbols differ only in the last bit.*

Proof. We prove the three points as follows.

1. Suppose that we have an optimal prefix code C for which $l_j > l_k$ for $p_j > p_k$. We design a new prefix code C' by swapping codewords j and k in C, and retaining all other codewords. Denoting the expected lengths of C and C' by L and L', respectively, we have:

$$L - L' = p_j l_j + p_k l_k - p_j l_k + p_k l_j$$

$$
\begin{aligned}
&= (p_j - p_k)(l_j - l_k) \\
&> 0
\end{aligned}
$$

where the last inequality holds since $p_j - p_k > 0$ and $l_j - l_k > 0$. This yields a contradiction, and therefore we must have $l_j \leq l_k$ for an optimal code.

2. From the first point, it follows that $l_1 \leq l_2 \leq \cdots \leq l_{m-1} \leq l_m$, and therefore the least probable symbols (i.e. $m-1$ and m) will be assigned the longest codewords. Moreover, we must have $l_{m-1} = l_m$. To show this, suppose that we have a prefix code where $l_m = l_{m-1} + 1$. We may remove the extra bit in codeword m, while still preserving the prefix property (as no codeword is a prefix of another), hence obtaining a prefix code with a shorter expected length.

3. If we construct a binary tree of an optimum prefix code, we will see that a codeword of maximum length must always have a sibling stemming from the same parent node (i.e. differing in only the last bit). If this is not the case, we may always trim the last bit of the longer codeword and obtain a new prefix code with a shorter expected length. Therefore, after a possible rearrangement, we may always assign sibling codewords that differ in only the last bit to symbols $m-1$ and m.

□

We are now ready to prove that Huffman codes are optimal: here is an outline of the proof. We start by defining a reduced pmf given by $\boldsymbol{p}' = (p_1, p_2, \ldots, p_{m-1} + p_m)$, obtained by combining the two smallest probabilities in $\boldsymbol{p}$. Let $B^\star$ be Huffman code for $\boldsymbol{p}'$ with lengths $k_1^\star, k_2^\star, \ldots, k_{m-1}^\star$. Looking at the diagram in Example 3, the reduced code constitutes the first three codewords in addition to a codeword obtained by omitting the least significant bit in the fourth (or fifth) codeword. We suppose that $B^\star$ is optimal for $\boldsymbol{p}'$, and then extend $B^\star$ to construct a new Huffman code C for $\boldsymbol{p}$ (by adding an additional bit to codeword $m-1$). We then prove that C is optimal by comparing its expected length to that of a canonical code. It then only remains to show that $B^\star$ is optimal to conclude the proof. This follows by induction, i.e. combining the least two probable codewords in $B^\star$ and then doing the same for the resulting reduced pmf until we reach the case of 2 symbols, where the optimal Huffman code is obvious.

We now present details, starting with extending $B^\star$ to construct a code C for $\boldsymbol{p}$. This is done as follows: for all $i \leq m-2$, we set $C_i = B_i^\star$. As for the last two codewords, they are given by extending $B_{m-1}^\star$ as $C_{m-1} = B_{m-1}^\star\ 0$ and $C_m = B_{m-1}^\star\ 1$. We now calculate the average length L of C as follows:

$$
\begin{aligned}
L &= \sum_{i=1}^{m} p_i l_i \\
&= (p_{m-1} + p_m)(k_{m-1}^\star + 1) + \sum_{i=1}^{m-2} p_i k_i^\star \\
&= (p_{m-1} + p_m) + \sum_{i=1}^{m-1} p_i' k_i^\star \\
&= K^\star + p_{m-1} + p_m. \qquad (18)
\end{aligned}
$$

We wish to compare L with the optimal expected length $L^\star$. To facilitate this comparison, we define the canonical code $C^\star$ with lengths $l_1^\star, l_2^\star, \ldots, l_m^\star$, where $l_1^\star \geq l_2^\star \geq \ldots \geq l_m^\star$. Next, from $C^\star$ we construct a new code B for the reduced pmf $\boldsymbol{p}'$ by merging the two longest codewords. In particular, we have $B_i = C_i^\star$ for all $i < m-1$, while B_{m-1} is obtained from $C_{m-1}^\star$ (or $C_m^\star$) by removing the least significant bit. The average length K of B_{m-1} is calculated as:

$$
K = \sum_{i=1}^{m-1} p_i' k_i
$$

$$
\begin{aligned}
&= (p_{m-1} + p_m)(l^\star_{m-1} - 1) + \sum_{i=1}^{m-2} p_i l^\star_i \\
&= -(p_{m-1} + p_m) + \sum_{i=1}^{m} p_i l^\star_i \\
&= L^\star - (p_{m-1} + p_m). \qquad (19)
\end{aligned}
$$

From (18) and (19), we have $L = K^\star + p_{m-1} + p_m$ and $K = L^\star - (p_{m-1} + p_m)$. By adding these two equations, we obtain

$$(L^\star - L) + (K^\star - K) = 0. \qquad (20)$$

Since $C^\star$ is optimal for $\boldsymbol{p}$, then we must have $(L^\star - L) \leq 0$. Similarly, since we have assumed that $B^\star$ is optimal for $\boldsymbol{p}'$, then we also have $(K^\star - K) \leq 0$. Combing these inequalities with (20), it follows that $K^\star = K$ and $L^\star = L$. This proves that the Huffman code C, obtained by extending $B^\star$, is optimal as it has an expected length equal to $L^\star$. The supposed optimality of $B^\star$ can be proved by induction as explained earlier. This concludes the proof of Theorem 6.