Unit 9: Compression - 1 (Entropic Compression)

A Variable length source codes

We begin with the most basic problem in compression: Given a symbol $X \in \mathcal{X}$ generated using a known pmf P, what is the minimum number of bits required to store it?

Earlier we formulated a fundamental quantity $L_\varepsilon(P)$ which represents this minimum number of bits for *fixed length codes*, namely codes that map each symbol to a binary sequence of the same length. We saw that this quantity is roughly $H(P)$ (a bound which was asymptotically precise). We used this notion to motivate the definition of security. But the scheme we formulated was rather naive. The decoder formed a guess list (comprising symbols x such that $-\log P(x) < \lambda$ for an appropriately chosen λ), the encoder sent a random hash $F(x)$ and the decoder declared the $\hat{x}$ element in its guess list for which $F(\hat{x}) = F(x)$.

The use of hashing as a tool for compression is prevalent in cache memories and databases (we will visit these topics in Unit 11), but in its basic form, the compression scheme described above is not very practical. Instead, we seek variable length codes where symbols are mapped to binary sequences of possibly different lengths. We remark that variable length codes are not suitable for many protocols, but clever engineering (using things like *bit packing*) is often used to convert variable length codes to fixed length codes.

A *variable length source code* for a source $X \in \mathcal{X}$ with pmf P consists of an encoder mapping $e : \mathcal{X} \to \{0,1\}^*$ and a decoder mapping $d : \{0,1\}^* \to \mathcal{X}$. The average length of the source code (e, d) is given by $\sum_x P(x)|e(x)|$, where $|b|$ denotes the length of a binary

vector $b \in \{0,1\}^*$. Finally, the probability of error for the source code (e,d) which estimates X as $\hat{X} = d(e(X))$ is given by $P(X \neq \hat{X})$.

The fundamental quantity of interest for us is the minimum average length of a source code with probability of error less than ε, denoted by $\overline{L}_\varepsilon(P)$. Namely,

$$\overline{L}_\varepsilon(P) = \min \left\{ \sum_x P(x)|e(x)| : \text{source code } (e,d) \text{ s.t.} P(X \neq d(e(X))) \leq \varepsilon \right\}.$$

When $\varepsilon = 0$, we abbreviate $\overline{L}(P) = \overline{L}_0(P)$. Note that while $L(P) = L_0(P) = \log|\mathcal{X}|$, the quantity $\overline{L}(P)$ is nontrivial.

We saw earlier that a basic benchmark for minimum length $L_\varepsilon(P)$ for fixed length codes is roughly $H(P)$. The next result shows that roughly the same benchmark holds for $\overline{L}_\varepsilon(P)$.

Theorem 1. *For a source X over a discrete alphabet $\mathcal{X}$ and $\varepsilon \in [0,1)$, we have*

$$\overline{L}_\varepsilon(X) \geq H(X) - \log e H(X) - \varepsilon \log|\mathcal{X}| - h(\varepsilon).$$

Proof. For any source code (e,d) with probability of error less than ε, denote $\hat{X} = d(e(X))$. Then, we have

$$H(P) = H(X) \leq H(X|\hat{X}) + H(\hat{X}) \leq \varepsilon \log|\mathcal{X}| + h(\varepsilon) + H(\hat{X}),$$

where the inequality on the right-side is by Fano's inequality. Next, denoting by $(C_1, ..., C_N)$ the random codeword $e(X)$ of random length N. We have

$$H(\hat{X}) \leq H(C^N) = H(N) + H(C^N|N) \leq H(N) + \mathbb{E}[N],$$

where in the inequality we used $H(C^N|N=n) \leq n$ for every $n \in \mathbb{N}$. We now take recourse a property we saw earlier:

Geometric distribution maximizes entropy among all $\mathbb{N}$-valued random variables with

a given expected value. Specifically, for a random variable $Y \in \mathbb{N}$ such that $\mathbb{E}[Y] = \alpha$, we have (see Unit 6)

$$H(Y) \leq \alpha \log \alpha - (\alpha - 1) \log(\alpha - 1).$$

Further, first-order Taylor's approximation gives $x \log x - (x-1) \log(x-1) \leq (1 + \ln x)/\ln 2$, whereby for an $\mathbb{N}$-valued Y with $\mathbb{E}[Y] = \alpha$ we must have

$$H(Y) \leq \log \alpha + \log e.$$

Combining all the bounds above, we get

$$H(P) \leq \mathbb{E}[N] + \log \mathbb{E}[N] + \varepsilon \log |\mathcal{X}| + h(\varepsilon) + \log e.$$

Finally, we note that if $y < x + \log x + c$, then $y - \log y - c < x$. Suppose not and $x \leq y - \log y - c$. Then, $y < y - \log y + \log(y - \log y - c)$ which implies $y < y - \log y - c$, a contradiction. Therefore, we must have

$$\mathbb{E}[N] \geq H(P) - \log H(P) - \varepsilon \log |\mathcal{X}| - h(\varepsilon) - \log e,$$

which completes the proof. □

B Prefix-free codes and Kraft's inequality

We saw earlier in the course a code that attains $\overline{L}(P)$. It simply arranges the symbols in decreasing order of their probabilities and assigns binary sequences of increasing lengths as codewords to the symbols. It is easy to show that this scheme has average length less than $H(P)$. However, this is just a theoretical construction and is not at all relevant for practical deployment.

A more practical class of codes are *prefix-free* codes where no codeword is a prefix

of another. For a concrete example, consider Table 1 with examples of variable length codes. Which of these codes is a prefix-free codes? Prefix free codes have the property that a $(x_1, ..., x_k)$ of the sequence $(x_1, ..., x_n)$ coded as $(c_1, ..., c_n)$ can be recovered from $(c_1, ..., c_k)$, even without the knowledge of k. Motivated by this property, prefix-free codes are sometimes called *instantaneous codes* (short for instantaneously decodeable codes). Code 3 in Table 1 is interesting. It is not prefix-free, but it satifies the property above but when k is known. Such codes are called *uniquely decodeable codes*.

Alphabet	code1	code2	code3	code4
a	00	0	00	0
b	01	01	10	10
c	10	10	11	110
d	11	010	110	111

Table 1: Which of these codes are prefix-free codes?

The minimum over average lengths of prefix-free codes (e, d) (with 0 probability of error) is denoted by $\overline{L}^{\mathsf{p}}(X)$. Note that prefix-free property is relevant in practice when we are compressing a sequence of symbols and not a single symbol.

The next result gives a simple characterization of the lengths of prefix-free codes.

Theorem 2 (Kraft's inequality)**.** *Given a prefix-free code which assigns a codeword of length $l(x)$ to symbol $x \in \mathcal{X}$, the following bound holds:*

$$\sum_{x \in \mathcal{X}} 2^{-l(x)} \leq 1. \tag{1}$$

Conversely, for every sequence $\{l(x) \in \mathbb{N} \cup \{0\}, x \in \mathcal{X}\}$ satisfying Kraft's inequality (1) *there exists a prefix-free code which assigns to each symbol x a codeword of length $l(x)$.*

Thus, lengths of codewords of a prefix-free code are simply characterized by the relation (1). Before we proceed and prove Theorem 2, we note an important consequence of satisfying Kraft's inequality.

Lemma 3. *Given a pmf* P_X *on* $\mathcal{X}$ *and a code which assigns a codeword of length* $l(x)$ *to the symbol* $x \in \mathcal{X}$*, suppose that* $l(x)$*,* $x \in \mathcal{X}$*, satisfy Kraft's inequality* (1)*. Then, the average length of the code is at least* $H(X)$

Proof. The average length of a code with lengths satisfying Kraft's inequality satisfies

$$\begin{aligned} l_a(\mathcal{C}) &= \sum_{x\in\mathcal{X}} \mathrm{P}_X(x)\, l(x) \\ &= \sum_{x\in\mathcal{X}} \mathrm{P}_X(x) \log \frac{1}{2^{-l(x)}} \\ &= \sum_{x\in\mathcal{X}} \mathrm{P}_X(x) \log \frac{\mathrm{P}_X(x)}{2^{-l(x)}} + H(X) \\ &\geq \log \frac{1}{\sum_x 2^{-l(x)}} + H(X) \\ &\geq H(X), \end{aligned}$$

where the last bound holds using Kraft's inequality. □

Thus, prefix-free codes can only be worse in average length than the best variable length codes. But we will soon see that not by much.

We prove Theorem 2 by providing two generic constructions.

Binary-tree representation of prefix-free codes. A binary tree is a convenient pictorial representation of binary sequences. We first describe a complete binary tree. Starting from a fixed node, the so-called *root*, we obtain nodes at depth 1 by drawing edges to two nodes below it, one on the left of the root and the second on the right, referred to respectively as the left- and the right-child of the root. The nodes at depth 2 are obtained similarly by treating each node of depth 1 as a root and repeating the process described above. We proceed this way to obtain nodes of depth d. This gives us the complete binary tree of depth d; the nodes which do not have any children, namely the nodes at depth d, are called the leaves of the tree. Clearly, there are 2^d leaves in a complete binary tree.

Note that the nodes below a node of depth l in a complete binary tree of depth d themselves form the root of the $d-l$-depth complete binary tree below them; we refer to this tree as a *sub-tree* of the original tree. In general, a binary tree is obtained by starting with a complete binary tree and removing all the nodes below them; this process will remove 2^{d-l} leaves from the original complete tree when a node at depth l removed.

Also, note that there is a one-to-one map from the set of binary sequences of length less than or equal to d and the nodes of a complete binary tree of depth d. Specifically, each node of the tree can be uniquely represented by the path from root to that node, which in turn can be represented as a binary sequence by labeling each edge to the left-child with a 0 and that to a right-child with a 1. Our proof relies on representing the codewords of a prefix-free code on a binary tree using this labeling. The key observation here is that a binary sequence c of length $\leq d$ is a prefix of another binary sequence c' of length $\leq d$ if and only if the node corresponding to c' in the complete tree representation above lies in the sub-tree with the node corresponding to c as the root.

Proof of Theorem 2. Suppose $l_1 \leq l_2 \leq ... \leq l_m$ denote the sorted lengths of the prefix-free code (this proof works when $l_m < \infty$). We first place this code over a complete binary tree of depth l_m using the labeling described above. Next, we "prune" the tree so formed by removing all the nodes below each node corresponding to a codeword. Since our code is prefix-free, none of the codeword nodes is removed in our procedure. We finally end-up with a binary tree where each codeword is a leaf node. Note that the number of leaf nodes removed corresponding to a codeword of length l_i is $2^{l_m - l_i}$. Since there were 2^{l_m} leaves to begin with, we must have

$$\sum_{i=1}^{m} 2^{l_m - l_i} \leq 2^{l_m},$$

which is equivalent to Kraft's inequality. Thus, the codeword lengths for a prefix-free code satisfy Kraft's inequality.

For the converse, we just "invert" the proof above. Given a set of lengths $l_1 \leq l_2 \leq ... \leq l_m$ satisfying kraft's inequality, consider the complete binary tree of depth l_m. Our

construction proceeds iteratively by identifying the codewords corresponding to $l_1, ..., l_m$. At the ith step, we start with a leaf of the original complete binary tree of depth l_m and identify a node at depth l_i above it, *i.e.*, we move from this leaf to the root and stop when we are at depth l_i. We remove the sub-tree with this node as its root. This results in the removal of $2^{l_m - l_i}$ leaves of the original tree. Next, we proceed to the next available leaf and repeat the process above. Note that the removed sub-tree at each step does not contain a codeword since otherwise the length of the removed codeword will be greater than the current codeword; this cannot be the case since *we have assumed that the lengths are sorted in a nondecreasing order*. This process can continue as long as the number of leaves removed till step i is less than the total number of leaves. That is, as long as

$$\sum_{j=1}^{i} 2^{l_m - l_j} \leq 2^{l_m}.$$

Since the lengths satisfy Kraft's inequality, the condition above continues to hold for $1 \leq i \leq m$. The desired prefix-free code is given by the binary sequences corresponding to the leaf nodes of the tree so obtained.

Interval representation of prefix-free code. Our second proof represents codewords of a prefix-free code by non-intersecting intervals which are subset of $[0, 1]$. Specifically, given $c = (c_1, ..., c_l) \in \{0, 1\}^l$, let $I_c = [0.c_1c_2...c_l, 0.c_1c_2...c_l + 2^{-l})$ where $0.c_1c_2...c_l$ denotes the corresponding binary number. For example, 0.1 is 1/2, 0.11 is 1/2+1/4, and so on. We first prove that the intervals I_c, $c \in \mathcal{C}$, corresponding to a prefix-free code $\mathcal{C}$ are non-intersecting. Therefore, since each interval is in $[0, 1]$, the sum of the lengths of the interval is less than 1, which gives Kraft's inequality. It only remains to prove the non-intersection claim. To that end, note that the interval I_c consists of all $x \in [0, 1]$ starting with $0.c_1...c_l$ since $x \in I_c$ iff

$$0.c_1...c_l \leq x \leq 0.c_1...c_l + 2^{-l} = 0.c_1...c_l1111\ldots.$$

Thus, if $x \in I_c \cap I_{c'}$ then x must start with $0.c$ as well as $0.c'$, which is a contradiction since

the code is prefix free.

Conversely, given lengths $l_1 \leq .... \leq l_m$ satisfying Kraft's inequality, we identify codewords $c^{(1)}, ..., c^{(m)}$ of lengths $l_1, ..., l_m$, respectively, such that the corresponding intervals I_{c^i} are non-intersecting. By the argument above, such codewords must be prefix-free. The construction is simple. Let $c^{(1)}$ be the all zero sequence of length l_1. For $1 < i \leq m$, let $x_i = \sum_{j=1}^{i-1} 2^{-l_j}$. Since the lengths satisfy Kraft's inequality, $x_i \in [0, 1]$ for all $1 < i \leq m$. The required codewords are given by binary representations of x_i truncated to l_i most significant bits. In particular, for $1 < i \leq m$, $c^{(i)}$ is sequence of length l_i with 1's at the $l_1, l_2, ..., l_i - 1$th locations and zero everywhere else. For example, let $l_1 = 1$, $l_2 = 2$, $l_3 = l_4 = 3$. Clearly, these lengths satisfy Kraft's inequality. Then, the required code is given by $\{0, 10, 110, 111\}$. Note that here, too, *it is necessary to have the lengths in a nondecreasing order.*

C Shannon codes

We now show that there exist prefix free code of average length less than $H(X) + 1$. Note that in view of Theorem 2 we only need to exhibit codeword lengths which satisfy Kraft's inequality and the average length is less than $H(X) + 1$; the code can be constructed using the tree based construction or the interval based construction in the proof of Theorem 2. To that end, we introduce Shannon codes, which actually is a class of codes.

Given a source with discrete alphabet $\mathcal{X}$ and pmf P_X, a prefix-free code is called a Shannon code for P_X if the codeword lengths $l(x)$ satisfy $l(x) = \lceil -\log \mathrm{P}_X(x) \rceil$, for each x in $\mathcal{X}$.

These lengths satisfy Kraft's inequality since

$$\sum_{x \in \mathcal{X}} 2^{-l(x)} = \sum_{x \in \mathcal{X}} 2^{-\lceil -\log \mathrm{P}_X(x) \rceil} \leq \sum_{x \in \mathcal{X}} 2^{\log \mathrm{P}_X(x)} = \sum_{x \in \mathcal{X}} \mathrm{P}_X(x) = 1.$$

Therefore, by Theorem 2 there exists a prefix-free code with these lengths. Furthermore,

the average length of such a code is

$$\sum_{x\in\mathcal{X}} \mathrm{P}_X(x)\, l(x) = \sum_{x\in\mathcal{X}} \mathrm{P}_X(x) \lceil -\log \mathrm{P}_X(x) \rceil \leq \sum_{x\in\mathcal{X}} (-\log \mathrm{P}_X(x) + 1) = H(X) + 1.$$

Thus, we have shown that for every discrete source with pmf P_X

$$H(X) \leq \overline{L}^{\mathsf{p}} \leq H(X) + 1.$$

For an example of a Shannon code, consider a source which takes 4 possible values a, b, c, d with probabilities $\{1/8, 1/4, 1/2, 1/8\}$, respectively. For this source, required lengths for a Shannon code are listed in Table 2. We provide two different codes for these lengths: Code 1 is based on the binary tree construction and code 2 on the interval construction.

For code 1, we start with a complete binary tree of depth 3; c is placed on the node 0, b on the node 10, a on the node 110 and d on the node 111. Note that it is important to first sort the lengths for this process to work. For instance, if we assign a to 000 first and then look for b above the leaf node 001, the process fails.

For code 2, once again we proceed in the nondecreasing order of lengths; c is assigned to the interval $[0.0, 0.1)$, b to the interval $[0.10, 0.11)$, a to the interval $[0.110, 0.111)$ and d to the interval $[0.111, 1)$.

Note that both the constructions lead to the same code. However, this is due to the order we followed in selected the next available leaf in the tree construction. We selected the available leaf with the least value in binary; a different rule for selecting the next available leaf will lead to different codes.

D Huffman code: Prefix-free codes of optimal length

We have seen that $H(P) \leq \overline{L}^{\mathsf{p}}(P) \leq H(P) + 1$. The exact value of $\overline{L}^{\mathsf{p}}(P)$ is attained by the average length of Huffman code, which we describe now. We represent a Huffman code

Alphabet	$\mathrm{P}_X(x)$	$l(x)$	code
a	0.125	3	110
b	0.25	2	10
c	0.5	1	0
d	0.125	3	111

Table 2: An example of a Shannon code.

using the binary-tree representation.

Input: Source distribution $\mathrm{P} = (p_1, ..., p_m)$

Output: Code $\mathcal{C} = \{c_1, ..., c_m\}$.

1. Associate with each symbol a leaf node.

2. **while** $m \geq 2$, **do**

   (i) If $m = 2$, assign the two symbols as the left and the right child of the root. Update m to 1.

   (ii) Sort the probabilities of symbols in a descending order. Let $p_1 \geq p_2 \geq ... \geq p_m$ be the sorted sequence of probabilities.

   (iii) If $m > 2$, assign the symbols $(m-1)$th and the mth symbols as the left and the right child of a node. Treat this node and its subtree as a new symbol which occurs with probability $(p_{m-1} + p_m)$ and associate a leaf with it. Update $m \to m-1$ and $\mathrm{P} \to (p_1, ..., p_{m-2}, p_{m-1} + p_m)$.

3. Generate a binary code from the tree by putting a 0 over each edge to a left child and 1 over each edge to the right child.

For illustration, consider our foregoing example once more. The algorithm proceeds as follows[1]:

$$((a, 0.125); (b, 0.5); (c, 0.25)); (d, 0.125))$$

[1]The reader can easily decode our representation of the evolving tree.

$$((b, 0.5); (c, 0.25)); (ab, 0.25))$$

$$((b, 0.5); (c(ab), 0.5))$$

$$(b(c(ab)), 1).$$

In summary, we have the following code: The average length for this example is $7/4$ which is

Alphabet	P_X	codeword
a	$1/8$	110
b	$1/2$	0
c	$1/4$	10
d	$1/8$	111

Table 3: An illustration of Huffman code.

equal to the entropy. Therefore, the average length must be optimal. In fact, the Huffman code always attains the optimal average length.

Theorem 4. *A Huffman code has optimal average length.*

We only provide a sketch of the proof.

Proof sketch. The proof relies on the following observation:

There exists a prefix-free code of optimal length which assigns two symbols with the least probability to the two longest codewords of length $l_{\max}$ such that the first $l_{\max} - 1$ bits for the two codewords are the same.

Next, we show the optimality of Huffman code by induction on the number of symbols. Denote by $L(\mathrm{P})$ the minimum average length of a prefix-free code. Suppose Huffman code attains the optimal length for every probability distribution over an alphabet of size $m-1$. For $\mathrm{P} = (p_1, ..., p_m)$, consider the prefix-free code $\mathcal{C}$ of average length $L(\mathrm{P})$ guaranteed by the observation above. Let $L_H(\mathrm{P})$ denote the length of the optimal Huffman code. Then,

$$\begin{aligned} L((p_1, ..., p_m)) &\leq L_H((p_1, ..., p_m)) \\ &= (p_{m-1} + p_m) + L_H((p_1, ..., p_{m-2}, p_{m-1} + p_m)) \end{aligned}$$

$$= (p_{m-1} + p_m) + L((p_1, ..., p_{m-2}, p_{m-1} + p_m)),$$

where the previous equality is by the induction hypothesis. On the other hand, by property of the optimal code $\mathcal{C}$, it also yields a prefix-free code for $(p_1, ..., p_{m-2}, p_{m-1}+p_m)$ of average length $L((p_1, ..., p_m)) - (p_{m-1} + p_m)$. But then

$$L((p_1, ..., p_m)) \geq (p_{m-1} + p_m) + (p_{m-1} + p_m) + L((p_1, ..., p_{m-2}, p_{m-1} + p_m)).$$

Thus, by combining all the bounds above, all inequalities must hold with equality. In particular,

$$L((p_1, ..., p_m)) = L_H((p_1, ..., p_m)).$$

□

In summary, Huffman code is the optimal prefix-free code and cost us at most roughly $\log H(P) + \varepsilon \log |\mathcal{X}|$ in average length than any variable-length code with probability of error ε.