Problem Set 3

published: 28 April 2022
discussion: 6 May 2022

Data Compression With And Without Deep Probabilistic Models
Prof. Robert Bamler, University of Tuebingen

Course materials available at https://robamler.github.io/teaching/compress22/

Problem 3.1: Kraft-McMillan Theorem

In the lecture, we discussed the Kraft-McMillan Theorem:

Theorem 1 (Kraft-McMillan)**.** *Let $B \geq 2$ be an integer and let $\mathfrak{X}$ be a finite or countably infinite set (referred to as "the alphabet"). Then the following two statements are true:*

(a) All B-ary uniquely decodable symbol codes C on $\mathfrak{X}$ satisfy the Kraft inequality,

$$\sum_{x \in \mathfrak{X}} \frac{1}{B^{\ell_C(x)}} \leq 1 \tag{1}$$

where $\ell_C(x) := |C(x)|$ is the length of the code word $C(x)$.

(b) For all functions $\ell : \mathfrak{X} \to \{0, 1, 2, \ldots\}$ that satisfy the Kraft inequality (Eq. 1), there exists a B-ary prefix-free symbol code (aka, a B-ary prefix code) C with code word lengths ℓ, i.e., $|C(x)| = \ell(x)\ \forall x \in \mathfrak{X}$.

We proved part (a) of the Kraft-McMillan theorem in the lecture but we left out the last bit of the proof of part (b). Let's fill this gap now. Consider Algorithm 1, which we introduced in the lecture.

(a) Line 4 of Algorithm 1 claims that $\xi \in [0, 1)$. Why is this the case every time the algorithm arrives at this line?

Algorithm 1: Constructive proof of Kraft-McMillan theorem part (b).

```
Input:  Base $B \in \{2, 3, \ldots\}$, finite alphabet $\mathfrak{X}$,
        function $\ell : \mathfrak{X} \to \{0, 1, 2, \ldots\}$ that satisfies Eq. 1.
Output: Code book $C : \mathfrak{X} \to \{0, \ldots, B-1\}^*$ of a prefix code that
        satisfies $|C(x)| = \ell(x)\ \forall x \in \mathfrak{X}$.
1 Initialize $\xi \leftarrow 1$;
2 for $x \in \mathfrak{X}$ in order of nonincreasing $\ell(x)$ do
3     Update $\xi \leftarrow \xi - B^{-\ell(x)}$;
4     Write out $\xi \in [0, 1)$ in its $B$-ary expansion: $\xi = (0.??? \ldots)_B$;
5     Set the code word $C(x)$ to the first $\ell(x)$ bits following the "0." in the above
        $B$-ary expansion of $\xi$ (pad with trailing zeros to length $\ell(x)$ if necessary);
```

(b) Denote the value of ξ immediately after its update on Line 3 as ξ_x (where x is the iteration variable of the `for` loop). Now consider two symbols $x, x' \in \mathfrak{X}$ with $x \neq x'$ and, without loss of generality, $\xi_{x'} > \xi_x$. Argue that $\xi_{x'} \geq \xi_x + B^{-\ell(x)}$. Then argue that neither can $C(x)$ be a prefix of $C(x')$ nor can $C(x')$ be a prefix of $C(x)$.

(c) *(Advanced:)* Algorithm 1 is limited to a finite alphabet $\mathfrak{X}$ because the `for` loop would not terminate for an infinite $\mathfrak{X}$. Why does part (b) of the Kraft-McMillan Theorem nevertheless hold for countably infinite alphabets too?

(d) More generally, why do we always insist that $\mathfrak{X}$ must be *countably* infinite if it is infinite? Argue why lossless compression on an uncountable alphabet is impossible. You don't need to think about Algorithm 1 to answer this question, just think about what a lossless compression code is from a purely mathematical perspective.

Problem 3.2: Shannon Coding

In Problem 2.1 on Problem Set 2, we constructed Huffman codes C_{Huff} for three different probability distributions. For your reference, the following table summarizes our results from that problem (you may have obtained a different Huffman tree in the third example if you broke the tie differently).

x	$p(x)$	$C_{\text{Huff}}(x)$	$C_{\text{Shannon}}(x)$	$p(x)$	$C_{\text{Huff}}(x)$	$C_{\text{Shannon}}(x)$	$p(x)$	$C_{\text{Huff}}(x)$	$C_{\text{Shannon}}(x)$
'a'	0.4	'0'		0.3	'00'		0.05	'000'	
'b'	0.3	'10'		0.28	'01'		0.07	'001'	
'c'	0.2	'110'		0.12	'100'		0.12	'010'	
'd'	0.1	'111'		0.1	'101'		0.12	'011'	
'e'	–	–	–	0.2	'11'		0.64	'1'	
		$L=1.9$			$L=2.22$			$L=1.72$	

(a) Calculate the entropy $H_2[p]$ of each of the three probability distributions p in the above table. Then verify explicitly for these three examples that

$$H_2[p] \leq L_{\text{Huff}} < H_2[p] + 1 \tag{2}$$

where L_{Huff} is the expected code word length of C_{Huff}, which is given in the last line of the above table (you already calculated this on the last problem set.

(b) For each of the three probability distributions p, construct the Shannon code C_{Shannon} by applying Algorithm 1 to the code word lengths $\ell(x) = \lceil -\log_2 p(x) \rceil$ $\forall x \in \mathfrak{X}$, where $\lceil \cdot \rceil$ denotes rounding up to the nearest integer (you may want to use a simple Python one-liner to calculate all $\ell(x)$ in one go). Calculate the expected code word length L_{Shannon} for each example and verify that

$$H_2[p] \leq L_{\text{Huff}} \leq L_{\text{Shannon}} < H_2[p] + 1. \tag{3}$$

(c) Come up with some probability distribution p with $p(x) > 0$ $\forall x \in \mathfrak{X}$ with $|\mathfrak{X}| = 5$ for which $H_2[p] = L_{\text{Huff}} = L_{\text{Shannon}}$. What property does p have to satisfy?

Problem 3.3: Entropy and Information Content

In the lecture, we defined the information content to base B of a symbol x under a probabilistic model p as

$$\text{information content} := -\log_B p(x). \tag{4}$$

Further, we defined the entropy $H_B[p]$ as the *expected information content*,

$$H_B[p] := \mathbb{E}_p[-\log_B p(x)] = -\sum_{x\in\mathfrak{X}} p(x) \log_B p(x) \tag{5}$$

(a) In the literature, the subscript B is often dropped. Depending on context, information contents and entropies are understood to be either to base 2 (mostly in the compression literature) or to the natural base e (in mathematics, statistics, or machine learning literature, and also often when you implement stuff in real code). How do entropies and information contents to base $B = 2$ and to base $B = e$ relate to each other?

(b) *(Additivity of information contents and entropies of statistically independent random variables:)* Consider two symbols $x_1 \in \mathfrak{X}_1$ and $x_2 \in \mathfrak{X}_2$ from alphabets $\mathfrak{X}_1$ and $\mathfrak{X}_2$, respectively. Assume that x_1 and x_2 are *statistically independent*, i.e., that the probability distribution $\tilde{p} : (\mathfrak{X}_1 \times \mathfrak{X}_2) \to [0, 1]$ of the tuple (x_1, x_2) is a product of two probability distributions,

$$\tilde{p}\big((x_1, x_2)\big) = p_1(x_1)\, p_2(x_2) \qquad \forall x_1 \in \mathfrak{X}_1, x_2 \in \mathfrak{X}_2 \tag{6}$$

where $p_1 : \mathfrak{X}_1 \to [0, 1]$ and $p_2 : \mathfrak{X}_2 \to [0, 1]$ are probability distributions on $\mathfrak{X}_1$ and $\mathfrak{X}_2$, respectively (more on statistical independence in the next lecture). Show that, in this case, information contents and entropies are additive, i.e., in particular,

$$H_B[\tilde{p}] = H_B[p_1] + H_B[p_2] \qquad \forall B > 0. \tag{7}$$

Problem 3.4: Block Codes and Source Coding Theorem

In the lecture, we showed that the expected code word length L_C of a uniquely decodable *symbol code* C is lower bounded by the entropy $H_B[p]$ of the symbols. We further showed that this lower bound is nontrivial: for any probability distribution p of symbols, there exists a symbol code (the so-called Shannon Code C_{Shannon}) that is prefix-free (and therefore uniquely decodable) and for which $L_{C_{\text{Shannon}}}$ comes within less than one bit of overhead (per symbol) of this lower bound. Thus, the *optimal* uniquely decodable symbol code C_{opt} (i.e., the one with lowest expected code word length) satisfies

$$H_B[p] \leq L_{\text{opt}} < H_B[p] + 1. \tag{8}$$

So far, Eq. 8 is limited to symbol codes, i.e., to codes for which the encoding $C^*(\mathbf{x})$ of a sequence $\mathbf{x} = (x_i)_{i=1}^k$ of symbols x_i is given by simple concatenation of individual code words $C(x_i)$. In this problem, you will analyze how Eq. 8 changes if we generalize it beyond symbol codes.

We introduce the concept of a *block code*: Let $m \in \{2, 3, \ldots\}$ and assume, for simplicity, that you only care about messages $\mathbf{x} \in \mathfrak{X}^k$ whose length k, is a multiple of m, i.e., $k = nm$ for some integer n. You can then group the symbols in $\mathbf{x}$ into n blocks of m consecutive symbols each, and you can construct a symbol code for these blocks.

For example, a message $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$ of length $k = 6$ can be reinterpreted as a message of $n = 2$ blocks of size $m = 3$ each: $\tilde{\mathbf{x}} = \big((x_1, x_2, x_3), (x_4, x_5, x_6)\big)$. Each block is an element of the product alphabet $\mathfrak{X}^m$. One can now construct a code book for blocks, $\tilde{C}^{(m)} : \mathfrak{X}^m \to \{0, \ldots, B-1\}^*$. In particular, we will consider an *optimal* (uniquely decodable) code $\tilde{C}^{(m)}_{\text{opt}} : \mathfrak{X}^m \to \{0, \ldots, B-1\}^*$ on the product alphabet $\mathfrak{X}^m$ with respect to the product probability distribution $\tilde{p}\big((x_1, \ldots, x_m)\big) := \prod_{i=1}^m p(x_i)$.

(a) Use Eqs. 7-8 to derive a lower and an upper bound for the expected length of the encoding *per original symbol (from $\mathfrak{X}$)* for $\tilde{C}^{(m)}_{\text{opt}}$. You should find that the lower bound does not change compared to Eq. 8, but the upper bound shrinks, i.e., the range of possible values narrows.

(b) When we set $m = k$ then the entire message is considered as a single block, and $\tilde{C}^{(m)}_{\text{opt}}$ is really the optimal code *in general* (not just the optimal *block code*) on the message space $\mathfrak{X}^k$. What can you say about $\tilde{C}^{(m)}_{\text{opt}}$ with $m = k$ in the (in practice highly relevant) limit of large k? Think about

   (i) its overhead over the theoretical lower bound $H_B[p]$ for the expected number of bits per original symbol $x_i \in \mathfrak{X}$; and

   (ii) the run-time complexity as a function of m for the process of constructing $C^{(m)}_{\text{opt}}$, e.g., by Huffman coding.