Problem Set 2

published: 26 April 2023
discussion: 3 May 2023

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

Course materials available at https://robamler.github.io/teaching/compress23/

Problem 2.1: Kraft-McMillan Theorem

In the lecture, we discussed the Kraft-McMillan Theorem. Here's a reminder:

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,

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

where $|C(x)|$ is the length of the code word $C(x)$.

(b) For all functions $\ell : \mathfrak{X} \to \mathbb{N}$ that satisfy the Kraft inequality (i.e., $\sum_{x \in \mathfrak{X}} \frac{1}{B^{\ell(x)}} \leq 1$), there exists a B-ary prefix-free symbol code (aka, a B-ary prefix code) C_ℓ with code word lengths $|C_\ell(x)| = \ell(x)\ \forall x \in \mathfrak{X}$.

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

(b) Denote the value of ξ on Line 4 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 $C_\ell(x)$ be a prefix of $C_\ell(x')$ nor can $C_\ell(x')$ be a prefix of $C_\ell(x)$. Thus, C_ℓ is a prefix code as claimed.

(c) 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 also hold for countably infinite alphabets?

(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.

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

Input: Base $B \in \{2, 3, \ldots\}$, finite alphabet $\mathfrak{X}$, function $\ell : \mathfrak{X} \to \mathbb{N}$ that satisfies the Kraft inequality (i.e., $\sum_{x \in \mathfrak{X}} \frac{1}{B^{\ell(x)}} \leq 1$).

Output: Code book $C_\ell : \mathfrak{X} \to \{0, \ldots, B-1\}^*$ of a prefix code that satisfies $|C_\ell(x)| = \ell(x)\ \forall x \in \mathfrak{X}$.

```
Initialize ξ ← 1;
for x ∈ 𝔛 in order of nonincreasing ℓ(x) do
    Update ξ ← ξ − B^{−ℓ(x)};
    Write out ξ ∈ [0, 1) in its B-ary represenation: ξ = (0.???...)_B;
    Set C_ℓ(x) to the first ℓ(x) bits after the "0." in the above B-ary
     representation of ξ (pad with trailing zeros to length ℓ(x) if necessary);
```

Problem 2.2: Shannon Coding

In Problem 1.1 on the last problem set, you constructed Huffman codes C_H for three different probability distributions. The following table shows these codes for your reference. Throughout this problem, we assume $B = 2$.

x	$p(x)$	$C_\mathrm{H}(x)$	$C_\mathrm{S}(x)$	$p(x)$	$C_\mathrm{H}(x)$	$C_\mathrm{S}(x)$	$p(x)$	$C_\mathrm{H}(x)$	$C_\mathrm{S}(x)$
'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"	
$L_C =$		1.9			2.22			1.72	

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

$$H_2[p(x)] \leq L_{C_\mathrm{H}} < H_2[p(x)] + 1 \tag{2}$$

where L_{C_H} is the expected code word length of C_H, which is given in the last line of the above table (you already calculated these values on the last problem set).

(b) For each of the three probability distributions p, construct the Shannon code C_S 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). Verify explicitly that you get a prefix code in each example. Then calculate the expected code word length L_{C_S} of the Shannon code for each example and verify that

$$H_2[p(x)] \leq L_{C_\mathrm{H}} \leq L_{C_\mathrm{S}} < H_2[p(x)] + 1. \tag{3}$$

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

Problem 2.3: Entropy and Information Content

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

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

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

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

(a) In the literature, the subscript B is often dropped. Depending on context, information contents and entropies are usually understood to be either to base 2 (mostly in the data 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 (i.e., they both sum to 1) on $\mathfrak{X}_1$ and $\mathfrak{X}_2$, respectively (we will discuss statistical independence in more detail in Lecture 4). Show that if Eq. 6 holds, then both information contents and entropies are additive, i.e., in particular,

$$H_B\big[\tilde{p}\big] = H_B[p_1] + H_B[p_2] \quad \forall B > 0 \quad \text{(in case of statistical independence)}. \tag{7}$$

Note: you will prove on Problem Set 4 that, if we drop the restriction to statistical independence (Eq. 6), then entropies are *subadditive* in general, but no general statement can be made about the sum of two information contents.

Problem 2.4: Understanding Entropy: a Trivial Example

In this problem, we aim to gain some more intuition on why information content is defined the way it is (Eq. 4). To this end, we will consider a compression method that is so trivial that the word "compression" will almost seem like an overstatement in this context. Curiously, however, this trivial compression method turns out to be a natural starting point for understanding the modern and highly effective "Asymmetric Numeral Systems" (ANS) entropy coder, which we will discuss in Lecture 6.

The strategy that we use in this problem is something that you'll find useful for approaching many new topics, not just in this course: when trying to understand a complicated new concept, it is often a good idea to use act similarly as if you were *debugging code*: reduce the new concept to its absolute simplest form, try to understand it in this simple form, and then gradually build back up to the general form.

Problem Setup. Consider a data source that generates a sequence $(x_1, x_2, \ldots, x_{k(\mathbf{x})})$ of symbols from a *finite* alphabet $\mathfrak{X}$. Now, assume the simplest possible probability distribution p for the symbols: the *uniform distribution*, i.e., $p(x) = 1/|\mathfrak{X}| \ \forall x \in \mathfrak{X}$.

(a) What is the entropy $H_2[p(x)]$ per symbol (with base $B = 2$)?

(b) Take a step back from the problem setup and consider the binary representation of a positive integer $n \in \mathbb{N}$. How long is this binary representation, i.e., how many bits does it contain, assuming that there are no leading zeros? Express your result as a mathematical function of n and test it for $n \in \{1, 2, 3, 4, 5\}$ to make sure you don't have an off-by-one error.

(c) Back to the problem setup: combine your findings from parts (a) and (b) to come up with a trivial yet near-optimal prefix code $C_{\text{trivial}} : \mathfrak{X} \to \{0, 1\}^*$. The expected code word length $L_{C_{\text{trivial}}}$ should exceed the entropy $H_2[p(x)]$ by less than 1 bit.

Hint 1: no need to think about Huffman or Shannon coding; it's much simpler.

Hint 2: a trivial way of ensuring that a code book is prefix free is by making all code words $C_{\text{trivial}}(x)$ for $x \in \mathfrak{X}$ different in content but equal in length.

Problem 2.5: Implementing a Huffman Decoder

In Problem 1.3 of the last problem set, you implemented the Huffman coding algorithm in Python. Your Huffman tree there was optimized for *encoding*. For your reference, the accompanying jupyter notebook contains again the suggested solution to that exercise.

The notebook then guides you through the implementation of a Huffman tree that is optimized for *decoding*. Fill in the missing lines of code in the class `HuffmanDecoder` and test your implementation using the provided unit test. Then implement some round-trip tests as explained in the notebook.