Solutions to Problem Set 2

discussed:
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 2.1: 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 ξ (pad with trailing zeros to length $\ell(x)$ if necessary);

Solution: The variable ξ is initialized as $\xi \leftarrow 1$ and then never increased during the execution of the algorithm. When we come to line 4, ξ has been decreased by a strictly positive amount (since $B^{-\ell(x)} > 0$) at least once, thus ξ is strictly smaller than 1. Further, since ℓ satisfies Eq. 1, the entire `for` loop decreases ξ by a total of at most 1, and thus ξ never drops below zero. ■

(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 $C(x)$.

Solution: Since each step of the `for`-loop makes ξ smaller and since $\xi_{x'} > \xi_x$, the symbol x' must come *before* the symbol x. Since the `for` loop iterates in order of nonincreasing $\ell(x)$, this means that $\ell(x') \geq \ell(x)$. Therefore, the only way how $C(x')$ could be a prefix of $C(x)$ is if $\ell(x) = \ell(x')$ and $C(x) = C(x')$, in which case $C(x)$ is also a prefix of $C(x')$. Thus, we only have to prove that $C(x)$ is not a prefix of $C(x')$.

Each step of the algorithm reduces ξ by $B^{-\ell(x)}$. Thus, at the beginning of the iteration for symbol x, the variable ξ had value $\xi_x + B^{-\ell(x)}$, and all $\xi_{x'}$ for symbols x' that come before symbol x satisfy $\xi_{x'} \geq \xi_x + B^{-\ell(x)}$.

Now assume that $C(x)$ (which has length $\ell(x)$) is a prefix of $C(x')$. This means that the fractional parts of the B-ary expansions of ξ_x and of $\xi_{x'}$ agree on the first $\ell(x)$ digits. Thus, they are both in the interval $\left[(0.C(x))_B, (0.C(x))_B + B^{-\ell(x)}\right)$ and thus they differ by strictly less than $B^{-\ell(x)}$, which is a contradiction. ■

(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?

Solution: Prooving part (b) of the Kraft-McMillan Theorem doesn't require executing Algorithm 1 for the entire alphabet $\mathfrak{X}$. We only have to show that there *exists* a prefix code C with the requested code word lengths, $|C(x)| = \ell(x)$ for all $x \in \mathfrak{X}$. Such a prefix code is well defined: for each $x \in \mathfrak{X}$, the code word $C(x)$ is given by executing Algorithm 1 but terminating it once we've found $C(x)$. This takes finite time for any given $x \in \mathfrak{X}$, so it is well defined. ■

(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.

Solution: A lossy compresion code (such as a uniquely decodable symbol code) is an *injective* mapping from the message space to the space of finite-length bit strings. The space of finite-length bit strings is clearly countable, i.e., there exists

an injective mapping from the finite length bit strings to the set of the natural numbers (e.g., just prepend the bit string with a "1" bit and then interpret the resulting sequence of symbols as a number in the positional numeral system of base B). Thus, by chaining together the lossless compression code (which maps injectively from the message space to the space of finite-length bit strings) with the injective mapping from finite-length bit strings to natural numbers, we obtain an injective mapping from the message space to the natural numbers. Existence of such an injective mapping means that the message space is countable.

This argument may seem trivial but it has important consequences: while, strictly speaking, non-countable message spaces don't really exist in digital computing anyway, a lot of data that we might want to compress (e.g, scientific measurements, neural network weights, ...) is really meant approximate real-valued data, typically via floating point numbers. In such situations, theorems for lossless compression—while technically still valid—aren't typically very useful, and it is more important to think about bounds on lossy compression, which we'll discuss later in this course. ■

Problem 2.2: Shannon Coding

In Problem 1.1 on Problem Set 1 (formerly known as 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'	'01'	0.3	'00'	'10'	0.05	'000'	'11111'
'b'	0.3	'10'	'10'	0.28	'01'	'01'	0.07	'001'	'1110'
'c'	0.2	'110'	'110'	0.12	'100'	'1111'	0.12	'010'	'1101'
'd'	0.1	'111'	'1111'	0.1	'101'	'1110'	0.12	'011'	'1100'
'e'	–	–	–	0.2	'11'	'110'	0.64	'1'	'0'
	1.85	$L=1.9$	$L=2.4$	2.20	$L=2.22$	$L=2.64$	1.63	$L=1.72$	2.13

(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.

Solution: See last entries in the three columns labeled $p(x)$ in the above table. ■

(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

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

Solution: See filled-in entries in above table. Note that you might obtain slightly different Shannon codes depending on the order in which you iterate over symbols of equal code word lengths. But your Shannon codes should all be prefix free and you should get the same code word lengths. ■

(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?

Solution: To solve this problem, we don't have to think about Huffman coding at all. It suffices to find a probability distribution p where $L_{\text{Shannon}} = H_2[p]$. Since we know that $H_2[p] \leq L_{\text{Huff}} \leq L_{\text{Shannon}}$ for all probability distributions p, having $L_{\text{Shannon}} = H_2[p]$ implies also $L_{\text{Huff}} = H_2[p]$.

The Shannon code for a probability distribution p has code words with lengths $\ell_{\text{Shannon}}(x) = \lceil -\log_B p(x) \rceil$. Thus, $L_{\text{Shannon}} = H_2[p]$ means $\mathbb{E}_p\big[\lceil -\log_B p(x) \rceil\big] = \mathbb{E}_p[-\log_B p(x)]$. Since $\lceil -\log_B p(x) \rceil \geq -\log_B p(x)$ for all x, the two expectations are equal if and only if the information content, $-\log_B p(x)$, of every symbol x is an integer (so that rounding it up is a no-op). In other words, all symbol probabilities $p(x)$ must be negative integer powers of B.

For example, we can start from the code word lengths of C_{Huff} in the second example above ($\ell(\text{'a'}) = 2$, $\ell(\text{'b'}) = 2$, $\ell(\text{'c'}) = 3$, $\ell(\text{'d'}) = 3$, and $\ell(\text{'e'}) = 2$). Then, we set $p(x) = 2^{-\ell(x)}$ for all x, i.e., $p(\text{'a'}) = \frac{1}{4}$, $p(\text{'b'}) = \frac{1}{4}$, $p(\text{'c'}) = \frac{1}{8}$, $p(\text{'d'}) = \frac{1}{8}$, and $p(\text{'e'}) = \frac{1}{4}$. These probabilities do indeed add up to one, as they should for a properly normalized probability distribution, and we have (by construction), $\lceil -\log p(x) \rceil = -\log p(x)$ for all x, and thus $L_{\text{Shannon}} = H_2[p]$. ■

Problem 2.3: 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?

Solution: Since $\log_B \alpha = \frac{\ln \alpha}{\ln B}$ for all $B, \alpha > 0$ (where ln denotes the natural logarithm to base e), we have

$$-\log_2 p(x) = \frac{-\ln p(x)}{\ln 2}; \qquad \text{and} \qquad H_2[p] = \frac{H_e[p]}{\ln 2} \tag{6}$$

where $\ln 2 \approx 0.69$ (or $\frac{1}{\ln 2} \approx 1.44$). ■

(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,

$$\tilde{p}\big((x_1, x_2)\big) = p_1(x_1)\, p_2(x_2) \qquad \forall x_1 \in \mathfrak{X}_1, x_2 \in \mathfrak{X}_2 \tag{7}$$

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,

$$H_B[\tilde{p}] = H_B[p_1] + H_B[p_2] \qquad \forall B > 0. \tag{8}$$

Solution: Additivity of information contents follows directly from Eq. 7 and the property of the logarithm, $\log_B(\alpha\beta) = \log_B \alpha + \log_B \beta$. For the entropy, we find:

$$\begin{aligned}
H_B[\tilde{p}] &= -\sum_{(x_1,x_2)\in\mathfrak{X}^2} \tilde{p}\big((x_1, x_2)\big)\, \log_B \tilde{p}\big((x_1, x_2)\big) \\
&= -\sum_{x_1\in\mathfrak{X}} \sum_{x_2\in\mathfrak{X}} p_1(x_1)\, p_2(x_2)\, \big[\log_B p_1(x_1) + \log_B p_2(x_2)\big] \\
&= -\Big(\sum_{x_2\in\mathfrak{X}} p_2(x_2)\Big) \sum_{x_1\in\mathfrak{X}} p_1(x_1) \log_B p_1(x_1) \\
&\quad - \Big(\sum_{x_1\in\mathfrak{X}} p_1(x_1)\Big) \sum_{x_2\in\mathfrak{X}} p_2(x_2) \log_B p_2(x_2) \\
&= -\sum_{x_1\in\mathfrak{X}} p_1(x_1) \log_B p_1(x_1) - \sum_{x_2\in\mathfrak{X}} p_2(x_2) \log_B p_2(x_2) \\
&= H_B[p_1] + H_B[p_2]
\end{aligned}$$

■

Problem 2.4: 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

$$H_B[p] \leq L_{\text{opt}} < H_B[p] + 1. \tag{9}$$

So far, Eq. 9 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. 9 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. 8-9 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. 9, but the upper bound shrinks, i.e., the range of possible values narrows.

Solution: Let $L^{(m)}_{\text{opt}}$ be the expected code word length of $\tilde{C}^{(m)}_{\text{opt}}$. Since $\tilde{C}^{(m)}_{\text{opt}}$ is an optimal code on the product alphabet $\mathfrak{X}^m$ with probability distribution $\tilde{p}$, Eq. 9 applies and we have:

$$H_B[\tilde{p}] \leq L^{(m)}_{\text{opt}} < H_B[\tilde{p}] + 1. \tag{10}$$

Using Eq. 8 for the factorized probability distribution $\tilde{p}$, we further find $H_B[\tilde{p}] = mH_B[p]$ (where p is the probability distribution for each individual symbol from $\mathfrak{X}$). Thus, the expected number of bits *per original symbol from* $\mathfrak{X}$ of the code $\tilde{C}^{(m)}_{\text{opt}}$, i.e., $\frac{L_{\text{opt}}}{m}$, satisfies:

$$H_B[p] \leq \frac{L_{\text{opt}}}{m} < H_B[p] + \frac{1}{m}. \tag{11}$$

Thus, the fundamental lower bound does not change compared to Eq. 9, but the upper bound on the overhead shrinks from one bit per symbol to $\frac{1}{m}$ bits per symbol. ■

(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.

Solution: For long messages, $m = k \gg 1$, the upper bound of $\frac{1}{k}$ bits per symbol on the compression overhead becomes negligibly small. Thus, for long messages, (i) there exists a lossless compression code whose expected bit rate per (original) symbol is essentially the entropy $H_B[p]$ (up to a very small overhead). However, (ii) constructing this optimal code with, e.g., Huffman coding would require exponential runtime $O(|\mathfrak{X}^k|) = O(|\mathfrak{X}|^k)$, which is completely infeasible for even moderately large message lengths k. This exponential run time is necessary because the Huffman algorithm constructs a tree with $O(|\mathfrak{X}^k|)$ nodes, and there's no obvious way how to obtain the Huffman code for only a single message $\mathbf{x} \in \mathfrak{X}^k$ of interest without constructing the entire tree for all possible messages in $\mathfrak{X}^k$.

If we used Shannon coding instead of Huffman coding, then the Shannon code $C^{(k)}_{\text{Shannon}}$ on blocks of size $m = k$ would strictly speaking no longer be an optimal code, but its overhead per symbol would still be negligible (i.e., less than $\frac{1}{k}$). The advantage of the Shannon algorithm is that one doesn't have to construct the whole code book for $C^{(k)}_{\text{Shannon}}$ explicitly; instead, when encoding a message $\mathbf{x} \in \mathfrak{X}^k$, it suffices to construct only the encoding $C^{(k)}_{\text{Shannon}}(\mathbf{x})$ of this particular message $\mathbf{x}$. A variant of Shannon coding, called Arithmetic Coding, can encode a single message in linear time $O(k)$. We will discuss Arithmetic Coding (as well as other so-called stream codes) later in the course. ■