CS3236 Lecture Notes #2: Symbol-Wise Source Coding

Jonathan Scarlett

February 8, 2024

Useful references:

- Cover/Thomas Chapter 5
- MacKay Chapter 5

1 Symbol-Wise Coding

Setup.

- Consider a discrete random variable X with probability mass function (PMF) P_X.
  - For example, for text (without spaces/punctuation) X might take one of 26 characters $\{a, \dots, z\}$, with $P_X(e)$ being highest, $P_X(q)$ being low, etc.
  - More generally, the set of all symbols is denoted by $\mathcal{X}$ (called the "alphabet" even when not referring to the English alphabet or even text).
- Symbol-wise source coding maps each $x \in \mathcal{X}$ to some binary sequence $C(x)$. The length of this sequence is denoted by $\ell(x)$.
  - e.g., Map 'x' to 0011010 and 'q' to 0010100 because they are uncommon symbols, map 'e' to 1 because it is a very common symbol.

  Since having a small length is so fundamental, we provide the following formal definition.

- **Definition.** The average length of a code $C(\cdot)$ is given by

$$L(C) = \sum_{x \in \mathcal{X}} P_X(x) \ell(x).$$

- We need to be able to map back from the binary sequences to the original alphabet, so we cannot make every binary sequence short!
- Let's look at the decoding conditions we would like to have.

Decodability conditions.

- To have any hope of mapping the binary sequences back to the original alphabet, we need that $C(x) \neq C(x')$ whenever $x \neq x'$. This condition is so trivial that it doesn't really need a name, but sometimes it's called the *nonsingular* property.

- We are actually interested in coding multiple alphabet symbols in succession, so being nonsingular is not enough. Consider the following code for $\mathcal{X} = \{1, 2, 3, 4\}$:

$$\begin{aligned} a &\to 0 \\ b &\to 1 \\ c &\to 00 \\ d &\to 11 \end{aligned}$$

  Clearly there is no way to distinguish the sequence 'aabb' from 'cd'.

- **Definition.** A code $C(\cdot)$ is said to be *uniquely decodable* if no two sequences (of equal or differing lengths) of symbols in $\mathcal{X}$ are coded to the same concatenated binary sequence. That is, $x_1, \ldots, x_n$ can always uniquely be identified from the string $C(x_1) \ldots C(x_n)$.

- **Example.** The following code is uniquely decodable:

$$\begin{aligned} a &\to 1 \\ b &\to 10 \\ c &\to 100 \\ d &\to 1000 \end{aligned}$$

  While unique decodability is easy to see in this example, it can be tricky to verify in larger codes. It is more convenient to work with the following *seemingly* more restrictive condition.

- **Definition.** A code $C(\cdot)$ is said to be *prefix-free* if no codeword is a prefix of any other (i.e., it is not possible to append more bits to some $C(x)$ in order to produce some other $C(x')$).
  - Sometimes the terminology *instantaneous code* is used to mean the same thing.
  - It turns out (we will omit the proof) that restricting to prefix-free codes instead of general uniquely decodable codes does not lose us anything in the average length we can achieve.
  - However, while decoding uniquely decodable codes is challenging in general, decoding prefix-free codes is trivial: As soon as the binary sequence matches some $C(x)$, output x, and then iteratively continue with the rest of the binary sequence.
  - We will therefore focus primarily on prefix-free codes.

- **Example.** The following code is prefix-free:

$$\begin{aligned} a &\to 0 \\ b &\to 10 \\ c &\to 110 \\ d &\to 111 \end{aligned}$$

Here is a simple example of decoding an encoded sequence for this code:

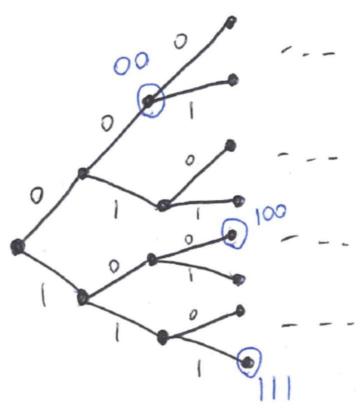
$$\text{cab} \to \underbrace{11\,0}\,\underbrace{0}\,\underbrace{10}$$
$$\text{abba} \to \underbrace{0}\,\underbrace{10}\,\underbrace{10}\,\underbrace{0}$$
$$\text{dad} \to \underbrace{111}\,\underbrace{0}\,\underbrace{111}$$

- **Note:** We will focus on binary codes (which are by far the most common), but the vast majority of what we will cover can be easily extended to D-ary codes for $D > 2$ (e.g., for $D = 3$, we have ternary codes with digits $\{0, 1, 2\}$, so we can have mappings like $b \to 21$, $q \to 0121$, $z \to 2222$).

2 Kraft's Inequality and the Entropy Bound

Kraft's Inequality.

- Clearly not all possible combinations of lengths are possible, e.g., we cannot map every letter $a \ldots z$ to a sequence of length 3 or less – there are not enough such sequences. Even when there are enough sequences, not all allocations of symbols to sequences will be prefix-free (or uniquely decodable). Kraft's inequality gives a useful condition that any prefix-free code must satisfy.
- **Theorem (Kraft's inequality).** Any prefix-free code $C(\cdot)$ that maps each $x \in \mathcal{X}$ to a codeword of length $\ell(x)$ must satisfy
$$\sum_{x \in \mathcal{X}} 2^{-\ell(x)} \leq 1.$$
- **Proof**:
  - Represent the codewords by a binary tree as follows:



  - By the prefix-tree assumption, if there is a codeword at some point in the tree, there are no codewords further down the tree.
  - Now, consider starting at the root and then repeatedly branching either way with probability $\frac{1}{2}$ each until a codeword is hit.
  - Clearly, the probability of a given length-$\ell(x)$ codeword being hit is exactly $2^{-\ell(x)}$.

- But the total probability of hitting codewords cannot exceed one, so summing up all the probabilities gives $\sum_{x\in\mathcal{X}} 2^{-\ell(x)} \le 1$.

- **Theorem (Existence property).** If a given set of integers $\{\ell(x)\}_{x\in\mathcal{X}}$ satisfies $\sum_{x\in\mathcal{X}} 2^{-\ell(x)} \le 1$, then it is possible to construct a prefix-free code that maps each $x \in \mathcal{X}$ to a codeword of length $\ell(x)$.
  - Proof outline: Essentially proved by choosing codewords of a suitable length on a tree like the one shown above, starting with those having the smallest length. When $\sum_{x\in\mathcal{X}} 2^{-\ell(x)} \le 1$, we never "run out of space" on the tree; see the relevant tutorial question for details.
  - Hence, the condition in Kraft's inequality is both necessary and sufficient for the existence of a prefix-free code having such lengths.

Entropy bound.

- **Theorem.** For $X \sim P_X$ and any prefix-free code $C(\cdot)$, the expected length satisfies

$$L(C) \ge H(X),$$

  with equality if and only if $P_X(x) = 2^{-\ell(x)}$ for all $x \in \mathcal{X}$.
  - Hence, entropy provides a fundamental limit – we can never get an average length smaller than the entropy using a prefix-free code.
  - Even though we are only stating/proving it for the prefix-free case, it can be shown that the same holds for any uniquely decodable code.

- **Proof.**
  - Recall the definition of KL divergence, $D(P\|Q) = \sum_x P(x)\log_2 \frac{P(x)}{Q_X(x)}$.
  - Observe that

$$\begin{aligned} L(C) - H(X) &\overset{(a)}{=} \sum_x P_X(x)\ell(x) - \sum_x P_X(x)\log_2 \frac{1}{P_X(x)} \\ &\overset{(b)}{=} \sum_x P_X(x)\log_2 2^{\ell(x)} - \sum_x P_X(x)\log_2 \frac{1}{P_X(x)}, \end{aligned}$$

    where (a) is by definition, and (b) simply uses $c = \log_2(2^c)$.
  - To simplify notation, let $Z = \sum_{x\in\mathcal{X}} 2^{-\ell(x)}$, and define $Q_X(x) = \frac{2^{-\ell(x)}}{Z}$ which is a valid PMF (i.e., has non-negative values summing to one).
  - Re-arranging terms in the definition of $Q_X(x)$ gives $2^{\ell(x)} = \frac{1}{Z\cdot Q_X(x)}$, and substitution into the above equation gives

$$\begin{aligned} L(C) - H(X) &= \sum_x P_X(x)\log_2 \frac{1}{ZQ_X(x)} - \sum_x P(x)\log_2 \frac{1}{P_X(x)} \\ &\overset{(a)}{=} \log_2 \frac{1}{Z} + \sum_x P_X(x)\log_2 \frac{P_X(x)}{Q_X(x)} \\ &\overset{(b)}{=} \log_2 \frac{1}{Z} + D(P_X\|Q) \\ &\overset{(c)}{\ge} 0, \end{aligned}$$

where (a) uses simple re-arranging (and $\log_2 \frac{1}{\alpha} = -\log_2 \alpha$), (b) uses the definition of KL divergence, and (c) uses $Z \le 1$ (by Kraft's inequality) and $D(P_X \| Q_X) \ge 0$ (KL divergence between two PMFs is always non-negative).

- To get $L(C) = H(X)$, we need both $Z \le 1$ and $D(P_X \| Q_X) \ge 0$ to hold with equality. The former condition gives $Q_X(x) = 2^{-\ell(x)}$ (see the definition of Q_X), and the latter gives $P_X = Q_X$ (as established in the previous lecture), so overall we require $P_X(x) = 2^{-\ell(x)}$ for all x.

- **Implication.** If our probabilities all contain powers of two ($\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, etc.), we can bring the average code length all the way down to the entropy.
  - A simple example:

SYMBOL	PROBABILITY	CODEWORD
a	$\frac{1}{2}$	0
b	$\frac{1}{4}$	10
c	$\frac{1}{8}$	110
d	$\frac{1}{16}$	1110
e	$\frac{1}{16}$	1111

  - Notice that the lengths satisfy $\ell(x) = \log_2 \frac{1}{P_X(x)}$ for all $x \in \{a, b, c, d, e\}$

3 Shannon-Fano Code

- Based on the "...equality if and only if..." statement in the entropy bound theorem, we can think of $\ell^*(x) = \log_2 \frac{1}{P_X(x)}$ as being the "ideal" code length. However, it can only be attained when all values of $\frac{1}{P_X(x)}$ are powers of two.

- The **Shannon-Fano code** simply rounds the ideal lengths up to the nearest integer:

$$\ell(x) = \left\lceil \log_2 \frac{1}{P_X(x)} \right\rceil,$$

where $\lceil \cdot \rceil$ is the ceiling operation (i.e., rounding up).

- These lengths satisfy the conditions of the "Existence property" theorem above, since

$$\sum_{x \in \mathcal{X}} 2^{-\ell(x)} = \sum_{x \in \mathcal{X}} 2^{-\lceil \log_2 \frac{1}{P_X(x)} \rceil} \le \sum_{x \in \mathcal{X}} 2^{-\log_2 \frac{1}{P_X(x)}} = \sum_{x \in \mathcal{X}} P_X(x) = 1,$$

where we used $\lceil \alpha \rceil \ge \alpha$ and $-\log_2 \frac{1}{\alpha} = \log_2 \alpha$. Hence, that theorem implies that we can indeed construct a prefix-free code with the above lengths.

- **Theorem.** The average length $L(C)$ of the Shannon-Fano code satisfies

$$H(X) \le L(C) < H(X) + 1,$$

so is within one bit of the best average length possible.

  - Proof: The lower bound is just a repetition of the entropy bound. To prove the upper bound, we use the fact that $\lceil \alpha \rceil < \alpha + 1$ to deduce the following:

$$\begin{aligned} L(C) &= \sum_{x \in \mathcal{X}} P_X(x) \ell(x) \\ &= \sum_{x \in \mathcal{X}} P_X(x) \left\lceil \log_2 \frac{1}{P_X(x)} \right\rceil \\ &< \sum_{x \in \mathcal{X}} P_X(x) \left(\log_2 \frac{1}{P_X(x)} + 1 \right) \\ &= H(X) + 1, \end{aligned}$$

    where the last step uses the definition of entropy and $\sum_{x \in \mathcal{X}} P_X(x) = 1$. (Note: By following similar steps but instead applying $\lceil \alpha \rceil \geq \alpha$, we get an alternative proof of the lower bound.)

- While the addition of at most 1 bit may seem innocuous, it can be very significant (e.g., for a "low-information" source, maybe $H(X)$ itself is only 0.5 bits!)

- **Theorem (Mismatched case).** If the true distribution is P_X but the lengths are chosen according to Q_X (i.e., $\ell(x) = \lceil \log_2 \frac{1}{Q_X(x)} \rceil$), then the Shannon-Fano code satisfies

$$H(X) + D(P_X \| Q_X) \leq L(C) < H(X) + D(P_X \| Q_X) + 1.$$

  - Proof: Similar to above, also using $\mathbb{E}_P\big[\log_2 \frac{1}{Q_X(X)}\big] = \mathbb{E}_P\big[\log_2 \frac{P_X(X)}{Q_X(X) P_X(X)}\big] = \mathbb{E}_P\big[\log_2 \frac{1}{P_X(X)} + \log_2 \frac{P_X(X)}{Q_X(X)}\big] = H(X) + D(P_X \| Q_X)$.
  - Hence, if an inaccurate distribution is used (not-so-small $D(P_X \| Q_X)$) then the penalty due to mismatch may also be significant.

4 Huffman Code

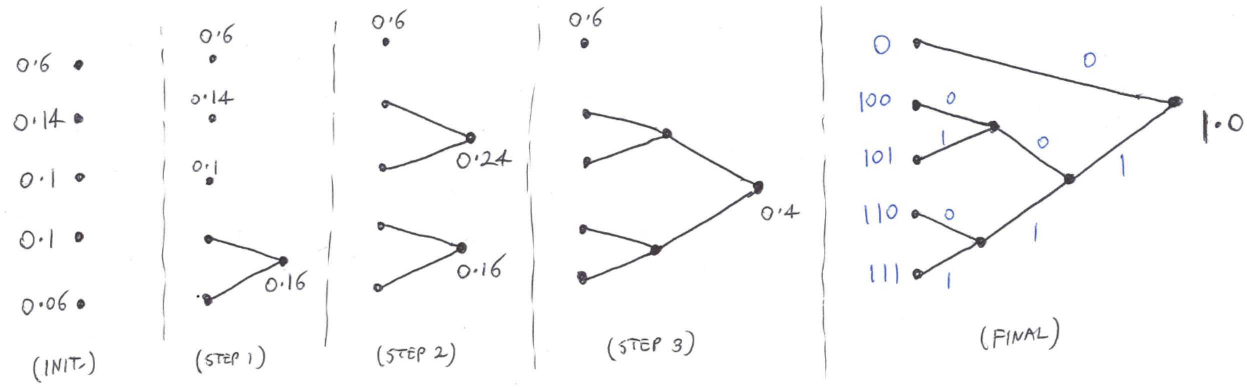
- At this stage it is natural to ask whether it is possible to find the *optimal* symbol code, in the sense of minimizing $L(C)$ while being uniquely decodable. This remained a seemingly challenging open problem until an *extremely simple* solution was given by Huffman (as part of a homework question!).

- **Huffman code.** Construct a tree as follows:

  - List the symbols of $\mathcal{X}$ from highest probability from highest to lowest.
  - Draw a branch connecting the two symbols with the lowest probability, and label the merged point with the sum of the two associated probabilities.
  - Repeat the previous step (with the two original probabilities replaced by the merged probability) until everything has merged to a single point with total probability 1.

Once this tree is constructed, we label the two edges in each branch as 0 and 1, and then let the codewords be the labels encountered when traversing from the end back to the start.

- An illustration:



- **Theorem.** No uniquely decodable symbol code can achieve a smaller average length $L(C)$ than the Huffman code.
  - The proof is based on a recursive argument, and can be found in Section 5.8 of Cover/Thomas.
- Since Huffman coding is at least as good as Shannon-Fano coding, it also satisfies the average length bound $H(X) \leq L(C) < H(X) + 1$.

5 Discussion

- Perhaps the most significant limitation of symbol codes is that they do not exploit *memory* (i.e., dependence between subsequent symbols). For instance, given a 'q' in English the next character is very likely to be 'u', or to go even further, we can be very confident about the next character in the sequence "Fill in the blan_".
- However, we can extend symbol codes to exploit such dependencies as follows:
  - Instead of coding one character at a time, group them into chunks of a given length (say, 5 symbols) and do Huffman coding with $\mathcal{X}^5$ in place of $\mathcal{X}$, and $P_{X_1X_2X_3X_4X_5}$ in place of P_X.
  - Advantage 1: We can exploit the fact that, for example, "apple" is a much more probable sequence than "ealpp" despite containing the same characters.
  - Advantage 2: Even with independent symbols (e.g., $P_{X_1X_2X_3X_4X_5} = \prod_{i=1}^5 P_X(x_i)$) we can improve the "one bit extra" guarantee. More generally letting $N = 5$ denote the length we are coding over, the standard one bit guarantee gives the following under independent symbols:

$$NH(X) \leq L(C) \leq NH(X) + 1,$$

    since $H(X_1, \dots, X_N) = \sum_{i=1}^N H(X_i)$ for independent symbols. But by dividing by N, this gives

$$H(X) \leq \text{Average length per symbol} \leq H(X) + \frac{1}{N},$$

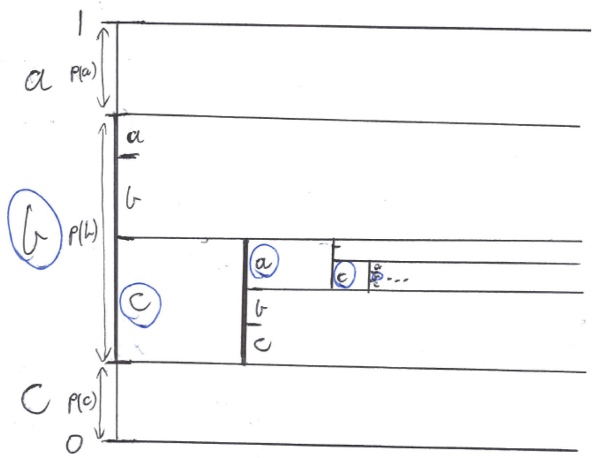
    so that the loss of 1 bit in the upper bound is improved to $\frac{1}{N}$.
  - Disadvantage 1: Determining the distribution $P_{X_1,\dots,X_N}$ accurately is very difficult.

– Disadvantage 2: Even if the joint distribution is known, sorting $|\mathcal{X}|^N$ probabilities in the Huffman coding algorithm becomes computationally challenging even for moderate values of N. The complexity increases exponentially with N.

6 (Optional) Beyond Symbol-Wise Codes

Arithmetic codes

- Arithmetic codes are a very elegant technique for sequentially coding sources with memory when the conditional distribution $P_{X_i|X_1,\ldots,X_{i-1}}$ is known. The idea is illustrated in the following:



  – For concreteness, suppose the alphabet is ‘a’, ‘b’, ‘c’.
  – Start with an interval ranging from 0 to 1.
  – Split the interval into three regions of width $P_{X_1}(a)$, $P_{X_1}(b)$, and $P_{X_1}(c)$.
  – After observing $X_1 = b$, move into the region of width $P_{X_1}(b)$.
  – Split the width-$P_{X_1}(b)$ region into three regions proportional to $P_{X_2|X_1}(a)$, $P_{X_2|X_1}(b)$, and $P_{X_2|X_1}(c)$.
  – After observing $X_2 = c$, move into the corresponding sub-region.
  – Continue recursively until the entire input sequence has been read.

- At the end of this process, we are left with a very small sub-interval $\mathcal{I}$ of $[0,1]$. How do we map this to a binary sequence?

- **Key idea.**

  – Every point in the interval $[0,1]$ corresponds to an infinite binary sequence (e.g., $\frac{1}{3}$ maps to $0.010101\ldots$, $\frac{1}{2}$ maps to $0.10000\ldots$, etc.).
  – A finite-length binary sequence (e.g., 0.010101) then corresponds to an *interval* (e.g. starting from the number represented by 0.010101 followed by infinitely many zeros, ending at the number 0.010101 followed by infinitely many ones).

– Output a finite-length binary sequence that is just long enough for its corresponding interval to be a sub-interval of $\mathcal{I}$.

- The notion of entropy as a fundamental compression limit can be extended to broad types of sources with memory (see Cover/Thomas Chapter 4), and it can be shown that arithmetic coding can encode a sequence $(x_1, \dots, x_n)$ down to at most

$$\ell(x_1, \dots, x_n) \le \log_2 \frac{1}{P_{X_1,\dots,X_n}(x_1, \dots, x_n)} + 2$$

bits, which is within 2 bits of the "ideal length". In particular, we get

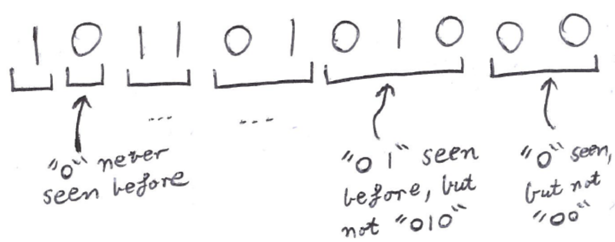
$$\text{Average Total Length} \le H(X_1, \dots, X_n) + 2,$$

and for a memoryless source, the number of bits per symbol is at most $H(X) + \frac{2}{n}$.

- See Cover/Thomas Section 13.3 or MacKay Section 6.2 for further details.

Lempel-Ziv code

- A disadvantage of arithmetic codes is the need to know $P_{X_1,\dots,X_n}$. A class of codes known as *Lempel-Ziv* (LZ) codes are *universal*, in that they do not use any knowledge of the source distribution.
- For memoryless sources, and also broad classes of sources with memory, LZ codes are efficient enough to code almost down to the entropy (albeit with the "second-order" term being $O(\log n)$, which is a fair bit higher than the 2 bits attained by arithmetic coding).
- The encoding can roughly be described as follows:
  - Step 1: Parse the string into substrings that haven't been observed earlier:



  - Step 2: Encode each parsed sub-string into a sequence of bits of the form (pointer, new bit), where "pointer" identifies the index of the sub-string that matches the current one with the final bit removed, and "new bit" describes that final bit.
  - Encoding: Store the sequence of pointers (represented in binary) and new bits.
- See Cover/Thomas Section 13.4 or MacKay Section 6.4 for further details.