Stochastic Complexity via Two Part Codes

Dror Baron

This supplement provides more details about Question 2 in Homework 1.

General setting: The aim of this question is to demonstrate numerically that model classes should contain $O(\sqrt{N})$ representation levels per parameter. This result provides plenty of insights about the stochastic complexity. In particular, because coding length has the following relation to probability,

$$len = -\log_2(p), \tag{1}$$

then we can think of each parameter as requiring roughly $\frac{1}{2}\log_2(N)$ bits. With K parameters, we will need $\frac{K}{2}\log_2(N) + O(K)$ bits.

To show this result numerically, we consider a two part code. To keep things simple, we discuss length-N strings over a binary alphabet, i.e., $X \in \{0,1\}^N$. We will now construct a *two part code*. A two part code contains two parts.

1. **The first part estimates the parameter(s) governing the data, and encodes them.** In our example, we will model the bits in X as independent and identically distributed (i.i.d.), which you can think of as biased coin tosses. Each bit X_n, $n \in \{1,\ldots,N\}$, will have value 1 with probability θ and value 0 with probability $1-\theta$. However, the parameter θ is unknown. Therefore, we estimate θ by first counting the number of ones and zeros in the data as follows,

$$N_0 = \sum_{n=1}^{N} 1_{\{X_n=0\}} \quad \text{and} \quad N_1 = \sum_{n=1}^{N} 1_{\{X_n=1\}},$$

where $1_{\{\cdot\}}$ is an indicator function that takes the value 1 when the condition is true, else 0, and then we divide N_1 by N to obtain the estimated parameter $\widehat{\theta}$,

$$\widehat{\theta} = \frac{N_1}{N_0+N_1} = \frac{N_1}{N}.$$

It is readily seen that $1-\widehat{\theta} = \frac{N_0}{N}$. Having computed $\widehat{\theta}$, it would be nice to use it to encode (compress) X. However, we are at the encoder, and we know $\widehat{\theta}$, but the decoder does not know its value, and we must encode $\widehat{\theta}$ somehow. Note that there are $N+1$ possible values of $\widehat{\theta}$, because $N_0, N_1 \in \{0,1,\ldots,N\}$. So in principle we could encode $\widehat{\theta}$ precisely using $\log_2(N+1)$ bits. However, this question will demonstrate that it is better to use fewer representation levels.

Consider K representation levels. The quantizer will partition $\widehat{\theta}$ into one of K bins, where Bin 1 corresponds to $\widehat{\theta} \in [0, \frac{1}{K})$, Bin 2 corresponds to $\widehat{\theta} \in [\frac{1}{K}, \frac{2}{K})$, up to Bin K, which corresponds to $\widehat{\theta} \in [\frac{K-1}{K}, \frac{K}{K}] = [1 - \frac{1}{K}, 1]$. Once we have identified bin $k \in \{1, \ldots, K\}$, we encode k using $\log_2(K)$ bits. Finally, bin k corresponds to a representation level (also known as the quantization level) in the middle of the bin. It is easily seen that representation level r_k in bin k takes the value

$$r_k = \frac{k - \frac{1}{2}}{K}.$$

Note that we have used a uniform quantizer, and moreover having representation levels in the middle simplifies our implementation. It can be seen that non-uniform quantization of $\widehat{\theta}$ can have some beneficial properties; particularly eager students may want to browse through B. S. Clarke and A. R. Barron, "Jeffreys' prior is asymptotically least favorable under entropy risk," J. Statistical Planning Inference, 1994 or J. Rissanen, "Fisher information and stochastic complexity," IEEE Trans. Information Theory, 1996. (This direction may even be good for an individual project.)

2. **The second part compresses the string X with probablity model r_k.** When the compressor will encounter a 0 in X, i.e., x_0, it will assign probability $1 - r_k$ to x_n. Similarly, when $x_n = 1$ the compressor will assign probability r_k. Therefore, when $x_n = 0$ the coding length will be $-\log_2(1 - r_k)$ bits, and when $x_n = 1$ it will be $-\log_2(r_k)$ bits. The total coding length for compressing X in the second part of the two part code using quantized parameter r_k will be

$$\begin{aligned} len_{r_k}(X) &= -\sum_{n=1}^{N} \left[1_{\{x_n=0\}} \cdot \log_2(1 - r_k) + 1_{\{x_n=1\}} \cdot \log_2(r_k)\right] \\ &= -N_0 \log_2(1 - r_k) - N_1 \log_2(r_k) \\ &= -N(1 - \widehat{\theta}) \log_2(1 - r_k) - N\widehat{\theta} \log_2(r_k). \end{aligned} \tag{2}$$

At this point, the astute student may have noticed that $\log_2(r_k)$ and $\log_2(1 - r_k)$ may be non-integer. For example, if $r_k = 0.25$, then $-\log_2(r_k) = 2$ bits, but $-\log_2(1-r_k) = 0.415$ bits, which is non-integer. While it is probably clear how a 1 can be encoded using 2 bits, it is not clear how a 0 could be encoded using 0.415 bits. It turns out that a well-known technique called *arithmetic coding* can process such problems. Arithmetic coding proceeds symbol by symbol; first it processes x_1, then x_2, and so on. As it processes each bit, it outputs bits in order to allow the decoder to decode the bits that have appeared at the input so far. Therefore, while some x_n may yield multiple bits at the output, other values may not yield any additional bits at the output. This happens when x_n has relatively high probability (in our case, the probability for 0 was 0.75). An early paper about arithmetic coding is J. Rissanen and G. Langdon, "Arithmetic coding," IBM J. Research Development, 1978. A well written tutorial is I. H. Witten, R. M. Neal, and J. G, Cleary, "Arithmetic coding for data compression," Comm. ACM, 1987. (Again, arithmetic coding could certainly feed into an individual project.) The bottom line is that once the arithmetic encoder has processed all of X,

it outputs $-\log(\Pr(X)) + O(1)$ bits, where the $O(1)$ term is often 2–3 bits. Therefore, practically speaking the coding length $len_{r_k}(X)$ of the second part of the two part code can be achieved.

It might add intuition to think in terms of the probability for the entire sequence X,

$$\Pr(X) = \prod_{n=1}^{N} \Pr(x_n),$$

where the equality is due to symbols of x_n being independent. Because the coding length is proportional to the log of the probability (1),

$$len(X) = -\log_2(\Pr(X)) = -\sum_{n=1}^{N} \log_2(\Pr(x_n)),$$

which corresponds to (2).

Having developed our two part code and discussed it in detail, what is the entire coding length of both parts? Recall that the first part encodes $\log_2(K)$ bits, and the second encodes $len_{r_k}(X) = -N(1-\widehat{\theta})\log_2(1-r_k) - N\widehat{\theta}\log_2(r_k)$ bits (2). Therefore, the total coding length is

$$\log_2(K) - N(1-\widehat{\theta})\log_2(1-r_k) - N\widehat{\theta}\log_2(r_k).$$

In contrast, suppose that we somehow knew the true parameter θ. In that case, we could compress X at its entropy,

$$H(\theta) = -\theta\log_2(\theta) - (1-\theta)\log_2(1-\theta).$$

We do not know θ, of course. But the encoder knows $\widehat{\theta}$, which can be interpreted as the empirical parameter. The decoder only knows a quantized version of $\widehat{\theta}$, but suppose somehow that it could compress X with any parameter. It can be shown that the lowest coding length is obtained using $\widehat{\theta}$, resulting in coding length $NH(\widehat{\theta})$. This coding length serves as a best-possible benchmark.

As a quick example, suppose that $\widehat{\theta} = 0.2$, $N = 1000$, and $r_k = 0.21$. Because $N = 1000$, we have $N_1 = N\widehat{\theta} = 200$ and $N_0 = N - N_1 = 800$. The entropy is

$$H(0.2) = -0.2\log_2(0.2) - 0.8\log_2(0.8) = 0.7219,$$

yielding the benchmark coding length $NH(0.2) = 721.9$ bits. It can be seen that $NH(\widehat{\theta}) = len_{\widehat{\theta}}(X)$. However, our arithmetic coder compresses with quantized representation level r_k, and

$$len_{r_k}(X) = -800\log_2(1-0.21) - 200\log_2(0.21) = 722.4$$

bits. It can be seen that the quantized representation level slightly increases the coding length beyond the benchmark entropy.

Finally, we compute the *redundancy* or overhead of the two part code by subtracting the benchmark coding length from the total coding length of both parts,

$$\begin{aligned} R(X,\widehat{\theta}) &= \left[\log_2(K) - N(1-\widehat{\theta})\log_2(1-r_k) - N\widehat{\theta}\log_2(r_k)\right] - \left[NH(\widehat{\theta})\right] \\ &= \log_2(K) + N(1-\widehat{\theta})[-\log_2(1-r_k) + \log_2(1-\widehat{\theta})] + N\widehat{\theta}[-\log_2(r_k) + \log_2(\widehat{\theta})] \\ &= \log_2(K) + N(1-\widehat{\theta})\log_2\left(\frac{1-\widehat{\theta}}{1-r_k}\right) + N\widehat{\theta}\log_2\left(\frac{\widehat{\theta}}{r_k}\right). \end{aligned}$$

Examining the redundancy for various values of $\widehat{\theta}$ and K will show that $K = \Theta(\sqrt{N})$ has reasonably low redundancy levels. In particular, the first part of the code expends $\frac{1}{2}\log_2(N) + O(1)$ bits to encode the bin index k, and the second part expends $O(1)$ bits above the benchmark coding length.