Minimum Description Length and Model Complexity

Dror Baron

This document will describe how the minimum description length (MDL) principle (Section 4) can be used in conjunction with two part codes (Section 5) to learn the data's probabilistic properties. While we will mostly discuss things through the lens of data compression, MDL can also be used to solve other learning tasks, for example in machine learning problems. Finally, the latter parts of the document describe the model complexity (Section 6), which is the penalty required for learning how our data behave.

1 Models

Models are mathematical abstractions for trying to explain data. We will want to choose a good model that fits our data well, and in Section 4 will see how to do so using MDL. MDL is related to Occam's razor, "entities should not be multiplied beyond necessity." In other words, we want the simplest model that explains the data reasonably well; any model more complicated than that is overkill, and may lead to overfitting.

To start understanding the details, we begin with 3 main concepts. First, *models* are mathematical abstractions, and they are intended to help us understand data. Second, we have *data*. While man-made data may fit a model perfectly, real-world data is often more complicated. Third, we have *algorithms* that fit models to data.

Simple vs. complex models. A simple model has low model complexity, and subsequently it is easy to work with, but the complexity of the data, which is a measure of how unpredictable the data appears to be in light of this model,[1] is quite large, and so the overall or total complexity of the model along with the data is large. At the other extreme, if the model is very complex, it reduces the complexity of the data but often not by much. In contrast, the model complexity is much greater in this latter case, and so the overall complexity is again large. A good trade-off between model complexity and data is a mid-sized model that captures the data rather well while still not being overly complex. These trade-offs appear in the books by Rissanen and in Mohri et al., and are optimized using the minimum description length (MDL) framework.

[1] Formally, the complexity of the data given the model is the negative log likelihood of the data, conditioned on the model. The motivation for the log likelihood measure appears in Section 3.

2 Model classes

Suppose that we are looking at data, x, and have a large class of possible models to choose from. This is often called a *model class*. The model class is denoted by C, and an individual model will be denoted by c. We ask, given x, *what is the "correct" model c within this model class, C?*

Again, with real-world data, there is no "correct" or perfect model for our data. If we have more data, then we can decide with greater confidence between models in the same model class C, meaning that we can perform more precise learning among models in C. We may need plenty of computation to sift through many models, and this could be computationally difficult with more data. Therefore, there are likely fundamental trade-offs between the data size, computation, and learning. These trade-offs are beyond the scope of our course. Later, we will see how the minimum description length (MDL) principle identifies the best model, c^*, among the models of C. But before discussing MDL, it helps to go into some depth about model classes, specifically parametric model classes.

Parametric model classes. A model class can be controlled by parameters. The parameter may have various values, and the number of possible values of each parameter could depend on the length N of our data, x. For each set of parameters, we will have a model, and the model class in its entirety will be comprised of the different models, each associated with different values for the parameters.

Bernoulli model. Bernoulli models are relatively simple models for data. Our data x is comprised of N bits, 0 and 1. Denote the nth bit by $x_n \in \{0, 1\}$, where $n \in \{1, \ldots, N\}$. The Bernoulli model considers x_n to be independent and identically distributed (i.i.d.); the independent aspect means that different bits are independent of one another, just like outcomes of tossing a (possibly biased) coin are statistically independent of one another. The identically distributed aspect means that the N bits have identical probabilities of having the values 0 and 1. We further specify a parameter θ, which is the probability that a bit is 1. Therefore, $\Pr(x_n = 1) = \theta$ and $\Pr(x_n = 0) = 1 - \theta$. The parameter θ is unknown, and our goal below will be to estimate it.

Again, a model class C is typically comprised of many models. For Bernoulli models, we could consider C that includes a set of different parameter values θ, one value per model. The model class C could be constant and comprised of a finite number of models (a finite number of θ values). Alternately, the number of models in C could depend on the length of the data, N. Typically, $|C|$ will grow as N grows, and we will see in Section 6 for a model class comprised of Bernoulli models that $|C|$ proportional to $\sqrt{N}$ works well. Finally, the model class C could also be infinitely large, for example all $\theta \in [0, 1]$.

Two parameters. A somewhat more involved parametric model involves 2 continuous valued parameters. If the previous bit x_{n-1} was 0, then the probability that the next bit is 1 satisfies $\Pr(x_n = 1 | x_{n-1} = 0) = \theta_0$. In contrast, if the previous bit was 1, then $\Pr(x_n = 1 | x_{n-1} = 1) = \theta_1$. Based on the values of θ_0 and θ_1, we could get different behaviors. As a first example, if $\theta_0 = 0.1$ and $\theta_1 = 0.9$, then after 0 we will likely have another 0, resulting in runs of zeros (typically of length $10 = 1/\theta_0$). And after 1, we will likely have another 1. As a second example, if $\theta_0 = 0.9$ and $\theta_1 = 0.1$, then this will be a

"switching source" that typically alternates between states, 0, 1, 0, 1, and so on.

As a third example, $\theta_0 = \theta_1$ is the single parameter case that we discussed earlier. What if θ_0 and θ_1 are close to one another? How can we decide between 2 hypotheses, where the first hypothesis involves a single parameter θ (the Bernoulli model above), and the second hypothesis involves 2 nearby parameters, θ_1 and θ_2? We leave this question for a possible final project, and point out that the closer that θ_1 and θ_2 are, the more data we will need in order to conclude with confidence that indeed the data is governed by two parameters, and not the simpler Bernoulli model.

As before, a model class C comprised of 2-parameter models could include different pairs of values for θ_0 and θ_1.

More parameters? More complicated parametric models are possible. Not only can a model class C rely on numerous parameters, it can also have different models within it that have different numbers of parameter. For example $c_1, c_2, \ldots, c_{20} \in C$ could rely on 1 parameter, $c_{21}, \ldots, c_{100} \in C$ on 2 parameters, and so on.

3 Bits and probabilities

A core tenet of information theory is that an input (often called a source) whose probability is p can be encoded using

$$\text{coding length} = -\log_2(p) \text{ bits}, \tag{1}$$

where $\log_2(\cdot)$ denotes the base-2 algorithm. To provide this result some intuition, imagine that we toss a fair coin, and each side has probability of 0.5; it takes $-\log_2(0.5) = -(-1) = 1$ bit to describe either side of the coin. If we toss a fair coin k times, there are 2^k outcomes, each has probability 2^{-k}, and it takes $-\log_2(2^{-k}) = k$ bits to describe any of the outcomes.

Entropy. Let us now revisit our Bernoulli example with parameter θ. How many bits do we need to encode the average input sample, x_n? We have 1 with probability θ, and it requires $-\log_2(\theta)$ bits. We have 0 with probability $1-\theta$, and it requires $-\log_2(1-\theta)$ bits. The sum of these is the *entropy*,

$$H(\theta) = -\theta \log_2(\theta) - (1-\theta)\log_2(1-\theta). \tag{2}$$

The entropy is the *fundamental best-possible average rate of bits needed to encode each input sample*; an entropy of 0.7 means that you can encode an N-bit input at close to $0.7N$ bits, and no better than that. Also, while our definition of entropy considers a Bernoulli model, similar expressions can be computed for more elaborate models.

Arithmetic coding. When $\theta = 0.5$, $-\log_2(\theta)$ and $-\log_2(1-\theta)$ are both integers, they are both 1, and it is intuitive to encode samples with 1 bit. What if θ differs from 0.5? In this case, at least one among $\log_2(\theta)$ and $\log_2(1-\theta)$ are non-integers. For example, if $\theta = 0.25$, then $-\log_2(\theta) = 2$ bits, but $-\log_2(1-\theta) = 0.415$ bits, which is non-integer. While it is probably clear how a 1 can be encoded using 2 bits, it is likely 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 data compression 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 a final 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.

4 Minimum description length principle

Again, how do we choose a good model? What is the "correct" model within this model class? The minimum description length (MDL) priniciple considers an input x associated with an unknown model c, which belongs to a model class C. Given knowledge about c, the data x has probability $P_c(x)$. Per (1), x can be encoded with length

$$\text{len}_c(x) = -\log_2(P_c(x))$$

bits. The different models within the model class C may also have different probabilities associated with them; in any case, $c \in C$ can be encoded using $\text{len}(c)$ bits.

The MDL principle is based on a simple observation. Given our data x, a reasonable model $c^* \in C$ is one that *minimizes the total description length* of a two part code, which in Part 1 encodes c^* with length $\text{len}(c^*)$, and in Part 2 uses c^* to encode x using $\text{len}_{c^*}(x)$ bits. (Details about two part codes appear in Section 5.) Therefore, the optimal MDL model for x satisfies

$$c^* = \arg\min_{c \in C} \{\text{len}(c) + \text{len}_c(x)\}. \tag{3}$$

This optimal model c^* will often be called the MDL model.

Below, we will see that the optimal size of some parametric model classes is on the order of $\sqrt{N}$ values per parameter. Therefore, per (1), the coding length required for the MDL model c^* is roughly $\frac{1}{2}\log_2(N)$ bits per parameter. Similarly, when κ parameters are used in the model class, the coding length required is roughly $\frac{\kappa}{2}\log_2(N)$. Using this rule of thumb wherein an unknown parameter "costs" us an additive $\approx \frac{1}{2}\log_2(N)$ bits or multiplicative factor of $\sqrt{N}$ in probability,[2] we will now see how the MDL principle can be used to perform *model selection*.

Example. To see how MDL could be used to also optimize the number of parameters, recall our polynomial curve fitting example discussed earlier in the course. Consider two hypotheses: (*i*) H_3 is the hypothesis that our data was generated by a polynomial of order 3 and (*ii*) H_4 is the hypothesis that our data was generated by a polynomial of order 4.

[2]From a probability perspective, x having a large probability, which corresponds to a shorter coding length, is favorable. The penalty for probability means that it is divided by $\sqrt{N}$, and it is more accurate to call this a multiplicative factor of $1/\sqrt{N}$.

How can we evaluate which hypothesis is more plausible? Intuitively speaking, it seems that the second hypothesis, H_4, involves a higher order polynomial, which will probably fit the data better. On the other hand, H_4 is a more complicated model in the sense that we have an extra parameter.

To evaluate more formally which hypothesis is more plausible, we apply the MDL principle. The better fit to the data means that a model with 4 parameters will yield a smaller error than that with 3 parameters, and a smaller error is mapped to a larger probability for the error or noise. That is, $\Pr(\text{error}|H_4) > \Pr(\text{error}|H_3)$,[3] and so

$$\text{len}(\text{error}|H_4) = \text{len}(x|H_4) < \text{len}(\text{error}|H_3) = \text{len}(x|H_3),$$

where we remind the reader that coding lengths relate to probabilities through the negative logarithm (1).

Now that we can compare $\text{len}(x|H_3)$ and $\text{len}(x|H_4)$, the MDL principle informs us that we should also account for the coding length required for the models. A model with 3 parameters requires coding length of roughly $\frac{3}{2}\log_2(N)$ bits; H_4 involves 4 parameters and requires length $\approx \frac{4}{2}\log_2(N)$. Ignoring the constants in the model cost, we can state that H_3 and H_4 involve coding lengths of $\text{len}(x|H_3) + \frac{3}{2}\log_2(N)$ and $\text{len}(x|H_4) + \frac{4}{2}\log_2(N)$ bits, respectively.[4] The shorter among the coding lengths allows to select between the hypotheses H_3 and H_4.

MDL is related to MAP and ML. In maximum a posteriori (MAP) methods, we are given various hypotheses, and compute the probability for these conditioned on our data. To provide a simple illustration, suppose that we have data x, and want to compare between 2 different parameter values, θ_1 and θ_2. If we have no reason to believe that either θ_1 or θ_2 is more likely, then we need only consider $\Pr(x|\theta_1)$ and $\Pr(x|\theta_2)$. However, suppose that we believe that the hypothesis underlying θ_1 is more likely. This belief can be expressed by allocating prior probabilities to our hypotheses, $\Pr(\theta_1)$ and $\Pr(\theta_2)$; these are called *priors*. Using Bayes' rule, we can now compare

$$\Pr(\theta_1)\Pr(x|\theta_1) = \Pr(\theta_1, x)$$

and

$$\Pr(\theta_2)\Pr(x|\theta_2) = \Pr(\theta_2, x).$$

That is, we compare *joint probabilities*, $\Pr(\theta_1, x)$ and $\Pr(\theta_2, x)$. Dividing by $\Pr(x)$,

$$\Pr(\theta_1|x) = \frac{\Pr(\theta_1, x)}{\Pr(x)} = \frac{\Pr(\theta_1)\Pr(x|\theta_1)}{\Pr(x)} \tag{4}$$

and

$$\Pr(\theta_2|x) = \frac{\Pr(\theta_2, x)}{\Pr(x)} = \frac{\Pr(\theta_2)\Pr(x|\theta_2)}{\Pr(x)}. \tag{5}$$

[3]For polynomial curve fitting, we are dealing with continuous valued random variables, hence it would be more precise to use probability density functions, $f(\cdot)$, and not probabilities, $\Pr(\cdot)$. We use $\Pr(\cdot)$ to keep the presentation simple.

[4]We can also add an extra bit to specify whether 3 or 4 parameters are used.

Expressions of the form $\Pr(\theta_1|x)$ and $\Pr(\theta_2|x)$ are the *posterior* probabilities for the possible parameter values, θ_1 and θ_2, conditioned on the data x. Selecting between hypotheses (or parameter values) this way is called *maximum a posteriori* (MAP) parameter estimation. You can see that MAP closely resembles MAP.

What about *maximum likelihood* (ML)? ML parameter estimation selects between θ_1 and θ_2 by comparing $\Pr(x|\theta_1)$ and $\Pr(x|\theta_2$. Looking at our equations for the posterior probabilities, (4) and (5), we see that ML is identical to MAP estimation, provided that $\Pr(\theta_1)$ and $\Pr(\theta_2)$ are identical. After all, $\Pr(x)$ is identical in both these equations, (4) and (5), and ML simply compares between $\Pr(x|\theta_1)$ and $\Pr(x|\theta_2$. Again, ML can be seen to be closely related to MDL in special cases where the description length for all possible models is the same.

Dualities beyond data compression. While we have discussed how to use the MDL principle for data compression, it can also be used to process data beyond data compression. In machine learning (ML), we are given features x and outcomes y, and want to learn the dependence of y on x. Expressing this problem in terms of a model class C, individual models $c \in C$ provide different probabilistic ways for y to depend on x. That is, the penalty or coding length for c is still $\text{len}(c)$.

Using the MDL principle, we want to minimize some total description length. Given an optimal model, c^*, we use it in two parts. First, we encode c^* with length $\text{len}(c^*)$. Next, we use c^* to encode y while aided by x; this requires a coding length $\text{len}_c(y|x) = -\log_2(\Pr(y|x, c))$. Therefore, similar to (3), the optimal MDL model for features x and outcomes y satisfies

$$c^* = \arg\min_{c \in C}\{\text{len}(c) + \text{len}_c(y|x)\}.$$

While the rest of our discussion considers data compression, this explanation of how MDL can be used for machine learning demonstrates that MDL can be used for various learning problems.

Occam's razor. As we mentioned at the beginning, MDL is related to Occam's razor, "entities should not be multiplied beyond necessity." Among competing hypotheses, the one with the fewest assumptions should be selected. Different hypotheses may have different probabilities, and the data x may have different probabilities conditioned on different hypotheses. However, all in all, among the models that explain the data reasonably well, we want the simplest model.

We conclude our discussion of the MDL principle by mentioning that Section 8 discusses mixtures as a more precise way than MDL to evaluate probabilities when discussing model classes. While being more precise, mixtures are also more complicated in many cases. We also discuss an interpretation of different models as slices of a probabilistic pie in Section 8.

5 Two-part codes

We will see in Section 6 that each unknown continuous valued parameter increases the coding length by roughly $\frac{1}{2}\log_2(N)$ bits. With κ parameters, the extra coding length required is roughly $\frac{\kappa}{2}\log_2(N)$. To work toward this $\frac{1}{2}\log_2(N)$ result, we consider *two part codes*, which are comprised of two parts.

Part 1 estimates the parameter(s) governing the data, and encodes them. We will construct a two part code for an example Bernoulli distribution, where $\Pr(x_n = 1) = \theta$ and $\Pr(x_n = 0) = 1 - \theta$, $\forall n \in \{1, \ldots, N\}$, and the value of the parameter θ is unknown. To estimate θ, we first count the number of zeros and ones in the data,

$$N_0 = \sum_{n=1}^{N} 1_{\{x_n=0\}} \quad \text{and} \quad N_1 = \sum_{n=1}^{N} 1_{\{x_n=1\}}, \tag{6}$$

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

$$\widehat{\theta} = \frac{N_1}{N_0 + N_1} = \frac{N_1}{N}. \tag{7}$$

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 $\widehat{\theta}$; the encoder must somehow convey $\widehat{\theta}$ (or some information about it) to the decoder. Note that there are $N + 1$ possible values of $\widehat{\theta}$, because $N_1 \in \{0, 1, \ldots, N\}$ in the numerator (7), while the N in the denominator is fixed. In principle, we could encode $\widehat{\theta}$ precisely using $\log_2(N + 1)$ bits. However, we will demonstrate in Section 6 that it is better to quantize $\widehat{\theta}$ to one of roughly $\sqrt{N}$ representation levels, meaning that its quantized version is encoded with roughly $\frac{1}{2}\log_2(N)$ bits.

Consider K representation levels. The encoder will *quantize* or discretize $\widehat{\theta}$ into one of K bins. To keep things simple, we quantize $\widehat{\theta}$ *uniformly* in the range $[0, 1]$. 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 the encoder identifies the appropriate bin index, $k \in \{1, \ldots, K\}$, which contains $\widehat{\theta}$, the encoder conveys k to the decoder using $\log_2(K)$ bits.

Within each bin k, we have a *representation level* (also known as the quantization level) in the middle of the bin. Representation level r_k in bin k takes the value

$$r_k = \frac{k - \frac{1}{2}}{K}, \tag{8}$$

which is the mid point between the two edges of the bin, $\frac{k-1}{K}$ and $\frac{k}{K}$.

We have used a uniform quantizer, and representation levels in the middle of bins are simple to use. It can be shown that non-uniform quantization of $\widehat{\theta}$ can reduce the coding length in some cases. 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. (Non-uniform quantization may be a good direction for a final project.)

Part 2 compresses the string x with probablity model r_k. Following Part 1, the encoder and decoder have agreed to use some representation level r_k to compress the actual data x. Consider how a compression algorithm processes x_n at the encoder, where $n \in$

$\{1, \ldots, N\}$. If $x_n = 1$, the algorithm assigns probability r_k to x_n; else $x_n = 0$, and the algorithm assigns probability $1 - r_k$. Therefore, when $x_n = 1$, the coding length will be $-\log_2(r_k)$ bits, else $x_n = 0$, and it will be $-\log_2(1 - r_k)$ bits. The total coding length for compressing x in Part 2 of the two part code using the quantized parameter r_k will be

$$\begin{aligned} \text{len}_{r_k}(x) &= -\sum_{n=1}^{N} \left[1_{\{x_n=0\}} \cdot \log_2(1 - r_k) + 1_{\{x_n=1\}} \cdot \log_2(r_k)\right] \\ &= -N_0 \log_2(1 - r_k) - N_1 \log_2(r_k) \\ &= -N(1 - \widehat{\theta}) \log_2(1 - r_k) - N\widehat{\theta} \log_2(r_k), \end{aligned} \tag{9}$$

where we highlight that $\widehat{\theta}$ is the estimated parameter (7), and r_k is its quantized value.

The compression algorithm will use arithmetic coding (Section 3) to convert the probabilities for $x_1, x_2, \ldots, x_N$ into bits. Once the arithmetic encoder has processed all of x, it outputs $\text{len}_{r_k}(x) + O(1)$ bits, where the $O(1)$ term is often 2–3 bits. Therefore, practically speaking, the coding length $\text{len}_{r_k}(x)$ of Part 2 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),

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

which corresponds to (9).

Total coding length. Having developed our two part code and discussed it in detail, what is the total coding length of both parts? Recall that Part 1 encodes $\log_2(K)$ bits, and Part 2 encodes $\text{len}_{r_k}(x) = -N(1-\widehat{\theta})\log_2(1-r_k) - N\widehat{\theta}\log_2(r_k)$ bits (9). Therefore, the total coding length is

$$\log_2(K) - N(1 - \widehat{\theta}) \log_2(1 - r_k) - N\widehat{\theta} \log_2(r_k). \tag{10}$$

Now that we have a detailed understanding of two part codes, we can move toward model complexity.

6 Model complexity

The *model complexity* is the penalty for learning how our data behaves. If we evaluate things from a data compression perspective, then we are interested in saving on coding length, and quantify the model complexity in terms of excess bits. If we evaluate probabilities, then the model complexity is quantified by how much the probability of our data was reduced, owing

to our lack of initial knowledge about the data's properties. Irrespective of our perspective, coding length or probabilities, we have a model class C for possible relations in our data, and want to learn which $c \in C$ seems to fit our data best. To keep the discussion simple, we mostly take the coding length perspective below. We refer to the excess coding length due to needing to learn how our data behaves as *redundancy*.

Having discussed MDL and two part codes, given the model class C, we know how to identify c^*, the optimal MDL model. However, the structure of C may impact the model complexity, which in our case is redundancy (excess coding length). If C is designed poorly, then the redundancy will likely be large. Instead of specific results for specific model classes, we want general guidelines for model complexity. Specifically, we will see how the redundancy will grow with N and κ, the number of parameters.

To gain this more general type of understanding of model complexity, we will design C in a way that the redundancy will be small. In Section 5, we considered a two part code for a Bernoulli model. We applied a uniform quantizer with K quantization bins to $\widehat{\theta}$, leading to a total coding length given by (10). In this expression, the $\log_2(K)$ comes from Part 1. The expression for the coding length of Part 2,

$$\text{len}_2 = -N(1-\widehat{\theta})\log_2(1-r_k) - N\widehat{\theta}\log_2(r_k), \tag{11}$$

is less clear. How large will it be as a function of K? Intuitively, it seems that if K is too small, then we will waste coding length in Part 2; and if K is too large, then Part 1 will likely be wasteful. Later, we will quantify this trade-off carefully.

Ideal coding length for Part 2. Again, the coding length in Part 1 is well-defined, but what about Part 2? How does it compare to other possible coding lengths? Suppose hypothetically that we somehow knew the true parameter θ. In that case, we could compress x at $H(\theta)$ (2), which is the fundamental best-possible average per-sample rate of bits needed to encode x. (In other words, x will require $NH(\theta)$ bits to compress, on average, over the ensemble of data sets generated by a parametric model governed by θ.) We do not know θ, of course. But the encoder knows $\widehat{\theta}$ (7) and its quantized version, r_k; the decoder only knows r_k.

What is the *ideal* coding length? Suppose that we could compress at the encoder with any Bernoulli parameter, θ_i. In particular, suppose that we could use the *ideal* parameter that minimizes the coding length in Part 2. What is θ_i? Using (9) while substituting θ_i instead of r_k, the ideal length is

$$\text{len}_2 = -N(1-\widehat{\theta})\log_2(1-\theta_i) - N\widehat{\theta}\log_2(\theta_i). \tag{12}$$

Taking the derivative with respect to (w.r.t.) θ_i, and setting it to be zero (after all, θ_i is ideal),

$$\frac{\partial \text{len}_2}{\partial \theta_i} = -N(1-\widehat{\theta})\frac{-1}{\ln(2)(1-\theta_i)} - N\widehat{\theta}\frac{+1}{\ln(2)\theta_i} = 0, \tag{13}$$

where the $\ln(2)$ terms in the denominators account for the derivatives of $\log_2(x) = \ln(x)/\ln(2)$ being $\frac{1}{\ln(2)x}$, and the $+1$ and -1 in the numerators account for the derivatives of θ_i and $1-\theta_i$,

respectively. Because the derivative of len_2 (12) w.r.t. θ_i, (13), is zero,

$$(1-\widehat{\theta})\frac{1}{(1-\theta_i)} = \widehat{\theta}\frac{1}{\theta_i},$$

and we conclude that

$$\theta_i = \widehat{\theta} = \frac{N_1}{N}.$$

Encoding with an ideal $\theta_i = \widehat{\theta}$ results in coding length $NH(\widehat{\theta})$. This ideal coding length serves as a best-possible benchmark.

Example. 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 $NH(\widehat{\theta}) = \text{len}_{\widehat{\theta}}(x)$. However, our arithmetic coder compresses with the representation level r_k, and

$$\text{len}_{r_k}(x) = -800\log_2(1-0.21) - 200\log_2(0.21) = 722.4$$

bits. It can be seen that the quantized representation level, r_k, increases the coding length beyond the benchmark entropy by roughly half a bit.

Optimal quantizer size K. What is the optimal K? We will make a simplifying assumption that

$$K = N^{\alpha}, \tag{14}$$

and optimize α. We will see that this polynomial dependence of K on N arises from our analysis.

Using (14), the coding length of Part 1 is $\log_2(K) = \alpha\log_2(N)$. What about Part 2? Each quantization bin will be of width $1/K$, hence the quantization error will be upper bounded by $\frac{1}{2K}$. Specifically, we denote

$$r_k = \widehat{\theta} + \delta, \tag{15}$$

where δ is a correction term, and we have discussed that

$$|\delta| < \frac{1}{2K} = \frac{N^{-\alpha}}{2}. \tag{16}$$

Let us revisit len_2 (10), our expression for the length in Part 2 (11),

$$\begin{aligned}
\text{len}_2 &= -N(1-\widehat{\theta})\log_2(1-r_k) - N\widehat{\theta}\log_2(r_k) && (17)\\
&= -N(1-\widehat{\theta})\log_2(1-\widehat{\theta}-\delta) - N\widehat{\theta}\log_2(\widehat{\theta}+\delta) && (18)\\
&= -N(1-\widehat{\theta})\left[\log_2(1-\widehat{\theta}) + \log_2(1-\frac{\delta}{1-\widehat{\theta}})\right] - N\widehat{\theta}\left[\log_2(\widehat{\theta}) + \log_2(1+\frac{\delta}{\widehat{\theta}})\right] && (19)\\
&= -N\left[(1-\widehat{\theta})\log_2(1-\widehat{\theta}) + \widehat{\theta}\log_2(\widehat{\theta})\right]\\
&\quad -N\left[(1-\widehat{\theta})\log_2(1-\frac{\delta}{1-\widehat{\theta}}) + \widehat{\theta}\log_2(1+\frac{\delta}{\widehat{\theta}})\right], && (20)
\end{aligned}$$

where (17) is (11), (18) uses our correction term δ (15), (19) uses properties of logarithms, and (20) rearranges terms. Looking at (20), its first term is $NH(\widehat{\theta})$, which is our ideal coding length.

Let us focus on the second term, which is the redundancy of Part 2,

$$\text{redundancy}_2 = -N\left[(1-\widehat{\theta})\log_2(1-\frac{\delta}{1-\widehat{\theta}}) + \widehat{\theta}\log_2(1+\frac{\delta}{\widehat{\theta}})\right].$$

We can analyze redundancy_2 using a Taylor approximation,

$$\log_2(1+x) = c_1 x + c_2 x^2 + \text{higher order},$$

where c_1 and c_2 are constants, The first order Taylor approximation for redundancy_2 is zero, because θ_i provides the ideal (lowest) coding length among all possible Bernoulli parameters. We perform a second order approximation,

$$\begin{aligned}
\text{redundancy}_2 &\approx -N\left[(1-\widehat{\theta})c_2\left(-\frac{\delta}{1-\widehat{\theta}}\right)^2 + \widehat{\theta}c_2\left(\frac{\delta}{\widehat{\theta}}\right)^2\right] \\
&= -N\delta^2 c_2\left[\frac{1}{1-\widehat{\theta}} + \frac{1}{\widehat{\theta}}\right] \\
&= -N\delta^2 c_2 \frac{1}{\widehat{\theta}(1-\widehat{\theta})}, \qquad (21)
\end{aligned}$$

where the approximation is due to higher order terms that we are neglecting. In (21), because $|\delta| \leq \frac{N^{-\alpha}}{2}$ (16),

$$\text{redundancy}_2 = -\frac{c_2}{\widehat{\theta}(1-\widehat{\theta})}N^{1-2\alpha}, \qquad (22)$$

where we note that $c_2 < 0$, hence the redundancy of Part 2 is non-negative.

Our redundancy for Part 2 (22) is proportional to $N^{1-2\alpha}$, while the redundancy for Part 1 is $\log_2(K) = \alpha\log_2(N)$ (10). If $\alpha > \frac{1}{2}$, then the redundancy in Part 1 exceeds $\frac{1}{2}\log_2(N)$. But if $\alpha < \frac{1}{2}$, then the redundancy in Part 2 is polynomial in N, which dominates the logarithmic redundancy in Part 1. We conclude that the optimal α is $\frac{1}{2}$, hence (14) indicates that the optimal number of quantization bins, K, is proportional to $N^{0.5} = \sqrt{N}$.

Multiple parameters. Our discussion for the optimal K considered the two part code of Section 5, which is for Bernoulli models. When a more sophisticated model class with κ parameters is considered, an analogous derivation shows that the penalty per parameter should scale as $\frac{1}{2}\log_2(N)$ bits of redundancy per parameter.

7 Decoding perspective

Having analyzed the optimal model class size, we see that we want approximately $\sqrt{N}$ quantization bins per parameter. We now describe these results from a different perspective.

To keeps things simple, suppose that we have nearby parameters θ_1 and θ_2, and we generated x using one of them. Given x, can we decide reliably whether θ_1 or θ_2 generated our data x? This problem has a decoding flavor. From a communication engineering perspective, we can imagine that a genie is selecting between multiple possible hypotheses (in our case, 2 hypotheses), and is communicating the selection to us by using the distribution of x as a channel. Indeed, Merhav and Feder showed that one can think of the redundancy of data compression as being analogous to the capacity of a channel from the choice of parameter values to the data x being generated; see "A strong version of the redundancy-capacity theorem of universal coding," IEEE Trans. Information Theory, 1996. Returning to our problem, we will see that N must be sufficiently large in order to decide reliably between the 2 values for our parameters, which correspond to 2 hypotheses.

We will approach the decoding problem by computing the empirical probability, $\widehat{\theta}$ (7). What is the distribution of $\widehat{\theta}$? Suppose that our data x was generated by a Bernoulli model with true parameter θ. We have $\Pr(x_n = 1) = \theta$ and $\Pr(x_n = 0) = 1 - \theta$ We begin by analyzing the distribution of N_1.

Distribution of N_1. Using N samples in x, the expected value of N_1 is

$$E[N_1] = E\left[\sum_{n=1}^{N} 1_{\{x_n=1\}}\right] = \sum_{n=1}^{N} E\left[1_{\{x_n=1\}}\right] = NE\left[1_{\{x_n=1\}}\right],$$

where we relied on linearity of the expectation operator. Because $E\left[1_{\{x_n=1\}}\right] = \theta \cdot 1 + (1 - \theta) \cdot 0 = \theta$, we have that $E[N_1] = N\theta$.

Next, we want to compute the variance of N_1. In probability theory, the variance of the sum of N i.i.d. random variables (RVs) is N times the variance of each individual RV. That is,

$$\mathrm{Var}(N_1) = N\mathrm{Var}(x_n).$$

Let us compute the variance of x_n,

$$\begin{aligned}
\mathrm{Var}(x_n) &= E[(x_n - E[x_n])^2] \\
&= E[(x_n - \theta)^2] \qquad (23) \\
&= \Pr(x_n = 1)(1 - \theta)^2 + \Pr(x_n = 0)(0 - \theta)^2 \\
&= \theta(1 - \theta)[(1 - \theta) + (\theta)] \\
&= \theta(1 - \theta). \qquad (24)
\end{aligned}$$

where (23) arises from the expected number of ones x_n being θ. The variance of N_1 obeys

$$\mathrm{Var}(N_1) = N\theta(1 - \theta).$$

Because we have N i.i.d. RVs, the variance of each individual RV is some constant, and the variance of the sum is proportional to N. Because the variance, which is the expected square of the difference between N_1 and its expected value, is linear in N, the standard deviation is proportional to $\sqrt{N}$. More precisely,

$$\mathrm{Std}(N_1) = \sqrt{\mathrm{Var}(N_1)} = \sqrt{N\theta(1 - \theta)}.$$

Distribution of $\widehat{\theta}$. Recall that we are interested in computing $\widehat{\theta}$, and not N_1. Of course, $\widehat{\theta} = \frac{N_1}{N}$ (7), hence its expected value is

$$E[\widehat{\theta}] = E[N_1/N] = E[N_1]/N = N\theta/N = \theta,$$

and its standard deviation

$$\text{Std}(\widehat{\theta}) = \text{Std}(N_1/N) = \text{Std}(N_1)/N = \sqrt{\frac{\theta(1-\theta)}{N}},$$

because the standard deviation is a linear operator (multiplying an RV by some constant c_3 multiplies the standard deviation by c_3). In summary, the standard deviation of $\widehat{\theta}$ is proportional to $\frac{1}{\sqrt{N}}$.

Spacing of uniform quantizer. What is the impact of the standard deviation of $\widehat{\theta}$ on the size of the model class C? To keep things simple, consider C that corresponds to a uniform quantizer in Part 1, per Section 5. Our quantizer is comprised of representation levels that are uniformly spaced apart, $\{\frac{0.5}{K}, \frac{1.5}{K}, \ldots, \frac{K-0.5}{K}\}$ as implied by (8). Each of these representation levels corresponds to a Bernoulli model.

If the bin width, $\frac{1}{K}$, is much smaller than the standard deviation of $\widehat{\theta}$, then it will be impossible to distinguish which among several adjacent models in C the data actually came from. The complexity of such a model class is larger than would allow to differentiate between adjacent Bernoulli models. At the other extreme, if the bin width $\frac{1}{K}$ is much larger than the standard deviation, then for an arbitrary θ we may have $\widehat{\theta}$ that is not sufficiently close to any representation level. In conclusion, from a decoding perspective, we want the bin width to resemble the standard deviation of $\widehat{\theta}$.

8 Discussion

We have seen that the penalty for learning an unknown parameter is roughly $0.5\log_2(N)$ bits, where N is the number of samples of data. Alternately, the penalty for learning an unknown parameter can be interpreted as a multiplicative decrease in the probability of our data x by a factor of approximately $\sqrt{N}$ (1). Recall that if we have an optimal model c within a model class C, then the length-N data x has probability $P_c(x)$, and can be represented (encoded) using $-\log_2(P_c(x))$ bits. If C is a parametric model class and corresponds to one unknown parameter, then the $\approx 0.5\log_2(N)$ bits correspond to $\approx \sqrt{N}$ possible models in C.

To keep things simple, suppose that we have exactly $\sqrt{N}$ models.[5] Assigning each of the models a coding length of precisely $0.5\log_2(N)$ bits within an MDL framework is analogous to assigning each model class a probability $1/\sqrt{N}$. Therefore, one can now *envision a pie or cake of probabilities*. Earlier, x had probability $P_c(x)$ within the pie slice corresponding to the model c. We now cut the pie for all of C into $\sqrt{N}$ slices, and the slice for c is subsequently partitioned into sub-slices, where the sub-slice for x has probability $\frac{1}{\sqrt{N}}P_c(x)$. Each of the

[5]We conveniently ignore issues such as $\sqrt{N}$ needing to be integer.

$\sqrt{N}$ slices of the pie contains sub-slices for all possible inputs, and in particular our actual input x. That is, there are $\sqrt{N}$ sub-slices that correspond to x. The MDL approach allocates x the largest probability among these $|C| = \sqrt{N}$ sub-slices, i.e., $\max_{c \in C} \frac{1}{\sqrt{N}} P_c(x)$. In contrast, mixtures allocate the sum of probabilistic areas of all $\sqrt{N}$ sub-slices that correspond to x, i.e., $\sum_{c \in C} \frac{1}{\sqrt{N}} P_c(x)$.[6]

Irrespective of specific probabilistic approach used – MDL or mixtures – we have seen a reasonable way to assign probabilities to any length-N input x, despite lack of knowledge of the model that generated x. Our only assumption here was that x is generated by some parametric class. Computations of probabilities tie into the course in multiple ways. First, having reviewed probability and random processes materials, we now see how this knowledge can be deployed in designing useful probability spaces. More interestingly, model complexity concepts also tie into the machine learning materials that we will discuss later in the course. These complexity concepts show how quickly we can learn, and in some cases the underlying probabilistic concepts map to machine learning algorithms.

9 References

I am writing this informally for now.

The textbooks Rissanen and in Mohri et al. describe how to use MDL for various learning tasks.

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.

Papers that discuss finer details of two part codes include B. S. Clarke and A. R. Barron, "Jeffreys' prior is asymptotically least favorable under entropy risk," J. Statistical Planning Inference, 1994 and J. Rissanen, "Fisher information and stochastic complexity," IEEE Trans. Information Theory, 1996.

Merhav and Feder showed that one can think of the redundancy of data compression as being analogous to the capacity of a channel from the choice of parameter values to the data x being generated, see "A strong version of the redundancy-capacity theorem of universal coding," IEEE Trans. Information Theory, 1996.

[6]More formally, MDL does not yield a true probability space, because the probabilities sum to less than one. Mixtures do offer a valid probability space, but are used less often due to being less algorithmically tractable in many cases.