10-704: Information Processing and Learning **Spring 2012**

Lecture 9: Arithmetic Coding and Universal Codes

Lecturer: Aarti Singh *Scribes: Andrew Rodriguez*

9.1 Arithmetic Codes

One of the main drawbacks of Shannon codes and Huffman codes is that we need to compute the probability of sequences of symbols before we can code (Recall we want to encode sequences to get close to entropy). Once we do so, the codes are only good as long as the underlying probability distribution does not change. To address this issue, we introduce Arithmetic codes.

Arithmetic codes are family of streaming codes that code symbols sequentially and can adapt to changing distributions. It's usually assumed that both the encoder and decoder have the same predictor $p(x_n|x_1, \ldots, x_{n-1})$. If we have a good predictor, we should be able to encode well.

9.2 Encoding

Arithmetic codes map streams of symbols to a real number between 0 and 1. The first observation is that binary strings define points in $[0, 1)$. A string $x_1, \ldots, x_n$ maps to the binary number $0.x_1x_2 \ldots x_n$ which is equal to $\sum_i x_i 2^{-i}$. Let's define our alphabet to be $\{a_1 \ldots a_{|\mathcal{X}|}\}$.

We start by dividing the interval $[0, 1)$ into $|\mathcal{X}|$ intervals, each with length $p_i = p(x_1 = a_i)$ for $i = 1, \ldots, \mathcal{X}$. Then, after seeing that $x_1 = a_i$, we subdivide interval $p(x_1 = a_i)$ further into $|\mathcal{X}|$ sub-intervals so that the lengths of the sub-intervals are proportional to the probability of the second symbol ($p(x_2 = a_j \,|\, x_1 = a_i)$ for $j = 1, \ldots, \mathcal{X}$). Note that the length of the intervals at this second stage is $p(x_2 = a_j \,|\, x_1 = a_i)p(x_1 = a_i) = p(x_2 = a_j, x_1 = a_i)$.

After seeing that the second symbol $x_2 = a_j$, we subdivide the interval $p(x_2 = a_j \,|\, x_1 = a_i)$ into $|\mathcal{X}|$ sub-intervals so that the lengths of the sub-intervals are proportional to $p(x_3 = a_k \,|\, x_2 = a_j, x_1 = a_i)$ for $k = 1, \ldots, \mathcal{X}$. Note that if x^n is generated by a Markov chain, then $p(x_3 = a_k \,|\, x_2 = a_j, x_1 = a_i) = p(x_3 = a_k \,|\, x_2 = a_j)$. The length of the intervals at this third stage is $p(x_3 = a_k \,|\, x_2 = a_j, x_1 = a_i)p(x_2 = a_j, x_1 = a_i) = p(x_3 = a_k, x_2 = a_j, x_i = a_i)$.

We continue this process until we've consumed the entire input. At this point, we say that the interval for x^n is $[L(x^n), R(x^n))$ where L and R are the left and right endpoints respectively. Because all the intervals are contained in $[0, 1)$ and $\sum_{x^n} p(x^n) = 1$, $R(x^n) - L(x^n) = p(x^n)$. We can also view the interval as a cumulative distribution $F(x^n) = \sum p(y^n)$ where y_n are all the sequences that have a binary representation smaller than x_n ($\sum y_i 2^{-i} \leq \sum x_i 2^{-i}$). Using this definition, we can rewrite the interval as $[L(x^n), R(x^n)) = [F(x^n) - p(x^n), F(x^n))$.

Now that we have the interval, to encode we must spit out some z inside the interval. $z \in [L(x^n), R(x^n)); z = 0.z_1z_2z_3 \ldots$. We can pick any z in the interval we want, but our choice can affect propreties of the code.

9.2.1 Examples

See Figure 9.1. The numbers on the diagram are in binary. We'll be dealing with a four symbol alphabet: $\{a, b, c, d\}$. In both examples, we'll encode the string "aab".

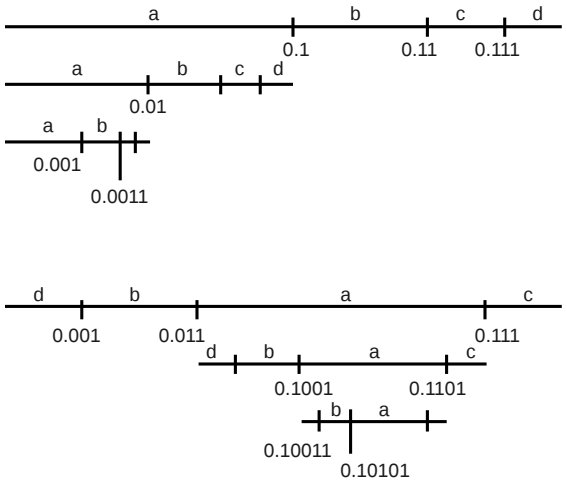

Figure 9.1: Two examples for arithmetic coding of sequence "aab" for the same iid source.

9.2.1.1 Natural example

Suppose we have an iid source that is described by symbol probabilities $p(a) = \frac{1}{2}$, $p(b) = \frac{1}{4}$, $p(c) = \frac{1}{8}$, and $p(d) = \frac{1}{8}$. When we process the first "a", we see that the interval of the final string must be inside the interval $[0, 0.1)$. At this point, we could emit a 0 as the first bit as all strings that start with "a" must lie in the first interval and have first bit 0. In general we can't always emit early (as can be seen in the next example). The next "a" is in $[0, 0.01)$. Again, the encoder can send out a 0 as all strings starting in "aa" must lie in the corresponding interval and have second bit 0. Finally, the "b" is in $[0.001, 0.0011)$. We could return any number in this interval to encode "aab", e.g. the total codeword for "aab" could be 001 or 0010 or 001011 etc. Also, notice that in this case the decoder can decode bits as they are sent - first 0 maps to an 'a" and so on. In the next example, we see that this is not always possible.

9.2.1.2 Out of order example

Now let's keep the probabilities the same (the same iid source), but put them in a different order. We'll divide the interval into d, b, a, c instead of a, b, c, d. The final intervals are different, as illustrated in the figure.

Notice that after observing the first "a", the encoder cannot yet transmit a bit as any string that begins with "a" lies in the corresponding interval $[0.011, 0.111)$ and can have its first bit as 0 or 1. This s because in this second example the bits in the final number don't necessarily correspond to the different subintervals. This is in constrast to Huffman codes where each bit in the encoded symbols represent a branch in the tree.

However, arithmetic codes can accomdate for context-sensitive distributions e.g. instead of an iid source, any other conditional distribution of second symbol given the first can be used in the second stage.

While spitting out bits as soon as the current intervals lower and upper ends agree in some bits is efficient, the resulting codeword may not correspond to a prefix code, so the decoder may not know when to stop (when the block length n ends). There is a more principled way to pick z such that the resulting code is prefix free as described next.

9.2.2 Shannon-Fano-Elias Encoding: picking z

We can pick z so that the resulting code is prefix free. Once we find the interval $[L(x^n), R(x^n))$, we can simply take the decimal part of the midpoint: $\frac{L(x^n)+R(x^n)}{2} = \bar{F}(x^n)$.

The midpoint could have a very long expansion, so we are going to round it off after $\tilde{m}$ bits.

We know that the midpoint is: $\frac{L(x^n)+R(x^n)}{2} = \bar{F}(x^n) = 0.z_1 z_2 \dots z_{\tilde{m}} \dots$. We define our rounding as $\tilde{F}(x^n) = 0.z_1 z_2 \dots z_{\tilde{m}}$ where $\tilde{m} = \lceil -\log p(x^n) \rceil + 1$. By the way we set $\tilde{m}$, we have $\bar{F}(x^n) - \tilde{F}(x^n) < 2^{-\tilde{m}} \leq \frac{p(x^n)}{2} = \bar{F}(x^n) - L(x^n)$. This implies $L(x^n) < \tilde{F}(x^n) \leq \bar{F}(x_n) < R(x^n)$, which means that $\tilde{F}$ is in the interval.

Theorem 9.1 *Shannon-Fano-Elias encoding is prefix-free.*

Proof: Our code is a prefix code if and only if no other sequence of length n can have $\tilde{F}(x_n)$ as the prefix of its binary encoding.

We can write any number with prefix $\tilde{F}(x^n) = \sum_i^{\tilde{m}} z_i 2^{-i}$, say w, as $w = \sum_i^{\tilde{m}} z_i 2^{-i} + \sum_{i=\tilde{m}+1}^{\infty} z_i 2^{-i}$. The series $\sum_{i=\tilde{m}+1}^{\infty} z_i 2^{-i} < \sum_{i=\tilde{m}+1}^{\infty} 2^{-i} = 2^{-\tilde{m}}$ (since all z_is can't be 1, otherwise the binary representation is same as flipping the $\tilde{m}$ bit and in that case w won't have as $\tilde{F}(x^n)$ prefix). Hence, we have

$$\begin{aligned} w &< \tilde{F}(x^n) + 2^{-\tilde{m}} \\ &\leq \bar{F}(x^n) + 2^{-\tilde{m}} \\ &\leq \bar{F}(x^n) + \frac{p(x^n)}{2} \\ &\leq R(x^n) \end{aligned}$$

It is obvious that $w \geq \tilde{F}(x^n) > L(x^n)$. Hence, $w \in [L(x^n), R(x^n))$. Therefore, all such ws can only represent the sequence x^n. ■

9.2.3 Encoding Length

Let's find the expected length using our method of picking z. $\mathbb{E}[L(x^n)] = \sum p(x^n) l(x^n)$. The length of x^n will be $\tilde{m}$, which depends on x^n. Using definition of $\tilde{m}$, $\mathbb{E}[L(x^n)] \leq \sum p(x^n) \log \frac{1}{p(x^n)} + 2 \leq H(x^n) + 2$.

Thus, the expected length per symbol $\mathbb{E}[L(x^n)/n] \leq H(x^n)/n + 2/n$, approaches the entropy rate.

9.2.4 Comparison with Shannon and Huffman codes

Notice that while all huffman, shannon and arithmetic codes can achieve per symbol expected length close to entropy, arithmetic codes are much more efficient as they can accomdate varying source distributions

while only requiring computation of $n|\mathcal{X}|$ conditional probabilities (probaility of each of the n observed symbols given past). On the other hand, for a non-iid source, huffman and shannon codes require computing probabilities of all length n sequences, i.e. $|\mathcal{X}|^n$ probability values. And the computations need to be repeated all over if the source distribution changes.

9.3 Connection between prediction accuracy to encoding

The **pointwise redudancy of a code** is defined as the length of the code minus the ideal code length (the Shannon information content) per symbol where $x^n \sim P$: $R_{P,C}(x^n) = \frac{1}{n}[L_C(x^n) - (-\log P(x^n))]$.

The **expected redundancy** is the expected pointwise redundancy: $\mathbb{E}_P[R_{P,C}(x^n)] = \frac{1}{n}[\mathbb{E}_P[L_C(x^n)] - H(x^n)]$. This is greater than zero if C is uniquely decodable, but could be negative otherwise.

Next, we define the concept of universal codes and show that arithmetic codes are universal codes.

9.3.1 Universal Codes

A code is *weakly universal* if for some given class $\mathcal{P}$ of processes $\mathbb{E}_P[R_{P,C}(x^n)] \xrightarrow[n\to\infty]{} 0$ for each $P \in \mathcal{P}$.

It is *strongly universal* if $\mathbb{E}_P[R_{P,C}(x^n)]$ goes to zero faster than some rate $r(n)$ for all distributions $P \in \mathcal{P}$.

This is similar to the goal of machine learning where we are trying to model a probability distribution in some class and we want to see how our model does compared to the best possible risk for any distribution. Ideally, we want this difference to go to zero.

We'll also defined the **worst case expected redundancy** of a code C with respect to a class of probability distributions $\mathcal{P}$ as

$$\bar{R}_C = \sup_{P\in\mathcal{P}} \mathbb{E}_P[R_{P,C}].$$

The **worst case max redudancy** (R^*_C) is defined as

$$R^*_C = \sup_{P\in\mathcal{P}} \max_{x^n} R_{P,C}(x^n)$$

where, instead of expectation, we have the worst case of max redundancy over all strings.

9.3.2 Prefix codes and corresponding distribtuions

For every prefix code C, there exists a corresponding probability distribution Q such that $L_C(x^n) \geq -\log Q(x^n)$. Because C is a prefix code, it satisfies the Kraft inequality so we can construct Q where $Q(x^n) = \frac{2^{-L(x^n)}}{\sum 2^{-L(x)}}$.

Conversely, for any probability distribution Q, there exists a prefix code C with $L_C(x^n) < -\log Q(x^n) + 1$ because we can just define C to be the Shannon code corresponding to Q. This gives us a correspondence between the probability distribution Q and the length of the codes for prefix codes.

9.3.3 Minimax Coding Redundancy and Minimax Excess Risk

Now we can talk about the minimax coding redundancy which is related to minimax excess risk in estimating distributions from the class of distributions $\mathcal{P}$.

The redundancy of any prefix code is bounded from below by the minimax risk in estimating a distribution from class $\mathcal{P}$ is

$$\bar{R}_C \geq \min_Q \max_{P \in \mathcal{P}} \frac{1}{n} \sum_{x^n} P(x^n) \log \frac{P(x^n)}{Q(x^n)} = \bar{R}$$

$\sum_{x^n} P(x^n) \log \frac{P(x^n)}{Q(x^n)}$ is the KL divergence $D_n(P||Q)$ between P and Q evaluated on strings of length n, which is the minimax excess risk corresponding to negative log loss (refer Lecture 1).

If we want to find the best Q, how well you can estimate any distibution from this class according to the negative log loss will decide what your expected coding redundancy should be for the worst case. In fact, we can talk about which Q achieves the minimax bound and we can take that distribution and design its corresponding Shannon code and that wil get you smallest possible worst case redundancy up to 1 bit.

Thus, there is a strong connection between how well you're estimating from a class of distributions $\mathcal{P}$ and how well you're doing your encoding.

We can do the same thing for the worst case max redundancy:

$$R^*_C \geq \min_Q \max_{P \in \mathcal{P}} \max_{x^n} \frac{1}{n} \log \frac{P(x^n)}{Q(x^n)} = R^*,$$

where we just have max instead of average.

We can talk about best distribution, either in the max or average case, but either way, if we don't know the best distribution, we can still make an Arithmetic code using some other Q. As long as the Q we pick is close to the best distribuion, the code will be universal over all the distributions in $\mathcal{P}$. I.e. For a class of processes $\mathcal{P}$, $\exists$ strongly universal codes with expected or maximum redundancy going to zero uniformly for all $P \in \mathcal{P}$ iff $\bar{R} = o(1)$ or $R^* = o(1)$ respectively. More next time.