10-704: Information Processing and Learning **Fall 2016**

Lecture 16: Oct 26

Lecturer: Aarti Singh

Note: *These notes are based on scribed notes from Spring15 offering of this course. LaTeX template courtesy of UC Berkeley EECS dept.*

16.1 Review: Prediction and Coding

So far, we have seen several connections between prediction and coding. To summarize:

- Length of prefix codes can measure complexity of models (predictors), which can be used for designing complexity penalized ERM procedures for prediction.
- Under log-loss, i.e. for density estimation, in addition to encoding models, we can also prefix encode data using a model (predictor) and minimize the total codelength of the two codes resulting in the Minimum Description Length criteria.

  Recall that to encode the data $X_1^n = X_1, \dots, X_n$ using a predictor q, we can use Shannon code i.e. the prefix code with codelengths $\ell(X_1^n) = \lceil \log(1/q(X_1^n)) \rceil$. We can also use Huffman code based on q. If $q = p$, where p is the true data-generating distribution, then Huffman codes are the optimal (smallest length) prefix codes and both Shannon and Huffman codes satisfy

$$\frac{H(X_1^n)}{n} \le \frac{\mathbb{E}_p[\ell(X_1^n)]}{n} < \frac{H(X_1^n) + 1}{n}.$$

- We also talked about how to do sequential prediction $q(X_i|X_1^{i-1})$ and guarantee universality i.e. bound $\sup_{p\in\mathcal{P}} \textit{Redundancy}(q,p)$ or $\sup_{p\in\mathcal{P}} \textit{Regret}(q,p)$. This means the predictor is close to the best possible predictor in a class $\mathcal{P}$.

In this class, we will talk about a remaining question: **What about sequential and universal coding?** Both Shannon and Huffman codes assume access to the true data-generating distribution beforehand. Even if we had access to an estimate, for non i.i.d. source, they require computing the (estimated) probability of all length n sequence i.e. $|\mathcal{X}^n|$ probability values before any coding can take place, i.e. they are not sequential. This also means they cannot accommodate changing source distributions since the probability computations need to be repeated all over if the distribution changes. Also, in practice we would like to design codes that are universal i.e. the same encoder-decoder can be used for all distributions in some class e.g. gzip can be used to compress multiple language files - English, German, French, Italian etc. - which have different distributions of characters.

Can we use the sequential and universal predictor $q(X_i|X_1^{i-1})$ to do coding? The answer is yes and it can be done using arithmetic codes. Arithmetic codes are a family of streaming codes that code symbols sequentially assuming access to a common sequential predictor $q(X_i|X_1^{i-1})$ at the encoder and decoder.

16.2 Arithmetic Coding

Arithmetic codes map streams of symbols to a real number between 0 and 1. The first observation is that there is a one-ot-one mapping between points in $[0,1)$ and a binary code. A code $z_1,\ldots,z_m$ where $z_i \in \{0,1\}$ maps to the binary representation of a number $0.z_1z_2\ldots z_m$ which is equal to $\sum_i z_i 2^{-i}$.

Let the stream of symbols to be encoded be $x_1,\ldots,x_n$. Let's define our alphabet of symbols $\mathcal{X}$ 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 properties of the code, as we will see using the following examples.

16.2.1 Examples

See Figure 16.1. 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".

16.2.1.1 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.

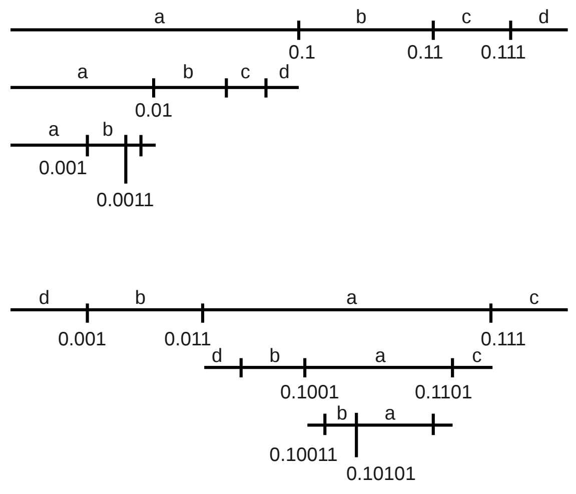

Figure 16.1: Two examples for arithmetic coding of sequence "aab" for the same iid source.

16.2.1.2 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 is because in this second example the bits in the final number don't necessarily correspond to the different subintervals. This is in contrast to Huffman codes where each bit in the encoded symbols represent a branch in the tree. However, arithmetic codes can accommodate 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.

16.2.2 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: $\frac{L(x^n)+R(x^n)}{2} = \bar{F}(x^n) = 0.z_1 z_2 \ldots z_{\tilde{m}} \ldots$. We define our rounding as $\tilde{F}(x^n) = 0.z_1 z_2 \ldots z_{\tilde{m}}$ 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 16.1 *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. ■

16.2.3 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.

16.2.4 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.

16.3 Context Tree Weighting (CTW) compression

Finally, let us mention a popular universal lossless coding algorithm known as CTW (Context Tree Weighting) which uses a specific sequential predictor to do arithmetic coding. The algorithm is both practical and theoretically sound.

Recall we saw that a sequential and universal predictor can be constructed as a mixture of experts in a class $\mathcal{P} = \{p_\theta\}_{\theta \in \Theta}$:

$$q_\pi(x_1^n) = \int_\Theta \pi(\theta) p_\theta(x_1^n) d\theta$$

where $\pi(\theta)$ is the weight of expert p_θ. This predictor can be implemented in a sequential form as:

$$q_\pi(x_i | x_1^{i-1}) = \int_\Theta \pi(\theta | x_1^{i-1}) p_\theta(x_i | x_1^{i-1}) d\theta$$

where $\pi(\theta|x_1^{i-1}) \propto \pi(\theta)p_\theta(x_1^{i-1})$ is the weight of expert p_θ at iteration i. We will refer to $\pi(\theta|x_1^{i-1})$ as $\pi^i(\theta)$.

The CTW method uses a specific mixture of "tree" experts as a sequential and universal predictor and performs arithmetic coding using this predictor. Each tree expert is a D-bounded variable-order Markov chain associated with a suffix set $\mathcal{S}$ (think $\theta \equiv \mathcal{S}$) where each suffix (also known as context) $s \in \mathcal{S}$ is a path from a leaf to the root in a depth D bounded $|\mathcal{X}|$-ary tree. For example, three $D = 2$ bounded binary tree experts are shown in Figure 16.2 with suffix set (or context) $\mathcal{S}_a = \{00, 10, 01, 11\}, \mathcal{S}_b = \{0, 01, 11\}$ and $\mathcal{S}_c = \{\emptyset\}$. Each leaf node is annotated with $p_\mathcal{S}(x_i|x_{i-D}^{i-1} = s)$, the probability of next symbol being 0/1 given the context s. The probability of a sequence x_1^n under tree expert $\mathcal{S}$ is $p_\mathcal{S}(x_1^n) = \prod_{i=1}^n p_\mathcal{S}(x_i|x_{i-D}^{i-1})$ by the chain rule. Thus, each tree-expert is a variable-order Markov chain since the order of dependency on the past can be different depending on the context (but it is bounded by D).

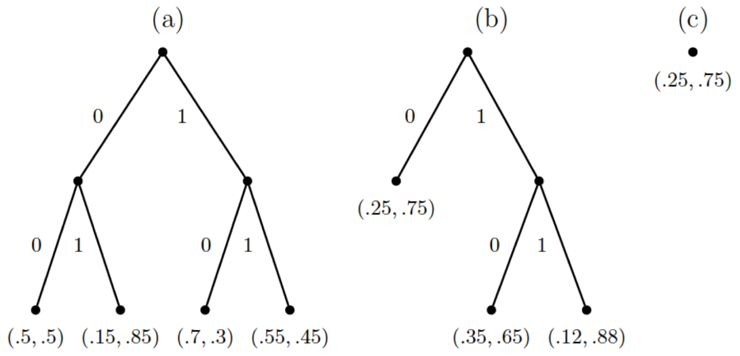

Figure 16.2: Example of three $D = 2$ bounded binary tree experts with suffix set (or context) $\mathcal{S}_a = \{00, 10, 01, 11\}, \mathcal{S}_b = \{0, 01, 11\}$ and $\mathcal{S}_c = \{\emptyset\}$. The probability of generating $x_1^3 = 100$ given initial context 00 are $p_{\mathcal{S}_a}(00; 100) = p_{\mathcal{S}_a}(00; 1)p_{\mathcal{S}_a}(01; 0)p_{\mathcal{S}_a}(10; 0) = 0.5 \cdot 0.7 \cdot 0.15$, $p_{\mathcal{S}_b}(00; 100) = p_{\mathcal{S}_b}(0; 1)p_{\mathcal{S}_b}(01; 0)p_{\mathcal{S}_b}(0; 0) = 0.75 \cdot 0.35 \cdot 0.25$ and $p_{\mathcal{S}_c}(00; 100) = p_{\mathcal{S}_c}(1)p_{\mathcal{S}_c}(0)p_{\mathcal{S}_c}(0) = 0.75 \cdot 0.25 \cdot 0.25$. The figure is reproduced from Figure 1 of http://www.jmlr.org/papers/volume7/begleiter06a/begleiter06a.pdf.

The CTW forms a mixture of these exponentially-many D-bounded tree experts in an efficient way. The initial weight of an expert $\mathcal{S}$ is

$$\pi(\mathcal{S}) = 2^{-|T_\mathcal{S}|}$$

and the weight at iteration i is

$$\pi^i(\mathcal{S}) \propto 2^{-|T_\mathcal{S}|}p_\mathcal{S}(x_1^{i-1})$$

as in mixture of experts, where $|T_\mathcal{S}|$ = number of nodes in $\mathcal{S}$ tree − number of leaf nodes at max depth D. We do not discuss how this infinite mixture can be implemented efficiently, but refer to sec 4.2 in http://alexandria.tue.nl/extra2/200213835.pdf for a nice explanation of how to implement it by recursively updating probabilities on a full $|\mathcal{X}|$-ary tree of depth D.

So far in the course, we have talked about lossless compression. Going further, we will begin talking about lossy compression. We will also make connections to sufficient statistics and information bottleneck principle in machine learning. We start with sufficient statistics.

16.4 Sufficient statistic

Sufficient statistics provide a (usually short) summary of the data, and are often useful for learning tasks.

A **statistic** is any function $T(X)$ of the data X. if θ parametrizes the class of underlying data-generating distributions, then for any statistic, we have the Markov chain

$$\theta \to X \to T(X)$$

i.e. $\theta \perp T(X)|X$ and data processing inequality tells us that $I(\theta, T(X)) \leq I(\theta, X)$.

A statistic is **sufficient** for a parameter θ if $\theta \perp X|T(X)$, i.e. we also have the Markov chain

$$\theta \to T(X) \to X.$$

In words, once we know $T(X)$, the remaining randomness in X does not depend on θ. This implies $p(X|T(X))$ does not depend on θ (this is a useful characterization when you may not want to think of θ as a random variable). Also, $I(\theta, T(X)) = I(\theta, X)$.

Examples of sufficient statistics:

1. $X = X_1, \ldots, X_n \sim \text{Ber}(\theta)$. Then $T(X) = \frac{1}{n}\sum_{i=1}^n X_i$ is sufficient for θ.
2. $X = X_1, \ldots, X_n \sim \text{Uniform}(\theta, \theta+1)$. Then $T(X) = \{\min_i X_i, \max_i X_i\}$ is sufficient for θ.

Note: A sufficient statistic can be multi-dimensional as the last example shows.
Note: A sufficient statistic is not unique. Clearly, X itself is always a sufficient statistic.

How do we find a sufficient statistic?

Theorem 16.2 (Fisher-Neyman Factorization Theorem) *$T(X)$ is a sufficient statistic for θ iff $p(X;\theta) = g(T(X), \theta)h(X)$.*

Here $p(X;\theta)$ is the joint distribution if θ is random, or is the likelihood of data under p_θ otherwise.

Lets consider the second example above. Then

$$p(X, \theta) = \prod_{i=1}^n p_\theta(X_i) = \prod_{i=1}^n 1_{X_i \in (\theta, \theta+1)} = 1_{\theta \leq \min_i X_i \leq \max_i X_i \leq \theta+1} = g(T(X), \theta)$$

If instead $X = X_1, \ldots, X_n \sim \text{Uniform}(0, \theta)$, then $T(X) = \max_i X_i$ is a sufficient statistic since

$$p(X, \theta) = \prod_{i=1}^n p_\theta(X_i) = \prod_{i=1}^n 1_{X_i \in (0, \theta)} = 1_{0 \leq \min_i X_i \leq \max_i X_i \leq \theta+1} = 1_{0 \leq \min_i X_i} \, 1_{\max_i X_i \leq \theta+1} = h(X)g(T(X), \theta)$$

A sufficient statistic $T(X)$ is **minimal** if $T(X) = g(S(X))$ for all sufficient statistics $S(X)$. For example, if $X = X_1, \ldots, X_n \sim \text{Ber}(\theta)$, then $T(X) = \frac{1}{n}\sum_{i=1}^n X_i$ is a minimal sufficient for θ but $S(X) = X$ is not.

Note: A minimal sufficient statistic is not unique. For the above example, both $\frac{1}{n}\sum_{i=1}^n X_i$ and $\sum_{i=1}^n X_i$ are minimal sufficient statistic.