Lecture 6:

Typical Sets and H_δ

September 27, 2006

Reminder: Block Codes for Achieving the Entropy 1

- Last class we proved that Huffman codes are the optimal single symbol codes (plus a warning: top-down splitting does not work).
- We also proved Shannon's first theorem by showing that if we encode long enough blocks we can get the average per-symbol entropy as close as we want to the entropy of the source.
- Our proof used Shannon-Fano or Huffman codes for blocks of N symbols. These codes are instantaneously decodable symbols codes of *variable length* (some blocks had codes longer than other blocks), which are guaranteed to get within an additive constant (one bit) of the entropy.
- There is another way to compress down to the entropy using long blocks. It uses codes which *cannot always be correctly decoded* and which are of *fixed length* (all blocks get codes of the same size).

Another Way to Compress Down to the Entropy 2

- Suppose we always encode N symbols into a block of exactly NR bits (fixed length code).
  Q: Can we do this in a way that is very likely to be decodable?
- A:Yes, for large values of N.
  The Law of Large Numbers (LLN) tells us that the sequence of symbols $a_{i_1}, \ldots, a_{i_N}$, is very likely to be a "typical" one, for which

$$\frac{1}{N}\log_2(1/(p_{i_1}p_{i_2}\cdots p_{i_N})) \;=\; \frac{1}{N}\sum_{j=1}^{N}\log_2(1/p_{i_j})$$

  is very close to the expectation of $\log_2(1/p_i)$, which is the entropy, $H(X) = \sum_i p_i \log_2(1/p_i)$. (See Section 4.3 of MacKay's book.)
- So if we encode all the sequences in this *typical set* in a way that can be decoded, the code will almost always be uniquely decodable.

How Big is the Typical Set? 3

- Let's define "typical" sequences as ones where

$$(1/N)\log_2(1/(p_{i_1}\cdots p_{i_N})) \;\leq\; H(X) + \eta/\sqrt{N}$$

  The probability of any such typical sequence will satisfy

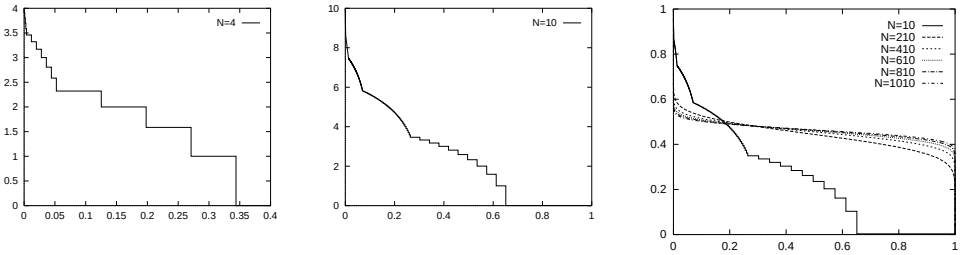
$$p_{i_1}\cdots p_{i_N} \;\geq\; 2^{-NH(X)-\eta\sqrt{N}}$$

- We scale the margin allowed above $H(X)$ as $1/\sqrt{N}$ since that's how the standard deviation of an average scales. The LLN (Chebychev's inequality) then tells us that most sequences will satisfy this condition, for some large enough value of η.
- The total probability for all such sequences can't be greater than one, so the number of "typical" sequences can't be greater than

$$2^{NH(X)+\eta\sqrt{N}}$$

- The number of "typical" sequences can't be greater than
$$2^{NH(X)+\eta\sqrt{N}}$$
- We will be able to encode these sequences in NR bits if $NR \geq NH(X) + \eta\sqrt{N}$. (Using any arbitrary/random code in which we assign each typical sequence to one of the 2^{NR} codes.) If $R > H(X)$, this will always be true if N is sufficiently large.
- How often will a sequence of length N fail to be in the typical set? To answer this, we need to know how many sequences live in the upper "tail" of the distribution of $(1/N)\log_2(1/(p_{i_1} \cdots p_{i_N}))$.
- We can define $H_\delta(X^N)$ to be average codeword length needed for the typical set to leave out only a fraction δ of possible sequences. Formally, it is the logarithm of the minimum number of sequences in the N^{th} extension of X whose probabilities sum to at least $1 - \delta$.

- Example: Consider flipping a coin with $p_{heads} = 0.1$.
- Here are the plots of δ vs. H_δ.



- For large N, H_δ becomes almost independent of δ.

- Let X be an ensemble with entropy $H(X) = H$ bits.
- Given *any* $\epsilon > 0$ and $0 < \delta < 1$, there exists a positive integer N_0 such that for $N > N_0$,
$$H - \epsilon \; < \; \frac{1}{N} H_\delta(X^N) \; < \; H + \epsilon$$
- Both sides of the inequality are interesting. The first part tells us that even if the probability of error δ is extremely small, the average number of bits per symbol $\frac{1}{N}H_\delta(X^N)$ needed to specify a long N-symbol string with vanishingly small error probability does not have to exceed $H + \epsilon$ bits.
- In other words, we need to have only a tiny tolerance for error, and the number of bits required drops significantly from $H_0(X)$ to $(H + \epsilon)$.

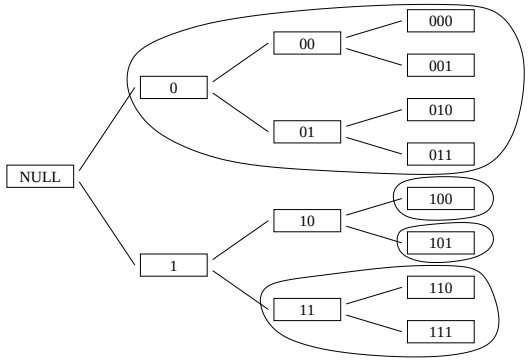
- Let X be an ensemble with entropy $H(X) = H$ bits.
- Given *any* $\epsilon > 0$ and $0 < \delta < 1$, there exists a positive integer N_0 such that for $N > N_0$,
$$H - \epsilon \; < \; \frac{1}{N} H_\delta(X^N) \; < \; H + \epsilon$$
- What happens if we are yet more tolerant to compression errors? The second part tells us that even if δ is very close to 1, so that errors are made most of the time, the average number of bits per symbol needed must still be at least $H - \epsilon$ bits.
- These two extremes tell us that regardless of our specific allowance for error, the number of bits per symbol needed is essentially H bits; no more and no less.

- If we *require* error-free decoding (i.e. we cannot tolerate even a δ probability of failure), we can always encode the "atypical" blocks (sequences) by reserving one codeword (say the all-zeros bitstring) as a prefix for those blocks and using any UD code after that.
- Since this will only occur a fraction δ of the time it won't affect the average length at all and so the arguments above are still valid.

Shannon's Noiseless Coding Theorem is mathematically satisfying.
From a practical point of view, though, we still have two problems:

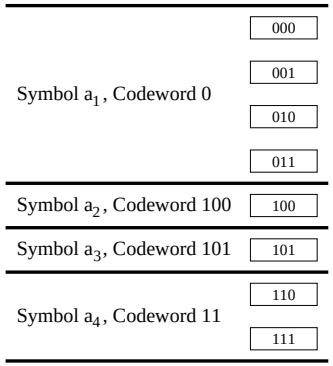
- How can we compress data to nearly the entropy *in practice*? The number of possible blocks of size N is I^N — huge when N is large. And N sometimes must be large to get close to the entropy by encoding blocks of size N.
  Solution: Instead of symbol codes or block codes, we will introduce a more powerful set of codes called *stream codes*. The most important example is known as *arithmetic coding* (coming next).
- Where do the symbol probabilities $p_1, \ldots, p_I$ come from? And are symbols really independent, with known, constant probabilities? This is the problem of *source modeling*.
  Solution: *adaptive methods*, which update their estimates of the source model as they encode more and more data.
  (We'll see these shortly.)

- Any instantaneous code can be represented by a tree such as the following, with subtrees for codewords circled:



Rather than concentrate on the codewords that head each subtree, let's concentrate on the leaves...

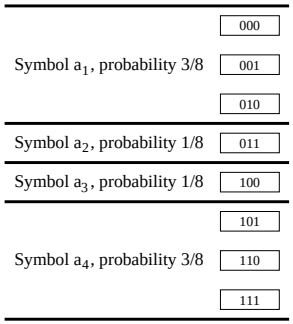
- Here are the codetree leaves, divided up according to codeword:



If we view $\{000, 001, 010, 011, 100, 101, 110, 111\}$ as an available "codespace", we see that this code divides it up so that symbol a_1 gets $1/2$ of it, symbols a_2 and a_3 get $1/8$, and symbol a_4 gets $1/4$.

What About Other Divisions? 12

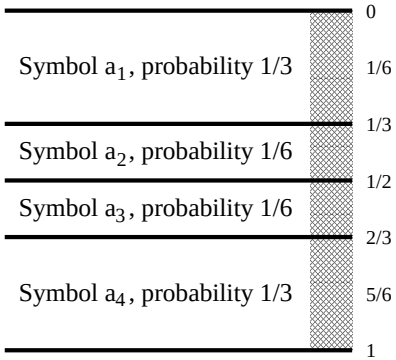
- We know that this code is optimal if the fraction of codespace assigned to a symbol is equal to the symbol's probability.
- But suppose the symbol probabilities were $3/8, 1/8, 1/8, 3/8$. We would then like to divide up codespace as follows:



- Unfortunately, these divisions don't correspond to subtrees — so there's no prefix-free code like this.

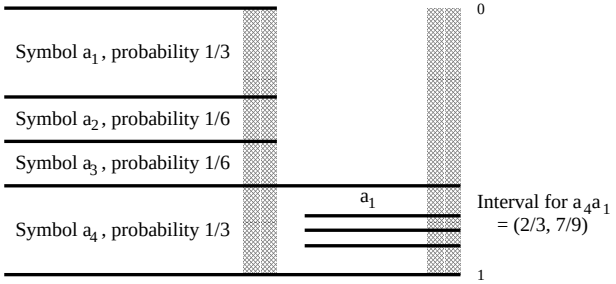
Viewing the Codespace as the Interval [0,1) 13

- Even if we could solve the problem of how to generate codewords corresponding arbitrary divisions of codespace, how can we handle symbols with probabilities like $1/3$, which aren't multiples of 2^{-k}?
- A solution: Consider the codespace to be the interval of real numbers between 0 and 1. Example:



Key Concept: Encode Blocks by Subdiving Further 14

- Consider the source with probabilities $\{1/3, 1/6, 1/6, 1/3\}$.
- Suppose we want to encode blocks of two symbols from this source. We can do this by just subdividing the interval corresponding to the first symbol in the block, in the same way as we subdivided the original interval.
- Here's, how we encode the block $a_4\, a_1$:



Encoding Large Blocks as Intervals 15

- A general scheme for encoding a block of N symbols, $a_{i_1}, \ldots, a_{i_N}$

  1) Initialize the interval to $\left[u^{(0)},\ v^{(0)}\right)$; $u^{(0)} = 0$ and $v^{(0)} = 1$.

  2) For $k = 1, \ldots, N$:

  Let $u^{(k)} = u^{(k-1)} + \left(v^{(k-1)} - u^{(k-1)}\right) \sum_{j=1}^{i_k - 1} p_j$

  Let $v^{(k)} = u^{(k)} + \left(v^{(k-1)} - u^{(k-1)}\right) p_{i_k}$

  3) Output a codeword that corresponds (somehow) to the final interval, $\left[u^{(N)},\ v^{(N)}\right)$.

- This scheme is known as *arithmetic coding*, since codewords are found using arithmetic operations on the probabilities.