EE5585 Data Compression January 31, 2013

Lecture 4

Instructor: Arya Mazumdar *Scribe: John Havlik*

Huffman Code (Review)

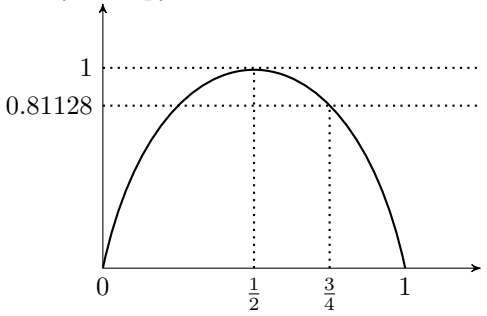
Huffman coding has been shown to be optimal for sources with a discrete alphabet. For any source, given a distribution, a Huffman code is the best code for compressing the source. A Huffman code is formed using a Huffman tree.

Example: Given a sequence of binary numbers: 01000110100... and that $Pr(0) = \frac{3}{4}$ $Pr(1) = \frac{1}{4}$
Calculate the entropy:

$$\begin{aligned} H &= -\sum_i P_i \log_2 \frac{1}{P_i} \\ &= -\frac{3}{4}\log_2 \frac{3}{4} - \frac{1}{4}\log_2 \frac{1}{4} \\ &= \frac{1}{2} + \frac{3}{2} - \frac{3}{4}\log_2 3 \\ &= 2 - \frac{3}{4}\log_2 3 \\ &\approx 0.81128 \end{aligned}$$

Bernoulli (p) Distribution

Is a distribution on $\{0, 1\}$ where $Pr(0) = p$
Binary entropy function for a Bernoulli experiment:



For the above example, with $Pr(0) = \frac{3}{4}$, we found an entropy of $H = 0.81128$. This is our limit for lossless compression, we can not compress past this. If we used Huffman code with binary alphabet we do not get any compression since it will, for the above example, assign 0 to 0 and 1 to 1 resulting in the exact sequence we started with. However, looking at the binary entropy function plot above, we see that we should be able to achive some compression.

How can we still use Huffman code to compress this? In the single symbol at a time case, we end up with $Pr(0) = \frac{1}{2}$ $Pr(1) = \frac{1}{2}$ which is not efficient. But, we can club multiple symbols together.

Given random variable X taking value from a set $\mathcal{X}$,

$$H(X) = -\sum_{x \in \mathcal{X}} Pr(x) \log_2 Pr(x)$$

This is known to be at its maximum when x is uniform. If X is uniform we know that $Pr(X = x) = \frac{1}{|\mathcal{X}|}$

where $\mid \mathcal{X} \mid$ represents the size of $\mathcal{X}$. The entropy for a uniform distribution is $H(X) = \log_2 \mid \mathcal{X} \mid$. We will show that this is the max value the entropy can take. The above example is a special case where we have a uniform distribution over a binary alphabet. Since we know H is at a maximum when we have uniform distribution, the maximum value for our case is $\log_2 2 = 1$.

Lemma 1

$$\begin{aligned} \ln x &\leq x - 1 \\ -\ln x &\geq 1 - x \end{aligned}$$

with equality only when x = 1

Theorem 2

$$H(X) \leq \log_2 \mid \mathcal{X} \mid .$$

Proof Show: $\log_2 \mid \mathcal{X} \mid -H(X) \geq 0$ met with equality only when x is uniform
Let $\mid \mathcal{X} \mid = M$

$$\begin{aligned} \log_2 \mid \mathcal{X} \mid -H(X) &= -\sum_i P_i \log_2 \frac{1}{M} + \sum_i P_i \log_2 P_i \\ &= -\sum_i P_i \log_2 \left(\frac{\frac{1}{M}}{P_i} \right) \\ &= -\log_2 e \sum_i P_i \ln \left(\frac{\frac{1}{M}}{P_i} \right) \\ &\geq -\log_2 e \sum_i P_i \left(1 - \frac{\frac{1}{M}}{P_i} \right) = -\log_2 e \left(\sum_i P_i - \sum_i \frac{1}{M} \right) = 0 \end{aligned}$$

Note that i is index of source symbols we could have used sum of x, $Pr(x)$. In the second step we group the two $\log_2$ terms using properties of logs. We must now change from $\log_2$ to ln to use the lemma. After using the lemma, we are left with $\sum_i P_i - \sum_i \frac{1}{M}$. Both summations add to 1, causing the pair to equal 0.

Note that per the same conditions found in Lemma 1, equality occures only when $P_i = \frac{1}{M}$, this is a characteristic of uniform distributions. Additonally, this happens to be a special case of more general topic. ■

Relative Entropy (Kullback-Leibler divergence)

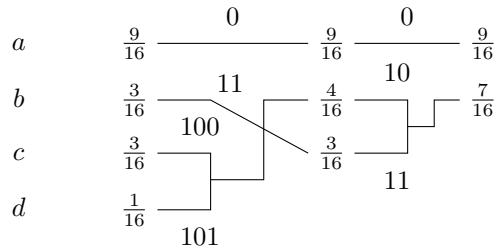
Suppose on same alphebet X we have two distributions $p = p(x)$ $q = q(x)$

Divergence: $D(p||q) = \sum_{x \in X} Pr(x) \log_2 \left(\frac{p(x)}{q(x)} \right)$

In a way, the divergence is the distance between the two distributions. Can we prove $D(p||q) \geq 0$, and when is this inequality true? The previous proof covered this inequality, and using the result from that proof, we have $D(p||q) = 0$ when $p \equiv q$. That is $p(x) = q(x) \ \ \forall \ x \in X$.

Huffman coding, as we've discussed thus far, doesn't show us how to handle single bit per character cases. We know we can do better since $H < 1$. Rather than just looking at one symbol at a time, let's look at pairs.

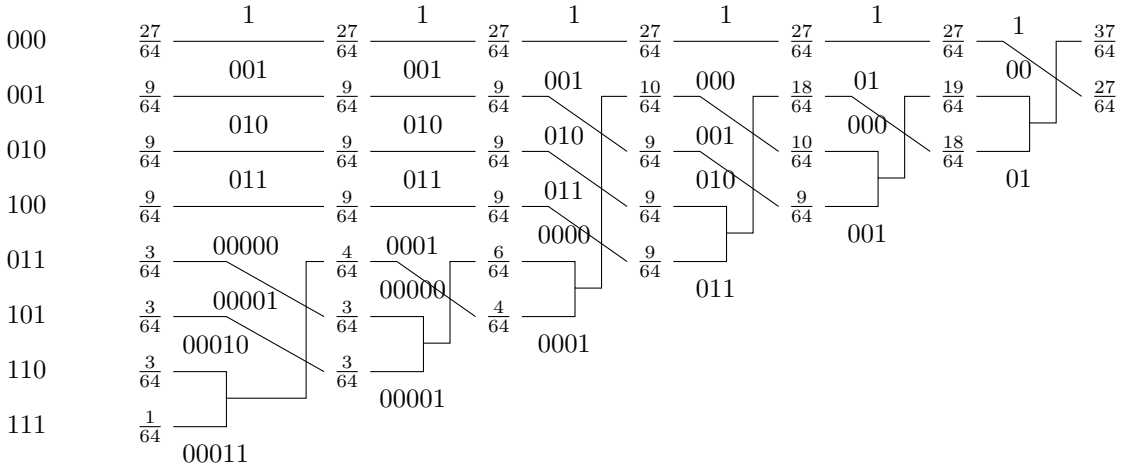
Example: Suppose we have $Pr(00) = \frac{9}{16}$ $Pr(01) = \frac{3}{16}$ $Pr(10) = \frac{3}{16}$ and $Pr(11) = \frac{1}{16}$ with $a = 00$ $b = 01$ $c = 10$ and $d = 11$



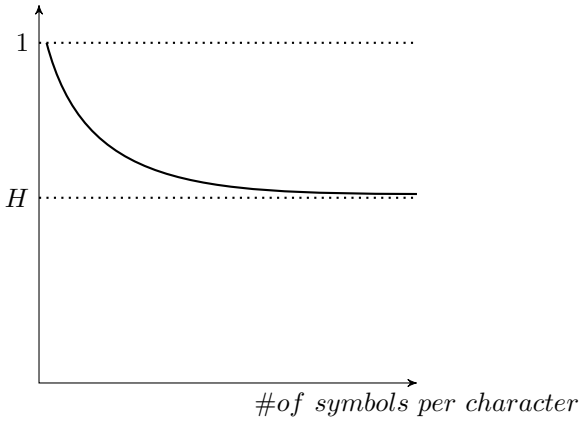
Average length of a codeword with this scheme:

$$\begin{aligned} \text{Ł}(C) &= 1 * \frac{9}{16} + 2 * \frac{3}{16} + 3 * \frac{3}{16} + 3 * \frac{1}{16} \\ &= \frac{9}{16} + \frac{6}{16} + \frac{9}{16} + \frac{3}{16} \\ &= \frac{27}{16} \ bits/char \\ &= \frac{27}{32} \ bits/symbol \approx 0.84375 \end{aligned}$$

We have a 2 symbol character, so we get 27/32 bits/symbol. We have achieved some compression. Since we are not at the lower limit of 0.81128 set by the entropy of the problem we can compress this more. Lets try 3 symbols per character.



Note that probabilities follow the number of 0's in a character. The average number of bits/symbol is 0.8229. This is getting closer to the entropy of the problem.



As the number of symbols grouped in a character increases, the average number of bits/symbol gets closer to the entropy. The size of alphabet is going to grow exponentially (2^n for n symbols in a character). Typically, 10 to 20 symbols need to be grouped into a character to get close to the entropy. This becomes very costly to generate the tree, as a result Huffman coding is not considered to be "on the fly". Another algorithm that is good for smaller words can help us. The problem we ran into is the size of the word is too small and Huffman is not very efficient in this case.

Arithmetic Coding

Arithmetic coding is the scheme used in Fax machines and in JPEG. Before we talk about arithmetic coding we fist need to know something about a continuous distribution. Let $x \rightsquigarrow U[0,1)$ when $Pr(a \leq x \leq b) = b - a \quad \forall\ 0 \leq a \leq b \leq 1$. If you have any real number between 0 and 1 you can write a binary or decimal expansion for that number. This can be infinite, and infact most real numbers require an infinite series representation.

Example:

$$\begin{aligned} a &= 0.1011010001\ldots \\ &= 1 * \frac{1}{2} + 0 * \frac{1}{4} + 1 * \frac{1}{8} + \ldots \end{aligned}$$

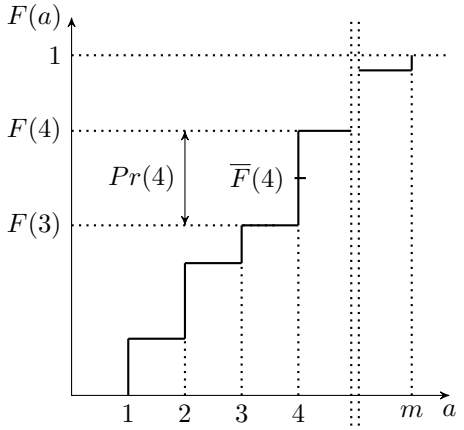
$1 * \frac{1}{2}$, $0 * \frac{1}{4}$, etc. are atoms of this representation.

Proposition: $x \rightsquigarrow U[0,1)$ then in the binary representation of x, the symbols are Bernoulli$\left(\frac{1}{2}\right)$ and independent. That is, for each position in a binary expansion for x we will have a Bernoulli$\left(\frac{1}{2}\right)$ random variable (each position takes the value of either 0 or 1 with $p = \frac{1}{2} \quad 1 - p = \frac{1}{2}$).This has the implication that for each position $Pr(0) = \frac{1}{2} \quad Pr(1) = \frac{1}{2}$, the entropy of the sequence is 1 and thus incompressible.
Proof Suppose we have $x = 0.010111001101\ldots$
The first term being 0 tells us that the random variable x is between 0 and $\frac{1}{2}$, since we have an interval of 0 to 1. The probability of the first term being 0 is $\frac{1}{2}$. Now we have a new interval between 0 and $\frac{1}{2}$. For the second term we must normalize for the new interval, which results in the probability being $\frac{1}{2}$ again. We can keep doing this for all of the terms, and we will keep getting $Pr = \frac{1}{2}$ due to normalizing for the new intervals. Thus, the terms are independent as the position ends up not mattering for the probability. ■

If you have a uniform random variable, each position ends up being 1 bit, and you can not compress it losslessly.

Shannon-Fano-Elias Coding

Suppose, without loss of generality, $\mathcal{X} = 1, 2, \ldots, m$.
$Pr(1), Pr(2), \ldots, Pr(m)$
We have a CDF of:
$F(a) = \sum_{x \leq a} Pr(x)$
Note that we have not sorted the members, we just took them as supplied. Sorting is one of the disadvantages of Huffman coding as it requires $O(n \log n)$ time. As long as we are talking about discrete random variables the CDF is easy to define.



$\overline{F}(a) = F(a-1) + \frac{1}{2}F(a)$
$\overline{F}(a)$ will be our code and happens to be a real number. We need to truncate this and store the remaining number of bits. Take the first $l(a)$ bits of the binary representation of $\overline{F}(a)$. This is the code for a. The claim is this will be a prefix free code. This is due to the fact that we select $l(a)$ so that the points we end up with are in disjoint intervals. We define $l(a)$ to be:

$$l(a) = \left\lceil \log_2 \frac{1}{Pr(a)} \right\rceil + 1$$

Our truncated value $\left\lfloor \overline{F}(a) \right\rfloor_{l(a)}$ truncated on $l(a)$

$$\begin{aligned} \overline{F}(a) - \left\lfloor \overline{F}(a) \right\rfloor_{l(a)} &< \frac{1}{2^{l(a)}} \\ &\leq \frac{Pr(a)}{2} \end{aligned}$$

Essentially, this is how far we are moving from the middle point due to truncation. In the worst case we are moving from $0.00\ldots0|1111111111111\ldots$ to $0.00\ldots01|$. Even if after truncating we still remain in the originial interval. Thus, the code will be prefix free since each code is in its own interval. Average number of bits/symbol:

$$\begin{aligned} &\leq \sum_{x \in \mathcal{X}} Pr(x) \left(\log_2 \frac{1}{Pr(x)} + 2 \right) \\ &\leq H(X) + 2 \end{aligned}$$

With Huffman we're between H and $H+1$, so this is not as good as Huffman. However, this doesn't require sorting and can be done "on the fly". We just need to find the $F(a)$, which is just addition and is computationally easier than sorting.

Arithmetic Coding

Arithmetic coding is similar to SFE coding, but combines large number of subsequences together first. Then you try to find the CDF for that combined sequence.
Suppose you have a continuous random variable Y then $F_Y(y)$ is the Cummlative Distribution Function of Y, that is $F_Y(y) = Pr(Y \leq y)$
Define a new random variable that is a function of Y

Lemma 3 *$Z = F_Y(Y)$ is a uniform random variable $Z \approx U[0,1)$*

Proof

$$\begin{aligned} Pr(Z \leq z) &= Pr(F_Y(Y) \leq z) \\ &= Pr(Y \leq F_Y^{-1}(z)) \\ &= F_Y(F_Y^{-1}(z)) \\ &= z \end{aligned}$$

$\therefore z$ is uniform
Note that F_Y is well defined so we can take the inverse. ■

Note that $Pr(Z \leq z) = z$ is the definiton of a uniform random variable. We started with continuous random variable and mapped it to a new random variable using the CDF of the first random variable. The new random variable has a range between 0 and 1 due the nature of the CDF. From the proof we see that this new random variable is a uniform random variable. We know all positions will be Bernoulli$\left(\frac{1}{2}\right)$ and that we have mapped a random variable to sequence of incompressable bits. Thus, we must have encountered compression if we started with a compressable sequence and ended with an incompressable sequence.

Example: Let $x = 0.010001010\ldots$ which is a random variable
$Pr(0) = 0.75$
$Pr(1) = 0.25$
$F_X(x) = Pr(X \leq x)$
$X = 0.X_1X_2\ldots$
$x = 0.x_1x_2\ldots$
$Pr(0.X_1X_2\ldots \leq 0.x_1x_2\ldots)$
$= Pr(X_1 < x_1) + Pr(X_1 = x_1, X_2 < x_2) + Pr(X_1 = x_1, X_2 = x_2, X_3 < x_3) + \ldots$

Example: Suppose we have the bit sequence 01001 with $Pr(0) = 0.75 = p$ and $Pr(1) = 0.25 = 1-p = q$

$$\begin{aligned} F(01001) &= Pr(X_1 < 0) + Pr(X_1 = 0, X_2 < 1) \\ &\quad + Pr(X_1 = 0, X_2 = 1, X_3 < 0) \\ &\quad + Pr(X_1 = 0, X_2 = 1, X_3 = 0, X_4 < 0) \\ &\quad + Pr(X_1 = 0, X_2 = 1, X_3 = 0, X_4 = 0, X_5 < 1) \end{aligned}$$

Note that the calculation of any single term can be done by calculating the previous term plus a modifier. This gives us a real number, which we then truncate.