EE5585 Data Compression January 22, 2013

Lecture 1

Instructor: Arya Mazumdar *Scribe: Arya Mazumdar*

Syllabus

- Preliminaries: Probability, Discrete Mathematics, Information Theory
- Lossless coding: Source Coding Theory(Prefix-free and Universally Decodable Codes), Huffman Coding, Arithmetic Codes, The Lempel-Ziv Algorithm
- Lossy Coding: Rate-distortion Theory, Scalar and Vector Quantization, Transform Coding Techniques, Subband Coding, Predictive Coding, Case Studies (JPEG/MPEG)
- Compressed Sensing, Update/Query Efficient Compression.

Compressible Data

Some of the data are more likely than others.

Example: The source produces a binary $\{0,1\}$-vector of length 6. The number of 1s in the vector is exactly 2. We know that there are $\binom{6}{2} = 15$ such sequences.

Data	Index	Compressed Data
110000	0	0000
101000	1	0001
100100	2	0010
100010	3	0011
100001	4	0100
011000	5	0101
010100	6	0110
010010	7	0111
010001	8	1000
001100	9	1001
001010	10	1010
001001	11	1011
000110	12	1100
000101	13	1101
000011	14	1110

Table 1: A scheme showing six bit compressed to four bits

Given a source sequence, say 010010, we find out its index from the above table and represent it in binary. The rate of compression here is $\frac{4}{6} = \frac{2}{3}$.

Let us generalize this example to the case where the source can produce any n length binary sequence with exactly k ones. It is clear that there are $\binom{n}{k}$ such sequences. Hence a rate of compression

$$\left\lceil \frac{1}{n} \log_2 \binom{n}{k} \right\rceil,$$

is achievable as well as best possible. We may use Stirling approximation, $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ to evaluate this compression rate. Indeed, we have,

$$\frac{1}{n}\log_2 \binom{n}{k} \approx \frac{1}{2n}\log_2 \frac{n}{2\pi k(n-k)} - \frac{k}{n}\log_2 \frac{k}{n} - (1-\frac{k}{n})\log_2(1-\frac{k}{n}).$$

Hence,

$$\begin{aligned}\lim_{n\to\infty} \frac{1}{n}\log_2 \binom{n}{k} &= -\frac{k}{n}\log_2 \frac{k}{n} - (1-\frac{k}{n})\log_2(1-\frac{k}{n}) \\ &= h(k/n),\end{aligned}$$

where

$$h(p) = -p\log_2 p - (1-p)\log_2(1-p)$$

is called the *binary entropy function*.

Note that data compression was possible because the set of feasible sequences was of a much smaller size than the ambient space.

Entropy

Suppose a random variable X has range the finite set $\mathcal{X}$. Let, $\mathbb{P}(X = x) \equiv p(x)$ for all $x \in \mathcal{X}$. Define,

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

to be the entropy of the random variable X. Note that the entropy is a function of the probability distribution and not the random variable itself.

Transform Coding

Let us look at another model of data. Let $\mathcal{Y} = \{0, 1, \ldots, 255\}$ and let the source is $\mathcal{X} \subset \mathcal{Y}^7$. However, for any $x = (x_1, \ldots, x_7) \in \mathcal{X}$, $|x_i - x_{i+1}| < 32$, for $1 \le i \le 6$. For example, one possible source sequence might be

$$(192, 218, 200, 170, 160, 160, 140).$$

We could have enumerated the number of such sequences. However, we choose to multiply the source sequence with the following matrix.

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Upon multiplication with a source sequence, say $(192, 218, 200, 170, 160, 160, 140)^T$, we will always obtain an array whose entries are between -32 and 32. Because the matrix is invertible, it is easy to get back the original sequence. This is a change of basis for the source sequence. As the amplitude of the source in this new basis has a significantly smaller range (-32 to 32) than the original one (0 to 255), data compression is immediate.

This change of basis for dimensionality reduction or data compression is called transform coding. In the most studied cases this transformation is unitary, for example multiplying with a DFT (discrete Fourier transform) matrix.

Source Coding

Suppose the finite set $\mathcal{X}$ is the source and for any $x \in \mathcal{X}$, $p(x)$ denotes the probability of x. We assume that the encoder will assign binary $\{0,1\}$ valued vectors (codewords) for each symbol. Let for the code $\mathcal{C}$, $C(x)$ denote the codeword corresponding to $x \in \mathcal{X}$. Also let $\ell(x)$ denote the length of $C(x)$.

Our measure of compression will be the expected value of $\ell(x)$, i.e., the average number of bits per source symbol. We denote it by

$$L(\mathcal{C}) \equiv \mathbb{E}\ell(x) = \sum_{x \in \mathcal{X}} p(x)\ell(x).$$

Example: Let $\mathcal{X} = \{1,2,3\}$ and $p(1) = \frac{5}{6}, p(2) = \frac{1}{12} = p(3)$. Let for a code $\mathcal{C}$, $C(1) = 0, C(2) = 10, C(2) = 11$. Hence,

$$L(\mathcal{C}) = \frac{5}{6} + \frac{1}{6} + \frac{1}{6} = \frac{7}{6}.$$

For unambiguous decoding it is apparent that we need no two source symbol to be mapped to the same codeword. That is, for all $x, x' \in \mathcal{X}$ such that $x \neq x'$, $C(x) \neq C(x')$. Such codes are called *nonsingular code*.

Example: Suppose $\mathcal{X} = \{1,2,3,4\}$ and $C(1) = 0, C(2) = 1, C(3) = 01, C(4) = 101$. This code is nonsingular. But note that, source sequences $(1,4)$ and $(3,3)$ produce the same bit stream 0101.

A code is called *uniquely decodable* if for any $\{0,1\}$-vector there is at most one sequence of source symbols that can produce it up on encoding.

Example: Suppose $\mathcal{X} = \{1,2,3,4\}$ and $C(1) = 00, C(2) = 10, C(3) = 11, C(4) = 110$. This code is uniquely decodable. Take for example the encoded sequence 10110100011110 that decodes uniquely to 242134.

Theorem 1 *For any uniquely decodable code $\mathcal{C}$ for the source $\mathcal{X}$ with source symbols following the distribution of random variable X,*

$$L(\mathcal{C}) \geq H(X).$$

We need the following famous result to prove this.

Lemma 2 (Kraft inequality) *For any uniquely decodable code for source $\mathcal{X}$,*

$$\sum_{x \in \mathcal{X}} 2^{-\ell(x)} \leq 1.$$

Proof Consider, for any $k > 1$, the following term.

$$\begin{aligned} \Big[\sum_{x \in \mathcal{X}} 2^{-\ell(x)}\Big]^k &= \textstyle\sum_{x_1 \in \mathcal{X}} \sum_{x_2 \in \mathcal{X}} \cdots \sum_{x_k \in \mathcal{X}} 2^{-\ell(x_1)-\ell(x_2)\ldots-\ell(x_k)} \\ &= \textstyle\sum_{x_1^k \in \mathcal{X}^k} 2^{-\ell(x_1^k)}, \end{aligned}$$

where x_1^k is a super symbol consisting of k source symbols. Suppose $a(m)$ is the number of such symbols such that $\ell(x_1^k) = m$. As the code is uniquely decodable $a(m) \leq 2^m$. Hence,

$$\begin{aligned} \Big[\sum_{x \in \mathcal{X}} 2^{-\ell(x)}\Big]^k &= \textstyle\sum_m a(m) 2^{-m} \\ &\leq \textstyle\sum_m 1 \\ &= k \max_{x \in \mathcal{X}} \ell(x). \end{aligned}$$

If the Kraft inequality is not satisfied, the left hand side will grow exponentially with k whereas the right hand side grows only linearly. For sufficiently large k we hence will have a contradiction. Therefore,

$$\sum_{x \in \mathcal{X}} 2^{-\ell(x)} \leq 1.$$

■

EE5585 Data Compression January 24, 2013

Lecture 2

Instructor: Arya Mazumdar *Scribe: Trevor Webster*

Announcements

- Class Webpage - http://www.ece.umn.edu/ arya/EE5585.htm
- Midterm - February 26th (Open Book)
- Everyone should have received LaTeX scribe template via email

Entropy (Review)

Suppose the finite set $\mathcal{X}$ is a data source and for any $x \in \mathcal{X}, p(x)$ denotes the probability distribution for x. If we know this probability distribution we can define something known as the entropy of this random variable – this is not a function of the random variable itself but its probabilities.

$$H(X) = -\sum p(x) \log_2(x) \text{ (measured in bits)}$$

Example: Suppose $\mathcal{X} = \{1,2,3\}$ and $p(1) = \frac{1}{2}$, $p(2) = \frac{1}{4}$, $p(3) = \frac{1}{4}$. Then,

$$H(X) = -\tfrac{1}{2}\log_2(\tfrac{1}{2}) - \tfrac{1}{4}\log_2(\tfrac{1}{4}) - \tfrac{1}{4}\log_2(\tfrac{1}{4}) = 2 \text{ bits}$$

Now, let us consider the entropy of the English language. A standard english keyboard contains 96 characters. To represent each of these uniquely will require

$$\lceil \log_2 96 \rceil = 7 \text{ bits/Char}$$

In order to determine the entropy of the english language we could consider a very large set of text and calculate $p(a)$, $p(b), \ldots$ and so on. Doing this we would find that the entropy of the english language is ~ 4.5 bits/Char. However, this neglects the fact that the probability of a given letter occurring is not independent of adjoining letters. For example, if we consider strings of two neighboring characters we might find $p(ab) \gg p(qu) \gg p(qn)$. If we continue this exercise by considering strings up to 8 adjoining characters, we would find the entropy to be 19 bits. However, this is for 8 characters, so if we consider our compression the entropy is now $\frac{19}{8} \sim 2.4$ bits/Char. If you continue this exercise for longer and longer character strings the entropy per character asymptotically approaches the limit of 1.3 bits/Char. Now let us consider the entropy achieved by some standard codes:

- Hoffman Coding ~ 4.7 bits/Char
- Gzip ~ 2.7 bits/Char
- Unix compress ~ 3.7 bits/Char
- Most complicated scheme to date ~ 1.89 bits/Char

Source Coding (Review)

Suppose the finite set $\mathcal{X}$ is a source where $x \in \mathcal{X}$ and $p(x)$ denotes the probability distribution for x. We assume an encoder will assign binary {0,1} valued vectors (codewords) for each symbol that is denoted by $C(x)$. Also let $\ell(x)$ denote the length of $C(x)$. Our compression is then defined by

$$\text{Avg. \# of bits per source symbol} - \sum_{x \in \mathcal{X}} p(x)\ell(x) = L(C)$$

Example: Consider the set $\mathcal{X} = \{1,2,3,4\}$ where $c(1) = 0$, $c(2) = 00$, $c(3) = 1$, $c(4) = 01$. Let us then decode the following sequence:

$$0000 \rightarrow 1111$$

Note that in this case the sequence can also be decoded as:

$$0000 \rightarrow 22$$

This brings us to our next topic.

Uniquely Decodable Codes

Before formally defining what a uniquely decodable code is we first define what is called a non-singular code. A non-singular code is a code where

$$if\ x \neq x'\ then\ C(x) \neq C(x')$$

This leads us to our definition of a uniquely decodable code, which states that an extension of C is non-singular. Thus, using the codewords from the previous example this would mean that:

$$C(22) \neq C(1111)$$

Theorem 1 *For any uniquely decodable code C for the source X with source symbols following the distribution of the random variable X,*

$$L(\mathcal{C}) \geq \text{H(X)}$$

This theorem will be proved in Lecture 3 with the use of the famous Kraft inequality which states that

Kraft Inequality *For any u.d. (uniquely decodable) code for source X ,*

$$\sum_{x \in \mathcal{X}} 2^{-\ell(x)} \leq 1$$

Example: Suppose $\mathcal{X} = \{1,2,3,4\}$ where $c(1) = 00$, $c(2) = 01$, $c(3) = 11$, $c(4) = 110$. Now let us decode the bit stream 00011100011 using these codewords

$$00011100011 \rightarrow 12413 \text{ or } 12312 + \text{remaining bit}$$

Thus, we see that this code is NOT uniquely decodable. However, let us consider modifying our codewords to $c(1) = 00$, $c(2) = 01$, $c(3) = 11$, $c(4) = 110$. Now suppose we are given the bitstream

$$00111101000110 \rightarrow 134214$$

Now we see that the code is uniquely decodable. However, each codeword cannot be decoded without looking at adjoining codewords, so in this respect the code is not instantaneous.

Instantaneous Code (Prefix Code)

An instantaneous code, or prefix code, is a code in which any codeword is not a prefix of any other codeword.

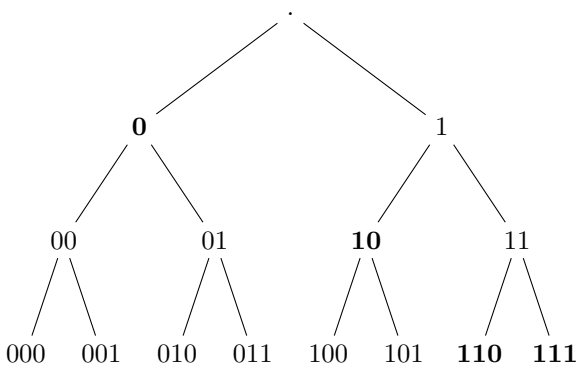
Example: $c(1) = 0, c(2) = 10, c(3) = 110, c(4) = 111$ is an instantaneous code.

Every instantaneous code is uniquely decodable. Using the codewords from the previous example, we see that Theorem 1 is withheld.

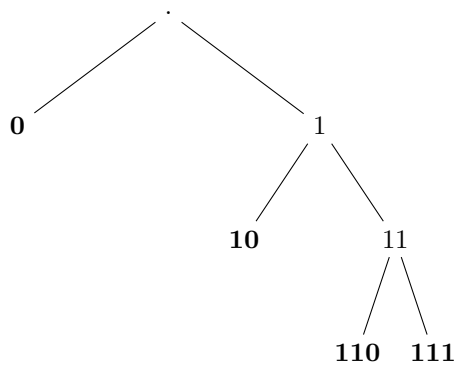
Example: Suppose $\mathcal{X} = \{1,2,3,4\}$ and $p(1) = 0.25, p(2) = 0.25, p(3) = 0.25, p(4) = 0.25$.

$$L(C) = 0.25 \times 2 + 0.25 \times 2 + 0.25 \times 2 + 0.25 \times 3 = 0.25 \times 9 = 2.25 \geq H(X) = 2$$

The Kraft inequality holds for instantaneous codes just as it does for uniquely decodable codes. We prove this again in a more insightful manner making use of a binary tree



Here the codewords used in the previous example are **bolded** within the binary tree. Because all of our codewords have a length no greater than 3 and this binary tree is three layers tall, it is bound to contain all 3 bit codewords. We notice within the tree the property of an instantaneous code – no codeword is a descendant of any other within the tree. Now suppose we erase all the unnecessary leaves of the tree



This is known as a Huffman tree. Now suppose in the general case that ℓ_{max} is the max of $\ell(x)\ \forall\ x \in \mathcal{X}$. Then the number of leaves on the tree will be equal to $2^{\ell_{max}}$. Now, consider the number of descendant leaves in the original tree (before erasing unnecessary codewords) which stem from a given codeword x. For instance, 100 and 101 would stem from the codeword 10 meaning it had 2 descendant codewords.

We note that for a given codeword x, the number of descendant leaves is $2^{\ell_{max}-\ell(x)}$. Additionally, the number of descendant leaves for two separate codewords are disjoint. Now if we sum all of the descendant leaves for all source symbols we note that we can have at most $2^{\ell_{max}}$ (the total number of leaves on the tree). In other words,

$$\sum_{x\in\mathcal{X}} 2^{\ell_{max}-\ell(x)} \leq 2^{\ell_{max}}$$

By dividing both sides of this equation by $2^{\ell_{max}}$ we see that this implies the Kraft Inequality

$$\sum_{x\in\mathcal{X}} 2^{-\ell(x)} \leq 1$$

Now, suppose that $\ell_1, \ell_2, \ldots, \ell_m$ are integers such that the Kraft inequality is satisfied i.e.,

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$$

Then $\exists$ a prefix code C on m source symbols such that the codeword lengths are $\ell_1, \ell_2, \ldots, \ell_m$. We can assume *w.l.o.g.* (without loss of generality) $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_m$. This code can be constructed by drawing a binary tree that is m levels tall, selecting a codeword from the i^{th} level, erasing all of the descendants of that codeword from the tree ($2^{\ell_{max}-\ell_i}$), and then continuing to the $i^{th}+1$ level until the m^{th} level is reached.

Optimal Code

An optimal code would minimize the length of the codewords

$$L(C) = \sum_{x\in\mathcal{X}} p(x)\ell(x)$$

While still remaining uniquely decodable, meaning it is subject to the restraint of Kraft's inequality

$$\sum_{x\in\mathcal{X}} 2^{-\ell(x)} \leq 1$$

Note that $\ell(c)$ are all integers. This becomes an optimization problem. Looking at Theorem 1 more closely and rewriting it in terms of $\ell(x)$ we note

$$\sum_{x\in\mathcal{X}} p(x)\ell(x) \geq \sum_{x\in\mathcal{X}} p(x)\log_2 \tfrac{1}{p(x)}$$

From this it becomes apparent that for an optimal code we are intuitively looking for lengths $\ell(x) \sim \log_2 \frac{1}{p(x)}$, but we don't know if these lengths exist. Now suppose $|\mathcal{X}| = \text{m}$ then say we have $\ell_1, \ell_2, \ldots, \ell_m$. Then we need to

$$\min \sum_{i=1}^{m} p_i\ell_i$$

$$\text{s.t (subject to)} \sum_{i=1}^{m} 2^{-\ell_i} \leq 1$$

We do a relaxation on this problem by assuming ℓ_i is real. It then becomes evident that if ℓ_i is real the optimal point will occur when $\sum_{i=1}^{m} 2^{-\ell_i} = 1$. We utilize the method of Lagrange's multiplier to optimize this problem,

$$J = \sum_{i=1}^{m} p_i\ell_i + \lambda(\sum_{i=1}^{m} 2^{-\ell_i} - 1)$$

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda 2^{-\ell_i}\ln 2 = 0$$

Note that above we have made use of the fact that $2^{-\ell_i} = e^{-\ell_i \ln 2}$. With this partial derivative set to zero we find the optimal points

$$2^{-\ell_i} = \frac{p_i}{\lambda \ln 2}$$

Now we subject this to the restraint imposed by Kraft's inequality in the optimal case ($\sum_{i=1}^{m} 2^{-\ell_i} = 1$) in order to find the value of lambda.

$$\frac{\sum p_i}{\lambda \ln 2} = 1 \Rightarrow \lambda = \frac{1}{\ln 2}$$

It follows that $p_i = 2^{-\ell_i} \Rightarrow \ell_i = \log_2 \frac{1}{p_i}$. If these were integers, the average number of bits per symbol would be

$$\sum p_i \log_2 \frac{1}{p_i} = H(X)$$

If we could use the optimal length, meaning it was an integer, then we could achieve the optimal entropy. However, we cannot guarantee this and instead make use of the Shannon code to find the closest integer length.

Shannon Code Take $\ell(x) = \lceil \log_2 \frac{1}{p(x)} \rceil$. Then to find the average number of bits per symbol

$$\begin{aligned}
\sum_{x \in \mathcal{X}} 2^{\lceil \log_2 \frac{1}{p(x)} \rceil} &\leq \sum_{x \in \mathcal{X}} 2^{\log_2 \frac{1}{p(x)}} \\
&= \sum_{x \in \mathcal{X}} p(x) \\
&= 1
\end{aligned}$$

Therefore, the average # of bits per symbol

$$\begin{aligned}
&= \sum p(x) \lceil \log_2 \tfrac{1}{p(x)} \rceil \\
&< \sum p(x) \log_2 \tfrac{1}{p(x)+1} \\
&= H(X) + \sum p(x) \\
&= H(X) + 1
\end{aligned}$$

Shannon coding might not always be optimal. Suppose we have only two symbols $\mathcal{X} = \{1,2\}$ where $p(1) = 0.9999$, $p(2) = 0.0001$. Then, $\ell(1) = \lceil \log_2 \frac{1}{0.9999} \rceil = 1$ and $\ell(2) = \lceil \log_2 \frac{1}{0.0001} \rceil = 14$. This is clearly not optimal, since you only need one bits to represent this (not 14).

EE5585 Data Compression January 29, 2013

Lecture 3

Instructor: Arya Mazumdar *Scribe: Katie Moenkhaus*

Uniquely Decodable Codes

Recall that for a uniquely decodable code with source set $\mathcal{X}$, if $l(x)$ is the length of a codeword with $x \in \mathcal{X}$ then Kraft's Inequality is satisfied

$$\sum_{x\in\mathcal{X}} 2^{-l(x)} \leq 1 \tag{1}$$

A prefix code is a kind of uniquely decodable code in which no valid codeword is a prefix of any other codeword. These are generally easier to understand and construct.

Theorem 1 *For any uniquely decodable code C, let $L(C)$ be the average number of bits per symbol, $p(x)$ be the probability of the occurrence of symbol x, and $H(X)$ be the entropy of the source. Then*

$$L(C) = \sum_{x\in\mathcal{X}} l(x)p(x) \geq H(X) = -\sum_{x\in\mathcal{X}} p(x)\log_2(x) \tag{2}$$

Theorem 1 signifies that the entropy function puts a fundamental lower bound on the average number of bits per symbol.

Proof: Rearranging equation (2) leads to

$$\begin{aligned} L(C) - H(X) &\geq \sum_{x\in\mathcal{X}} l(x)p(x) + \sum_{x\in\mathcal{X}} p(x)\log_2(x) \\ &= -\sum_{x\in\mathcal{X}} p(x)\log_2 2^{-l(x)} + \sum_{x\in\mathcal{X}} p(x)\log_2 p(x) \\ &= -\sum_{x\in\mathcal{X}} p(x)\log_2\left(\frac{2^{-l(x)}}{\sum_{x\in\mathcal{X}} 2^{-l(x)}}\right) - \sum_{x\in\mathcal{X}} p(x)\log_2\left(\sum_{x\in\mathcal{X}} 2^{-l(x)}\right) + \sum_{x\in\mathcal{X}} p(x)\log_2(p(x)) \end{aligned}$$

Notice that

$$0 \leq \frac{2^{-l(x)}}{\sum_{x\in\mathcal{X}} 2^{-l(x)}} \equiv r(x) \leq 1 \tag{3}$$

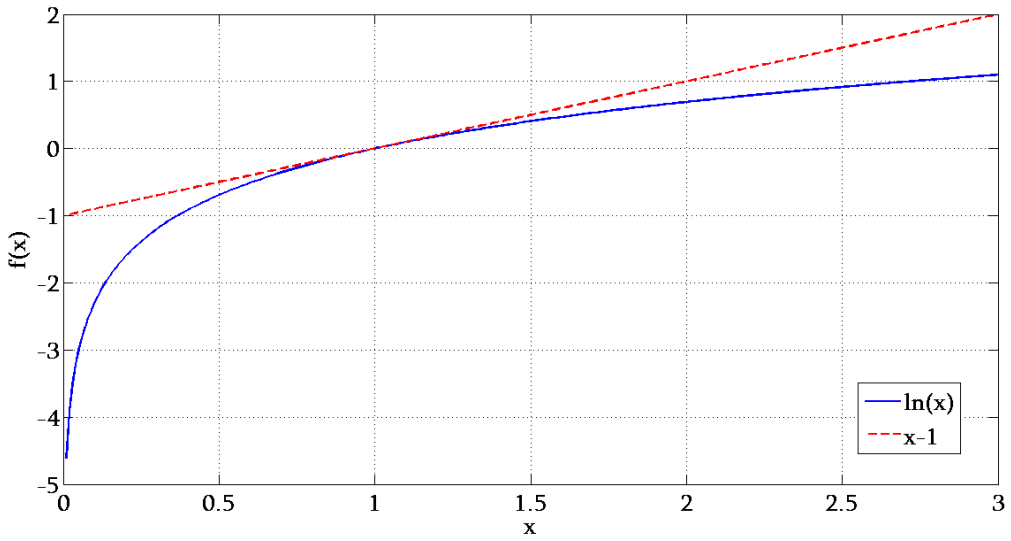
Combining the above equations leads to

$$-\sum_{x\in\mathcal{X}} p(x)\log_2\left(\frac{r(x)}{p(x)}\right) - \log_2\left(\sum_{x\in\mathcal{X}} 2^{-l(x)}\right)\sum_{x\in\mathcal{X}} p(x) \tag{4}$$

We have, $\sum_{x\in\mathcal{X}} p(x) = 1$. Additionally, change the base of the logarithm in the first term from 2 to e using the change of base formula for logs.

$$-\log_2(e)\sum_{x\in\mathcal{X}} p(x)\ln\left(\frac{r(x)}{p(x)}\right) - \log_2\left(\sum_{x\in\mathcal{X}} 2^{-l(x)}\right) \tag{5}$$

Recall the plot of $\ln(x)$:



The tangent line to the curve of $\ln(x)$ at $x = 1$ is $f(x) = x - 1$. It follows that

$$\ln(x) \leq x - 1 \tag{6}$$

Using equation (5) in conjunction with equation (6) leads to

$$-\log_2(e)\sum_{x\in\mathcal{X}} p(x)\ln(\frac{r(x)}{p(x)}) - \log_2(\sum_{x\in\mathcal{X}} 2^{-l(x)}) \geq -\log_2(e)\sum_{x\in\mathcal{X}} p(x)(1-\frac{r(x)}{p(x)}) - \log_2(\sum_{x\in\mathcal{X}} 2^{-l(x)}) \tag{7}$$

Consider the term $\sum_{x\in\mathcal{X}} p(x)(1-\frac{r(x)}{p(x)})$. Distributing, this becomes $\sum_{x\in\mathcal{X}} p(x) - r(x) = \sum_{x\in\mathcal{X}} p(x) - \sum_{x\in\mathcal{X}} r(x)$. As previously discussed, $\sum_{x\in\mathcal{X}} p(x) = 1$. Also, because of how r(x) is defined, $\sum_{x\in\mathcal{X}} r(x) = 1$. The subtraction of the two is then 0, so this term simplifies to 0. The right-hand side of equation (10) simplifies to

$$-\log_2 \sum_{x\in\mathcal{X}} 2^{-l(x)} \tag{8}$$

From Kraft's Inequality, this sum is always less than or equal to 1. Since $-\log_2(1) = 0$,

$$L(C) \geq H(X) \tag{9}$$

This holds for any uniquely decodable code.

Note: if $\log_e(x) \equiv \ln(x)$ is used, the unit that follows is nats instead of bits.

Optimal Codes

For a code to be optimal, the average length of the codewords is minimal. Recall that for Shannon Codes, $l(x)$ is given by

$$l(x) = \lceil \log_2(\frac{1}{p(x)}) \rceil \tag{10}$$

It can be shown that this code satisfies Kraft's Inequality. Also, for Shannon Codes,

$$H(X) \leq L(C) \leq H(X) + 1 \tag{11}$$

The Shannon code is not optimal in all cases.

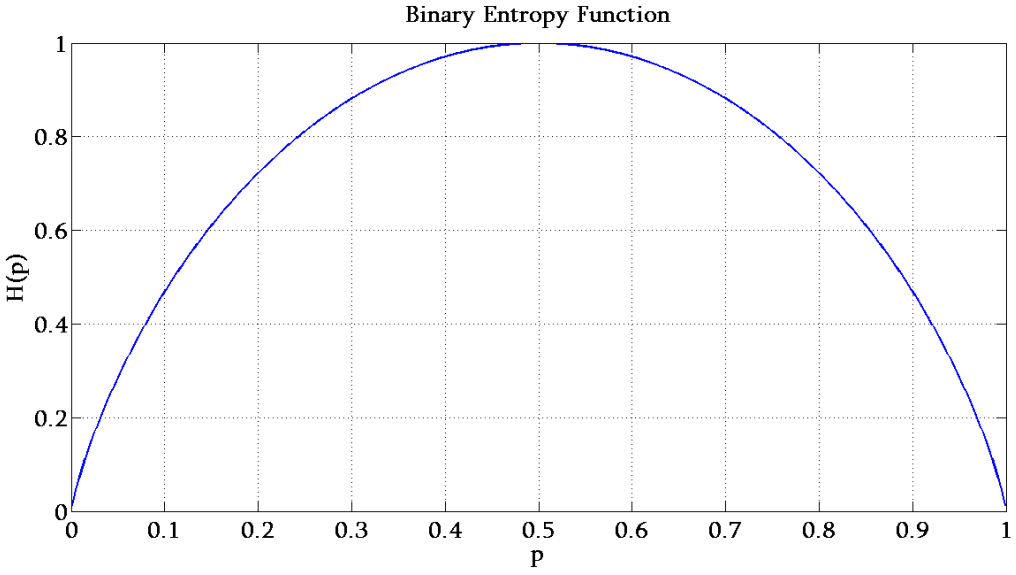
Example: Binary Entropy Function

$\mathcal{X} = \{0, 1\}$
$p(0) = p$
$p(1) = 1 - p$

The entropy of this set is

$$H(p) = -(p)\log_2(p) - (1-p)\log_2(1-p) \tag{12}$$

The graph of this function looks like



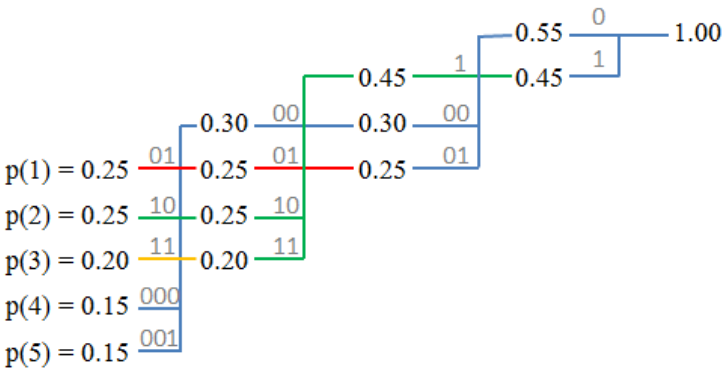
This function is symmetric about 1/2, i.e. $H(p) = H(1-p)$, and $H(p)$ achieves its maximum value at $p = 1/2$. This would represent data that contains equal numbers of zeros and ones. However, this is usually not the case for normal data, so the minimum number of bits needed to represent each symbol is actually less than one with a good compression algorithm.

Huffman Coding

Example: Suppose $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities

x	$p(x)$
1	0.25
2	0.25
3	0.20
4	0.15
5	0.15

First, arrange the probabilities in decreasing order (already done in the above table). Next, add the probability of the two least likely symbols together to make a supersymbol with probability $p(4)+p(5)$. Repeat this process until only one supersymbol is left. This will be the number 1 and is the root of the tree. Binary values are then assigned starting at the root of the tree as follows.



Thus, the Huffman code for this set is

x	$C(x)$
1	01
2	10
3	11
4	000
5	001

The average number of bits per symbol is

$$L(C) = \sum_{x \in \mathcal{X}} p(x)l(x) = 0.25 * 2 + 0.25 * 2 + 0.2 * 2 + 0.15 * 3 + 0.15 * 3 = 2.3 \tag{13}$$

The entropy for this example, $H(X)$, is 2.2855. $L(C)$ being very close to $H(X)$ suggests that the Huffman Code is at least close to optimal. The tree method of assigning codewords presented above ensures that no codeword is a prefix of any other codeword, so the Huffman Code is a prefix code.

Optimality of the Huffman Code

A code is optimal when $\sum_{x \in \mathcal{X}} p(x)l(x) = L(C)$ is minimized. Suppose $\mathcal{X} = \{1, 2, ..., m\}$ with decreasing probabilities $p(1) \geq p(2) \geq ... \geq p(m)$.

Propteries of Optimal Codes:

- $l(1) \leq l(2) \leq ... \leq l(m)$
  Suppose an optimal code (C_{opt}) has $p(j) > p(k)$ but $l(j) > l(k)$. Define a new code in which the codewords $l(j)$ and $l(k)$ are switched.
  $L(C_{opt}) - L(C) = p(j)l(j) + p(k)l(k) - p(j)l(k) - p(k)l(j) = (l(j) - l(k))(p(j) - p(k)) > 0$
  This implies that C_{opt} cannot be the optimal code, and the optimal code has to have the mentioned ordering.

- $l(m-1) = l(m)$ i.e. the lengths of the longest two codewords are the same.
  Suppose, in an optimal code C, $C(m-1)$ and $C(m)$ have different lengths. Because of the prefix-free property, the last bit of the longer codeword could just be eliminated. The code will still be prefix free, and the resulting code will be better than the optimal code, a contradiction.
- $C(m-1)$ and $C(m)$, which are the codewords for the least likely symbols, differ only by the last bit. If they differ more than one bit, the last bit can be omitted from the codeword.

Codes that exhibit all of these properties are called canonical codes. Note: optimal codes are not necessarily unique; multiple optimal codes may exist for the same source.

In some situations, the Shannon code can be optimal. This happens when $\lceil \log_2(\frac{1}{p(x)}) \rceil = \log_2(\frac{1}{p(x)})$.

Example: $p(a) = 1/2$, $p(b) = 1/4$, $p(c) = 1/4$. For this probability distribution, the Shannon code is optimal because the probabilities are of the form $p(x) = 2^{-t}, t \in \mathbb{Z}$.

Theorem 2 *The Huffman code is optimal. For any other uniquely decodable code C and Huffman code C_H,*

$$L(C) \geq L(C_H) \tag{14}$$

Proof: For a source with two symbols, Huffman coding is optimal because each symbol is assigned one bit. An optimal canonical code, C^m_{opt}, for m symbols has codeword lengths $l(1) \leq l(2) \leq ... \leq l(m)$ and probabilities $p(1) \geq p(2) \geq ... \geq p(m)$. A new prefix code, C^{m-1}, is made from C^m_{opt} on $m-1$ symbols. The new set of probabilities is formed using $p(1), p(2), ..., p(m-2), p(m-1) + p(m)$. This is called Huffman reduction, in which the two least likely symbols are combined into a super symbol. The codewords for C^m_{opt} and C^{m-1} are the same until $m-2$, i.e. $C^m_{opt}(k) = C^{m-1}(k)$ for $k = \{1, 2, ..., m-2\}$. Then $C^{m-1}(m-1)$ is the codeword $C^m_{opt}(m)$ without the last bit. C^{m-1} is a prefix-free code, and

$$L(C^{m-1}) = L(C^m_{opt}) - p(m) - p(m-1) \tag{15}$$

Now let C^{m-1}_{opt} be an optimal code for the probability distribution $\{p(1), p(2), ..., p(m-2), p(m-1) + p(m)\}$. A new code, C^m, is formed from the probability distribution $\{p(1), p(2), ..., p(m-2), p(m-1), p(m)\}$. $C^m(m-1) = [C^{m-1}_{opt}(m-1), 0]$ and $C^m(m) = [C^{m-1}_{opt}(m-1), 1]$, where [ ] denotes append.

$$L(C^m) = L(C^{m-1}_{opt}) + p(m-1) + p(m) \tag{16}$$

Adding equations (18) and (19) leads to

$$L(C^{m-1}) + L(C^m) = L(C^m_{opt}) + L(C^{m-1}_{opt}) \tag{17}$$

This means

$$[L(C^m) - L(C^m_{opt})] + [L(C^{m-1}) - L(C^{m-1}_{opt})] = 0 \tag{18}$$

Each term in the above equation is nonnegative. They must individually sum to zero, meaning that $L(C^m) = L(C^m_{opt})$. Thus, C^m is an optimal code. The above procedure outlines one level of building a Huffman code. If the code is optimal at one level, the procedure can be extrapolated to an arbitrarily long Huffman code. Therefore, by induction, the Huffman code is optimal.

EE5585 Data Compression January 31, 2013

Lecture 4

Instructor: Arya Mazumdar *Scribe: John Havlik*

Huffman Code (Review)

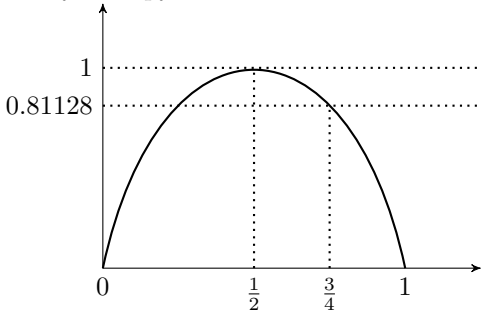
Huffman coding has been shown to be optimal for sources with a discrete alphabet. For any source, given a distribution, a Huffman code is the best code for compressing the source. A Huffman code is formed using a Huffman tree.

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)

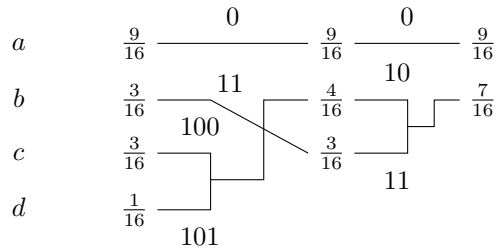
Suppose on same alphebet X we have two distributions $p = p(x)$ $q = q(x)$

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.

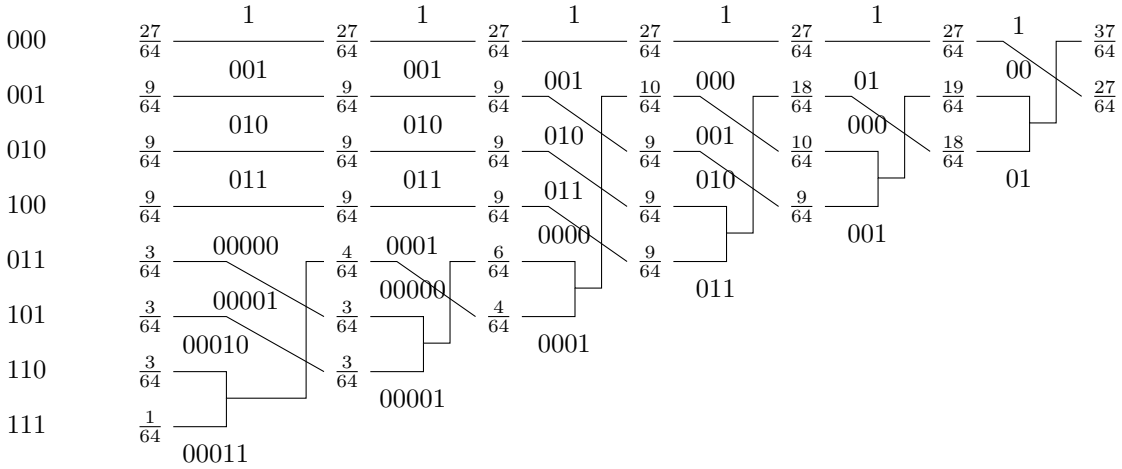
Example: Suppose we have $Pr(00) = \frac{9}{16}$ $Pr(01) = \frac{3}{16}$ $Pr(10) = \frac{3}{16}$ and $Pr(11) = \frac{1}{16}$ with $a = 00$ $b = 01$ $c = 10$ and $d = 11$



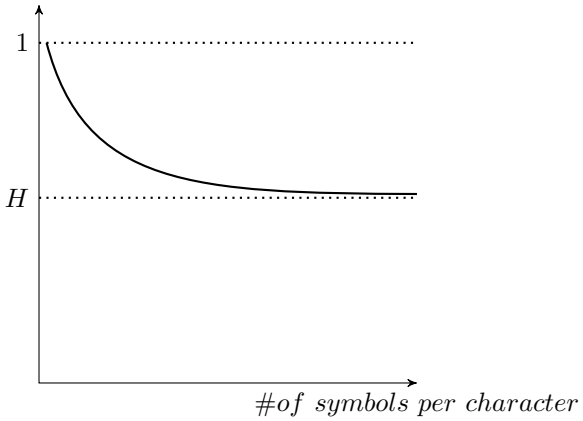
Average length of a codeword with this scheme:

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

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:

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

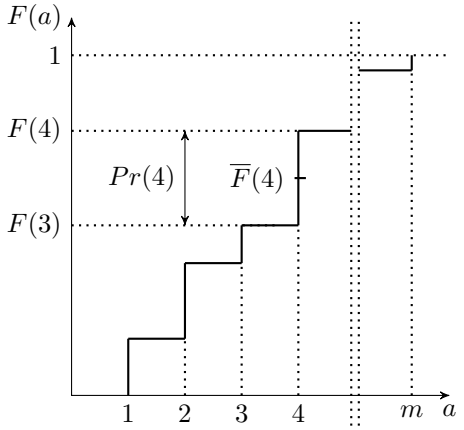
$1 * \frac{1}{2}$, $0 * \frac{1}{4}$, etc. are atoms of this representation.

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.

EE5585 Data Compression February 5, 2013

Lecture 5

Instructor: Arya Mazumdar *Scribe: Kriti Bhargava*

Arithmetic Coding

To understand arithmetic coding better, there are some points that we need to keep in mind :

1. Shannon Fano Elias Coding: The basic idea of using a modified CDF rather than sorted probabilities to find the code word is extended in arithmetic coding.
2. X is a random variable $\in \mathbf{R}$
   $F_X(X)$ is uniform in the interval $[0, 1)$
3. For any $X \sim U[0, 1)$, binary expansion of X is a sequence of independent Bernoulli($\frac{1}{2}$) random variables.
4. The binary entropy function $h(p) = 1$ for p= $\frac{1}{2}$, that is, a sequence of independent Bernoulli($\frac{1}{2}$) random variables cannot be compressed any further. $X \to F_X(X)$ gives a uniform random variable and thus, making it incompressible.

Using an example : 0011011 with $p(0) = \frac{3}{4} = 1 - p \equiv q$; $p(1) = \frac{1}{4} = p$

Consider a continuous real random variable,

$$X = 0.x_1x_2x_3\ldots$$

It may not be independent, may be biased.

$$F_X(x) = P(X \leq x)$$

$$F_X(0011011) = P(0.X_1X_2X_3\ldots \leq 0.0011011)$$

$$\begin{aligned} &= P(X_1 < 0) + P(X_1 = 0, X_2 < 0) \\ &\quad + P(X_1 = 0, X_2 = 0, X_3 < 1) + \cdots \\ &\quad + P(X_1 = 0, X_2 = 0, X_3 = 1, X_4 = 1, X_5 = 0, X_6 = 1, X_7 < 1) \end{aligned}$$

$$= q.0 + q^2.q + \cdots + q^3.p^3.q$$

So, if we have probabilities of $k-1$ bits, it can be extended further.

$$\textstyle\sum_k P(x_1^{k-1}).x_k * q \text{ for } x_1^n = x_1x_2\ldots x_n$$

For finite length : $0.x_1x_2\ldots x_n$

$$[0.x_1x_2\ldots x_n00\ldots 0; x_1x_2\ldots x_n111\ldots]$$

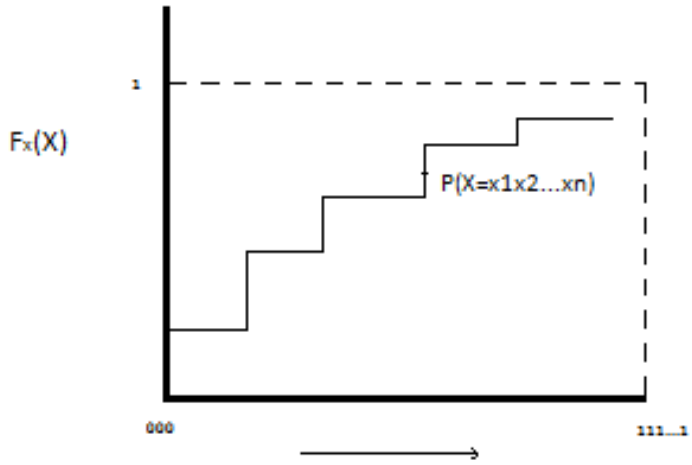
Size of this interval = $\frac{1}{2^n}$

$$F_X[\ ] = [F_X(0.x_1x_2\ldots x_n00\ldots 0), F_X(0.x_1x_2\ldots x_n111\ldots)]$$

All real numbers whose first n-bits are fixed, not a singleton set.

$$= P[X_1 = x_1, X_2 = x_2\ldots X_n = x_n]$$

Natural lexicographic order - 00,01,10,11



A seq of n random variable in SFE

$F_X^{-1}(u)$ = another real number whose first n bits are $x_1x_2\ldots x_n$

Length (for compression) $= \lceil \log \frac{1}{p(x_1\ldots x_n)} \rceil + 1$

$$< H(x_1\ldots x_n) + 2$$

If each bit is independent, then for each bit

$$= \frac{1}{n}(H(x_1\ldots x_n) + 2)$$

For longer blocks, that is, as $n \to \infty$

$$\frac{H(x_1\ldots x_n)}{n} \to H(X); \frac{2}{n} \to 0$$

Application of arithmetic coding is mostly in fax machines and a few other JPEG standards.

Statistics (probability; the number of ones etc.) are not always available. If we know that source generates a sequence with a fixed number of ones but that number is unknown, a penalty is to be paid of sending the information of the whole sequence as property of the system is not known.
Thus, for a sequence like 001100110110, in general, information sent is:

$$\log_2 n + \log_2 \binom{n}{k}$$

Since $\log_2 \binom{n}{k} = \frac{c}{\sqrt{n}} 2^{nh(\frac{k}{n})}$

where $h(x) = -x \log x - (1-x)\log(1-x)$

$$= \frac{1}{n}[\log_2 n + nh(\tfrac{k}{n}) - \tfrac{1}{2}\log_2 n + \log_2 c]$$

$$= h(\tfrac{k}{n}) + \tfrac{1}{2n}\log_2 n + \tfrac{\log_2 c}{n}$$

where $\frac{\log_2 c}{n} \to 0$ and $\frac{1}{2n}\log_2 n$ is the penalty for not knowing enough information

$$0010011000\ldots$$
$$\chi = \{1, 2, 3, \ldots, m\}$$

We need an estimate of $p(0) = 1 - p$ and $p(1) = p$. For this, we need to go through the whole sequence and from the laws of large numbers, calculate the estimate of p and then move on to applying Huffman or Arithmetic coding. But this is not sequential.

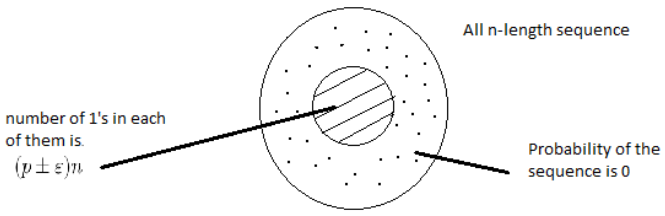
P(number of ones in the n length sequence $= w$) $= \binom{n}{w} p^w (1-p)^{n-w}$
{ Binomial distribution, therefore $\sum_{w=0}^{n} \binom{n}{w} p^w (1-p)^{n-w} = 1$}

$$\begin{aligned}
\binom{n}{w} p^w (1-p)^{n-w} &= \frac{c}{\sqrt{n}} 2^{nh(\frac{w}{n})} p^w (1-p)^{n-w} \\
&= \frac{c}{\sqrt{n}} 2^{n(-\frac{w}{n}\log\frac{w}{n} - (1-\frac{w}{n})\log(1-\frac{w}{n}))} 2^{w\log p} 2^{(n-w)\log(1-p)} \\
&= \frac{c}{\sqrt{n}} 2^{-n(\frac{w}{n}\log\frac{\frac{w}{n}}{p} + (1-\frac{w}{n})\log\frac{1-\frac{w}{n}}{1-p})} \\
&= \frac{c}{\sqrt{n}} 2^{-nD(\frac{w}{n}\|p)}
\end{aligned}$$

Suppose $\frac{w}{n} = \omega$.

$D(\omega \parallel p) \geq 0$
with eqaulity only if $\omega = p$
$\mid \omega - p \mid \geq \varepsilon > 0$
If $D(\omega \parallel p) =$ positive ; $2^{-n(+ve)} \to 0$
It is non-zero when ω is very close to p, i.e, $w \simeq pn$

Typical Set



Size of the typical set for small arbitrary ε say is $A_{n,p,\varepsilon} = \sum_{i=(p-\varepsilon)n}^{(p+\varepsilon)n} \binom{n}{i}$. Clearly

$$\binom{n}{pn} \leq A_{n,p,\varepsilon} \leq (2\varepsilon n + 1)\binom{n}{(p+\epsilon)n}$$

For very small ϵ, $A_{n,p,\varepsilon} \sim 2^{nh(p)}$
A file cannot be compressed beyond $h(p)$, compress up to $nh(p)$ bits

Rate of compression $\sim h(p)$ {Even for large alphabets}

For unknown p, the rate = $h(p)$+ error
As the block length is increased and made larger, there exists coding scheme for which error$\sim$ 0. One such coding scheme is Lempel Ziv coding scheme which is described next.

Lempel Ziv Coding

There are two kinds of Lempel Ziv Coding :

1. Sliding Window - Also known as LZ77.
2. Tree structured - First appeared in a paper in 1978. Also known as LZ78

Tree Structured Coding

Tree Structured is a dictionary based algorithm where a table is used for reference to determine the coding. In case of a tree structured algorithm, previous bits are referred to find out the sub sequences as we traverse along the whole sequence. This coding scheme is said to compress the sequence to the entropy of the sequence for large block lengths.

Take the following sequence - 000110110111010101011001011

For Tree Structured algorithm

- Parse the sequence : Traverse the whole sequence and parse the sequence into shorter sub sequences(phrases) where each sub sequence has not been visited earlier.

  Numbering(Marker) :1 2 3 4 5 6 7 8 9 10 11
  Parsed Sequence :0|00|1|10|11|01|110|101|011|001|011

- Encoding: For each sub sequence formed, encoding is done. A simple numbering system is used to mark the parsed string and the code is defined using marker,last bit in the sub sequence. This marker is called tuple. The final code word is made of the binary representation of the tuple and the last bit.
  For the above example, encoding is as follows :

  $(0,0)$ $(1,0)$ $(0,1)$ $(3,0)$ $(3,1)$ $(1,1)$ $(5,0)$ $(4,1)$ $(6,0)$ $(2,1)\ldots$

$c(n)$ = number of passed sub sequences

For each sub sequence -
Number of bits required $= \log c(n) + 1$
Total $= c(n)(\log c(n) + 1)$

Rate of compression $= \frac{c(n)(\log c(n)+1)}{n}$

EE5585 Data Compression February 7, 2013

Lecture 6

Instructor: Arya Mazumdar *Scribe: Joshua Krist*

Main Topic: Proof that Lampel-Ziv is Optimal

Lampel-Ziv background

Lampel-Ziv is a universal code, and as such does not depend on foreknowledge of the occurrence probablities of the symbols being sent. The Lampel-Ziv code has two main types- a "sliding window" and a "tree structure" algorithm. The proof will focus on the "tree structure" algorithm.

To encode a message using the Lampel-Ziv algorithm you must:

1) Start at the beginning and break the message into unique chunks (called phrases) truncating each chunk whenever a unique phrase is found.

For Example:
The Message: 01000111010010011100110101 00... would be broken into the following phrases
Phrases: 0, 1, 00, 01, 11, 010, 0100, 111, 001, 10, 101, 0...

2) Each phrase is then encoded with a prefix followed by a final bit. The prefix is the index of a previous phrase that contains the first part of the current phrase, with the exception of the last bit, which is contained in the last bit part of the encoded phrase. A zero is used to indicate that there is no previous phrase.

Consider the above example: 0 1 00 01 11 010 0100 111 001 10 101 0...
The binary encoding would be: (0,0) (0,1) (1,0) (1,1) (2,1) (4,0) (6,0) (5,1) (3,1) (2,0) (10,1) ...

Some Notes:
Let $c(n)$ be the number of distinct phrases
The size of each encoded phrase would then be: $\log c(n) + 1$
To Decode: Just take the encoded message and evaluate in blocks of $\log c(n) + 1$, where the first $\log c(n)$ bits is the reference to a previous phrase and the last bit is the last bit of the decoded phrase.

Proof: Lampel-Ziv is Optimal

Notes on proof:
Length of phrase is $\log c(n) + 1$
Length of compressed file $c(n)(\log c(n) + 1)$
Rate of compression $\frac{c(n)(\log\ c(n)+1)}{n}$
For this proof we will assume that all the elements in the message are independently identically distributed (i.i.d.)
Note: there is a broader proof that deals with a stationary ergodic case, but that will not be dealt with here.

Let $x_1, x_2, ..., x_n$ be i.i.d.

Claim: $\frac{c(n)(\log c(n)+1}{n}$ converges to H(X) as n $\to \infty$

Lemma 1: $-\frac{1}{n}\log\ p(x_1, x_2, ..., x_n)$ converges to H(X) in probablity

This shows the Asymptotic Equipartition Property (AEP)

Let $x_1, x_2, ..., x_n$ be i.i.d.

Claim: $-\frac{1}{n}\log\ p(x_1, x_2, ..., x_n)$ converges to H(X) in probability

To show this convereance it needs to be shown that $|-\frac{1}{n}\log_2\ p(x_1, x_2, ..., x_n) - H(x)| < \epsilon, \forall\epsilon > 0$

Because $x_1, x_2, ..., x_n$ are i.i.d. it can be split thus $-\frac{1}{n}\log_2\ p(x_1, x_2, ..., x_n) = -\frac{1}{n}\sum_{i=1}^{n}\log_2\ p(x_i)$

Let $y_i = \log_2\ p(x_i)$ note this is also a random variable and i.i.d. This leads to

$-\frac{1}{n}\log_2\ p(x_1, x_2, ..., x_n) = -\frac{1}{n}\sum_{i=1}^{n}\log_2\ p(x_i) = -\frac{1}{n}\sum_{i=1}^{n} y_i$

Using the Weak Law of Large Numbers we can arrive at

$-\frac{1}{n}\log_2\ p(x_1, x_2, ..., x_n) = -\frac{1}{n}\sum_{i=1}^{n} y_i = -EY$ in probability

$= E\log_2 p(X)$

$= -\sum_{x\in X} p(X = x)\ \log_2\ \text{p(X= x)} = \text{H(X)}$

Lemma 2: $c(n) \leq \frac{n}{(1-\epsilon_n)\log(n)}$ where $\epsilon_n \to 0$ as $n \to \infty$

Let n be the length of the binary sequence and c(n) be the number of distinct phrases

The sum of the length of distinct phrases that are less then or equal to k is $\sum_{j=1}^{k} j2^j$

This can be summed up like a geometric series where $\sum_{i=1}^{k} x^i = \frac{x^{k+1}-x}{x-1}$

$\sum_{j=1}^{k} jx^j = \frac{(k+1)x^k-1}{x-1} - x\frac{x^{k+1}-x}{(x-1)^2}$

placing x = 2 into the equation gives $\sum_{j=1}^{k} j2^j = 2[(k+1)2^k - 1 - 2^{n+1} + 2]$

$= 2[(k-1)2^k + 1]$

$= (k-1)2^{k+1} + 2$

$\equiv n_k$

now suppose n = n_k

then the maximum number of distinct phrases c(n) = $\sum_{j=1}^{k} 2^j = 2^{k+1} - 2 \leq 2^{k+1} \leq \frac{n_k}{k-1}$

for some k $n_k \leq n \leq n_{k+1}$

So $c(n) \leq c(n_k) + \frac{n-n_k}{k+1} \leq \frac{n_k}{k-1} + \frac{n-n_k}{k+1} \leq \frac{n}{k-1}$

$n_k \leq n = 2^{k+1} \leq (k-1)2^{k+1} + 2$

$k \leq \log(n) - 1$

$n \leq n_{k+1} = k2^{k+2} + 2 \leq \log(n-1)2^{k+2} + 2$

$2^{k+2} \geq \frac{n-2}{\log(n)-1}$

$k - 1 \geq \log(\frac{n-2}{\log(n)-1}) - 3$

$c(n) \leq \frac{n}{\log \frac{n-2}{\log(n)-1} - 3}$

the denominator expanded out is: $\log(n-2) - \log(\log\ n - 1) - 3$

$= (\log\ n)[\frac{\log(n-2)}{\log(n)} - \frac{\log(\log(n-1))-3}{\log(n)}]$

$\geq (\log\ n)[\frac{\log(n)-1)}{\log(n)} - \frac{\log(\log(n-1))-3}{\log(n)}]$

$= (\log\ n)(1 - \frac{\log(\log(n-1)-4}{\log(n)}) = (\log\ n)(1 - \epsilon_n)$

Note: $\epsilon \to 0$ as $n \to \infty$

Therefore $c(n) \leq \frac{n}{(1-\epsilon_n)\log(n)}$

Statement of Lemma 3

Lemma 3 is not proven here, and will be proved in the next set of notes. However, it is stated here for reference.

Given that z is a random varaiable that takes a non-negative integer value, with an expected value of $E(z) = \mu$

Then $H(z) \leq H(g)$ where $H(g)$ is a geometric random variable with an expected value of $E(g) = \mu$

Main Proof: $\frac{c(n)(\log c(n)+1}{n}$ converges to H(x) as n $\to \infty$

Starting with $-\frac{1}{n} \log\ p(x_1, x_2, ..., x_n)$ and $x_1, x_2, ..., x_n$ is i.i.d.

Let them be parsed as described by the Lampel-Ziv method above Let the distinct phrases be called $S_1, S_2, ..., S_{c(n)}$

So that $-\frac{1}{n} \log\ p(x_1, x_2, ..., x_n) = -\frac{1}{n} \log\ p(S_1, S_2, ..., S_{c(n)})$

$= -\frac{1}{n} \log \sum_{i=1}^{c(n)} p(S_i)$

$= -\frac{1}{n} \sum_{i=1}^{c(n)} \log\ p(S_i)$

Now by clumping phrases of the same length we can write

$= -\frac{1}{n} \sum_{l=1}^{l_{max}} \sum_{*} \log\ p(S)$ where $\sum_{*}$ is the summation of phrases that are of length l

$= -\frac{1}{n} \sum_{l=1}^{l_{max}} c_l \sum_{*} \frac{1}{c_l} \log\ p(S)$

where, c_l is the number of phrases of length l. By using Jensen's Inequality and the fact that the function is concave we can state that

$\geq -\frac{1}{n} \sum_{l=1}^{l_{max}} c_l \log \sum_{*} \frac{1}{c_l} p(S)$

The rest of the proof will be finished in the next set of notes.

EE5585 Data Compression February 8, 2013

Lecture 7

Instructor: Arya Mazumdar *Scribe: Seth Hays*

Lemmas

Lemma 1

AEP: given the sequence: $x_1x_2x_3...x_n$; an independant identically distributed (iid) random variable, then:

$\frac{1}{n}\log p(x_1x_2x_3...x_n) \to \mathbf{H}(X)$ in probability (proved in Lecture 6)

Lemma 2

Given a binary sequence of length n parsed so that all phrases are distinct. The number of distinct phrases, $c(n)$, is:
$c(n) \leq \frac{n}{(1-\varepsilon_n)\log n}$
where $\varepsilon_n \to 0$ as $n \to \infty$

Lemma 3

X is a random variable that takes non-negative integer values and $\mathbf{E}(X) = \mu$ then $\mathbf{H}(X) \leq \mathbf{H}(Z)$, where Z is a geometric random variable and

$Pr(Z = k) = (1 - \frac{1}{\mu+1})^k \frac{1}{\mu+1}$; $k = 0, 1, 2, ...$

Verify that:

$\sum_{k=1}^{\infty} Pr(Z = k) = 1$ and $\mathbf{E}(z) = \mu$
$\mathbf{H}(Z) = (\mu + 1)\log\mu + 1 - \mu\log\mu$
(Proved later)

Prove L.Z. Code is Optimal

Theorem: L.Z. Code is optimal:
$\frac{c(n)[\log c(n)+1]}{n} \to \mathbf{H}(X)$

True for any file of length n that is produced from the source alphabet, $\mathcal{X}$ and random variable, X:
$X_1, X_2, ..., X_n$; X_i iid with X

Proof of the theorem:
Start with Lemma 1:

$\frac{1}{n}\log p(X_1X_2X_3...X_n)$

take $p(X_1X_2X_3...X_n)$ with parsed phrases of $S_1, S_2, ...S_{c(n)}$

therefore $\log[p(S_1, S_2, ...S_{c(n)})]$

$= \sum_{i=1}^{c(n)} \log p[S_i] = \sum_{l=1}^{l_{max}} [\sum_{l(S)=l} \log p(S)]$

$= \sum_{l=1}^{l_{max}} c_l \sum_{l(S)=l} \frac{1}{c_l} \log[p(S)]$
where c_l = the number of phrases of length l

$\leq \sum_{l=1}^{l_{max}} c_l \log \sum_{l(S)=l} \frac{1}{c_l}[p(S)] \rightarrow$ Jensen's Inequality

$\leq \sum_{l=1}^{l_{max}} c_l \log \frac{1}{c_l}$

$= c(n) \sum_{l=1}^{l_{max}} \frac{c_l}{c(n)} \log \frac{c(n)}{c_l} - \sum_{l=1}^{l_{max}} c_l \log[c(n)]$
take $\sum_{l=1}^{l_{max}} c_l \log[c(n)] = \log c(n) \sum_{l=1}^{l_{max}} c_l = c(n) \log c(n)$

$= c(n) \sum_{l=1}^{l_{max}} \frac{c_l}{c(n)} \log \frac{c(n)}{c_l} - c(n) \log c(n)$

$= c(n) \sum_{l=1}^{l_{max}} \pi_l \log \frac{1}{\pi_l} - c(n) \log c(n)$
where $\pi_l = \frac{c(n)}{c_l}$

$= c(n)\mathbf{H}(\Pi) - c(n) \log c(n)$

Using $\sum_{l=1}^{l_{max}} l\pi_l = \frac{\sum lc_l}{c(n)} = \frac{n}{c(n)} = \mu$:
$c(n)\mathbf{H}(\Pi) - c(n) \log c(n) \leq c(n)((\frac{n}{c(n)} + 1) \log(\frac{n}{c(n)} + 1) - \frac{n}{c(n)} \log \frac{n}{c(n)}) - c(n) \log c(n)$

Going back to the beginning:
$\frac{1}{n} \log p(x_1x_2x_3...x_n) \geq \frac{c(n) \log c(n)}{n}$

$\frac{-c(n)}{n}[(\frac{n}{c(n)} + 1) \log(\frac{n}{c(n)} + 1) - \frac{n}{c(n)} \log \frac{n}{c(n)}]$

$= (1 + \frac{c(n)}{n}) \log \frac{n}{c(n)} - \log \frac{n}{c(n)}$

$= \log(1 + \frac{c(n)}{n} + \frac{c(n)}{n} \log \frac{n}{c(n)} + 1$

$\lim_{n\rightarrow\infty} \log(1 + \frac{c(n)}{n} + \frac{c(n)}{n} \log \frac{n}{c(n)} + 1$
take $\frac{n}{c(n)} = m$

$= \log(1) + \lim_{m\rightarrow\infty} \frac{\log m+1}{m} = 0$

So as $n \rightarrow \infty$; $\frac{c(n)[\log c(n)+1}{n} \rightarrow \mathbf{H}(X)$

Prove Lemma 3

Take $Pr(X = k) = p$ and $Pr(Z = k) = q$

$\mathbf{H}(Z) - \mathbf{H}(X) = -\sum(q \log q) + \sum(p \log p)$

$= \sum(q[k \log 1 - \frac{1}{\mu+1}] - \log \mu + 1] + \sum(p \log p)$

$= \sum(kq[\log 1 - \frac{1}{\mu+1}]) + \sum(q \log \mu + 1) + \sum(p \log p)$

$= -\sum(kp[\log 1 - \frac{1}{\mu+1}]) + \sum(p \log \mu + 1) + \sum(p \log p)$

$= \sum(p[\log (1 - \frac{1}{\mu+1})^k [\frac{1}{\mu+1}]) + \sum(p \log p)$

$= -\sum(p \log q + p \log p)$

$= \sum(p \log \frac{p}{q}) = D(X||Z) \geq 0$

Now Prove:
$\mathbf{H}(Z) = (\mu + 1) \log (\mu + 1) - \mu \log \mu$

$\mathbf{H}(Z) = -\sum_k (1 - \frac{1}{\mu+1})^k \frac{1}{\mu+1} [k \log (1 - \frac{1}{\mu+1}) - \log \mu + 1]$

$= -\sum(kq \log (1 - \frac{1}{\mu+1})) + \sum(q \log (\mu + 1))$

$= -\mu \log \frac{\mu}{\mu+1} + \log (\mu + 1)$

$= (\mu + 1) \log (\mu + 1) - \mu \log \mu$

EE5585 Data Compression	February 12, 2013
Lecture 8	
Instructor: Arya Mazumdar	*Scribe: Viola Z. Dsouza*

Review - An intuitive look at lossless compression techniques studied so far

When we look at Huffman coding, the procedure looks intuitive; we combine the least probable symbols and proceed iteratively. It is not difficult to understand why this algorithm works. But in the case of Lempel-Ziv coding, although we've been told it is optimal, it is not immedaitely clear as to why so. To get an intuitive understanding of why these compression algorithms work, we need to grasp some basic underlying concepts, which will be further elaborated in this text.

In real life, we may deal with many deterministic as well as random events. For example, a question such as,"Will the Sun rise tomorrow?", leads to a deterministic answer, namely, "Yes". Another event would be to determine the outcome of tossing a coin. If we knew all the factors that come in to play in this event, such as the wind velocity, force, torques and the likes, we would classify this event as being deterministic. But as it turns out, in real life, these are too many factors to account for, and as such, this event is random.

These concepts are directly related to our job of designing an efficient compression algorithm. We want to design an algorithm which is universal, i.e., one which does not depend on the file to be encoded. The entire file to be encoded is taken to be one random variable.

What is information?

Consider an event of tossing a *biased* coin, where heads is more likely to appear than tails. Now if we observe heads as the outcome, it does not convey much information since we already knew that it was more probable to appear. On the other hand, if a tails were to have appeared, we would've gained more information, since such an occurence is rare. This example serves to show that the information conveyed is directly related to the probability of occurence of an event.

Let us define a function $f(p)$ on a random variable which conveys the information contained in it and depends on the probability of occurnce of that random variable. This function $f(p)$ satisfies the following 3 properties :

1. $f(p) \geq 0$;
2. $f(p)$ is a decreasing function of p;
3. $f(p_1, p_2) = f(p_1) + f(p_2)$;

A known function which satisfies all the above 3 properties is the *log* function. Thus, we can define $f(p)$ as :

$$f(p) = c \log_2 \frac{1}{p}$$

Thus, any function of the form $f(p) = \log_2 \frac{1}{p}$ (ignoring the constant c) will represent the information contained in the random variable. The average information contained is given by the expectation of f, i.e., $E[f(p)]$

Now if $Pr[E_1] = p_1$ and $Pr[E_2] = p_2$, where E_1 and E_2 are two events, then the average information contained in both of these events is calculated by taking the weighted average of information contained

in individual events i.e.,

$$Average\ information = p_1 \log_2 \frac{1}{p_1} + p_2 \log_2 \frac{1}{p_2}$$

This equation is nothing but the equation for calculating entropy, $H(p)$. Thus, the information obtained from the file is the entropy. While designing compression algorithms, the entropy sets the lower limit on the average number of bits per symbol which can be achieved without loss of information

Lempel-Ziv Coding

Since Lempel-Ziv compression is designed to operate when statistics of the files to be encoded are not known beforehand, it appears as though it is a random algorithm. But it still works exceptionally well, at least for independent symbols or stationary/Ergodic random processes. Let us examine one of the Lemmas which we've used before, namely

$$C(n) \leq \frac{n}{\log_2 n}$$

Where $C(n)$ represents the distinct parsed phrases and n is the total number of bits in the file.

Now this formula is a general formula for any parsing which results in distinct phrases, even if these phrases are not Lempel-ziv specific. The phrases which we get by the Lempel-Ziv algorithm have a certain relationship between each other, i.e they are not parsed randomly. Whereas the result given by the lemma does not take into account any such relation between the parsed phrases.

Another relation for distinct phrases is :

$$H(File) \geq C(n) \log_2 C(n)$$

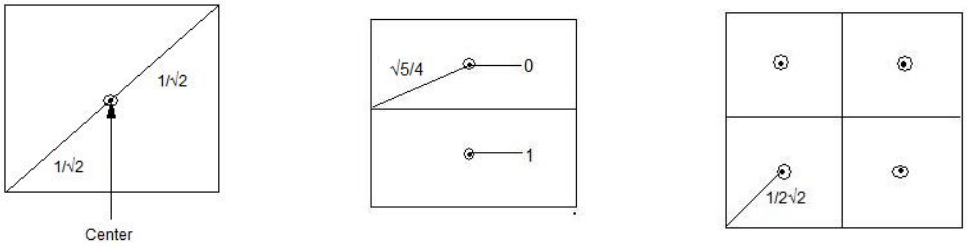
This is also a generic equation which does not take into account any specifications for Lempel-Ziv coding. Any algorithm which satisfies this equation will be optimal. Lempel-Ziv gives a way to accomplish this since the number of bits needed to be used is:

$$C(n)[\log_2(C(n)) + 1]$$

Quantization (Lossy Coding)

In lossy encoding, by giving up some information, we are able to compress the file in such a way that the average number of bits used to represent a symbol after encoding may be reduced to a value lower than the entropy $H(.)$. For example, when we implement lossy encoding for images, we discard some of the redundant pixel values. While reconstructing, these pixels get corrupted because they cannot be successfully recovered. But the human eye is not able to discern such a minute change and hence discarding such redundant information doesn't have much effect on the quality of the image. We can achieve a high rate of compression and save some bandwidth as well.

Now consider a square of area 1 $unit^2$. Each point in the square represents a file, i.e. the file is a coordinate in this square. If we were to perform quantization of any given file such that it can be represented with a certain number of bits, we would have to choose centers in this square to which we can map the files. Let us examine a few cases based on the number of bits used to represent the coded file.



(a) 0 bitl (b) 1 bit (c) 2 bits

Figure 1: Compression to different number of bits

1. Compression to 0 bits

   In this type of representation, we choose 1 point as the center to which all points in the square would be mapped to. Therefore, irrespective of the input, the code will always be same. Figure 1(a) represents such a scheme.

   The worst-case distortion is defined as the Euclidean distance between the center and the farthest point in the square. Here, this distance is given by $\frac{1}{\sqrt{2}}$.

2. Compression to 1 bit

   One way to choose centers for mapping files is as shown in figure 1(b). Here, the maximum distortion is $\frac{\sqrt{5}}{4}$. Comparing with case 1, we can see that the worst-case distortion has decreased, but we do not know if the centers chosen are optimal or not.

3. Compression to 2 bits

   A possible selection of centers is shown in figure 1(c). Here, worst-case distortion is given by $\frac{1}{2\sqrt{2}}$.

So far we have measured distortion as a function of the Euclidean distance. We will now move towards a different measure of distortion known as *Hamming distance*.

Definition : Hamming distance between two files of equal length is the number of bit positions in which the files are different.

Consider two files X_1^n and Y_1^n given by :

$$X_1^n = X_1 X_2 \dots X_n \in \{0,1\}$$

$$Y_1^n = Y_1 Y_2 \dots Y_n \in \{0,1\}$$

The *Hamming distance* between X_1^n and Y_1^n is given by

$$d(X_1^n, Y_1^n) = \text{ Number of disagreements}$$

Hamming distance will always satisfy the following 3 properties :

1. $d(x,y) \geq 0$ with equality if and only if $x = y$
2. $d(x,y) = d(y,x)$
3. $d(x,z) \leq d(x,y) + d(y,z)\ \forall y$

Covering Problem

Consider a region enclosed by the black boundary as shown in figure 2. Our first task would be to find 'Centers' for the given region. Once we have found these, we will map our given point (file) to the closest center (in terms of Hamming distance) .

Suppose we have chosen centers as shown in figure 2 (Red Dots). Let the worst case distance (distance of the farthest point in the region from the center) be d_W. We will now construct spheres in our given region centered at the red dots with radius d_W. Union of all such spheres must cover all the points in the space. Set of all points located in the sphere is given by :

$$\{y : d(x, y) \leq d_W\}$$

The *volume* of the sphere is given by the size of set of all points within the sphere.

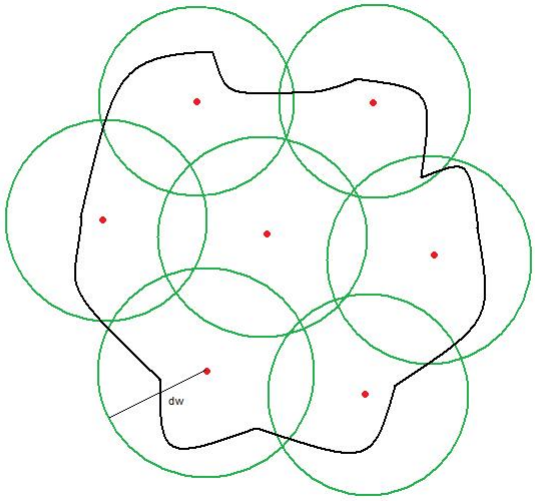


Figure 2: Example of optimal covering

If the total number of centers is given by M and the volume of each sphere is given by S, then

$$SM \geq U \tag{1}$$

where U is the volume of the entire space.

To understand the use of this inequality, we go back to the example of figure 1(c), where we used 2 bits to represent the coded output. Suppose we want to achieve a worst-case distortion $r = \frac{\sqrt{2}}{4}$, then using equation (1), we get

$$\begin{aligned} \pi r^2 \times M &\geq 1 \\ \pi \times \frac{2}{16} \times M &\geq 1 \\ M &\geq \frac{8}{\pi} \\ M &\geq 3 \end{aligned}$$

This shows that to achieve a distortion of $\frac{\sqrt{2}}{4}$, we need at least 3 points.

Example:
Consider a file source of 3 bits. To encode this particular set of files, we first map each file to a code that differs only in 1 bit, i.e. is a Hamming distance 1 away from the file. Now, we are left with only 4 values. We can now assign a 2 bit codeword to represent these.

$$\left.\begin{matrix}001\\000\end{matrix}\right\rangle 000 \text{ — } 00$$

$$\left.\begin{matrix}011\\010\end{matrix}\right\rangle 010 \text{ — } 01$$

$$\left.\begin{matrix}001\\000\end{matrix}\right\rangle 000 \text{ — } 00$$

$$\left.\begin{matrix}001\\000\end{matrix}\right\rangle 000 \text{ — } 00$$

Using the codewords that we have assigned, the worst-case distortion measured in terms of the Hamming distance is 1bit (absolute distortion). The normalized distortion is given by $\frac{1}{3}$ and the compression ratio achieved is $\frac{2}{3}$. If we were to calculate distortion based on probability, the *average distortion* would be $\frac{1}{2}$.

Now let us try to calculate the minimum number of centers we would require for the file space provided if we want a maximum distortion of 1 bit. Consider the file 000. Here, the volume of each sphere will be the number of points, which are at a distance less than or equal to 1 from 000. These points would be :

$$010$$

$$100$$

$$001$$

$$000$$

The total number of such points is 4. Therefore, the volume of the sphere is 4. Substituting our values in equation (1) we get :

$$\begin{aligned} M \times 4 &\geq 8 \\ M &\geq 2 \end{aligned}$$

Thus, we require minimum two centers to achieve a maximum distortion of 1 bit. This goes to show that the centers chosen for this example are not optimal.

Let us reexamine equation (1). By rearranging the terms we obtain :

$$M \geq \frac{U}{S} \tag{2}$$

Taking $\log_2$ on both sides we get

$$\log_2 M \geq \log_2 \frac{U}{S}$$

Here, $\log_2 M$ represents the number of bits required to represent the code. In practice, $\log_2 \frac{U}{S}$ is a good estimate for the lower bound for very large files.

Volume of sphere in Hamming Space

We have previously defined the volume of a sphere of radius r to be the number of points in space which are at a distance less than or equal to r from the center. Irrespective of the center location, the number of points within a distance r from the center are always going to be the same. Suppose we have an all zero file of n bits.

$$0000000\ldots0$$

Number of points at a distance 1 from this file $= n$
Number of points at a distance 2 from this file $= \binom{n}{2}$
Number of points at a distance 3 from this file $= \binom{n}{3}$
$\vdots$
Number of points at a distance n from this file $= \binom{n}{n}$
Thus, the volume of a sphere with radius r and center as the all zero n-bit file is:

$$\text{Volume } = \sum_{i=0}^{r} \binom{n}{i}$$

Now, the volume of the entire space, i.e U is given by:

$$U = 2^n$$

To achieve a worst case distortion of r, the number of centers required calculated using equation (2) is :

$$M \geq \frac{2^n}{\sum_{i=0}^{r} \binom{n}{i}}$$

Taking $\log_2$ on both sides, we obtain the number of bits required for representing M centers, given by:

$$\log_2 M \geq n - \log_2 \sum_{i=0}^{r} \binom{n}{i}$$

Thus the rate of compression, R is given by :

$$R \geq 1 - \frac{1}{n} \log_2 \sum_{i=0}^{r} \binom{n}{i} \tag{3}$$

Now as compression code designers, we would like to ensure a high rate of compression alongwith minimum distortion, i.e we want to decrease both R and r. But looking at equation (3), it is quite clear that both R and r cannot decrease at the same time. To observe this relationship clearly, lets take a look at the plot of compression rate v/s the worst case distortion distance in figure 3. To ensure that equation (3) holds, we want to operate in the shaded region shown in figure 3.

Let us now move on from absolute distortion measures to a normalized one. The normalized distortion D given by :

$$D = \frac{r}{n}$$

i.e.,

$$r = Dn$$

where $0 \leq D \leq 1$

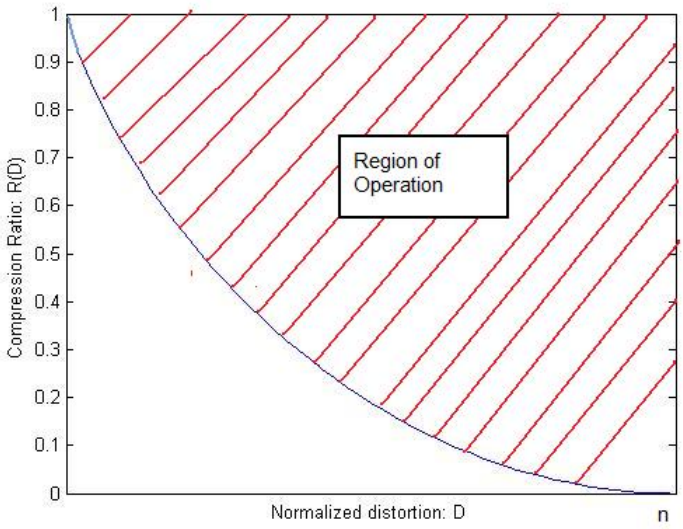


Figure 3: Plot of number of bits(n) v/s compression ratio(R)

Let $R(D)$ represent the optimal compression rate given D is the normalized distortion and number of bits, $n \to \infty$. Substituting in equation (3) we get:

$$R(D) \geq 1 - \lim_{n\to\infty} \log_2 \sum_{i=0}^{r} \binom{n}{i} \tag{4}$$

But recall that

$$\binom{n}{i} \approx \frac{c}{\sqrt{n}} 2^{nH(\frac{k}{n})}$$

where

$$H(x) = -x \log_2 x - (1-x)\log_2(1-x)$$

On using Stirling's inequality and performing necessary calcukations, the result obtained is

$$\frac{2^{nH(D)}}{\sqrt{8D(1-D)n}} \leq \binom{n}{Dn} \leq \frac{2^{nH(D)}}{\sqrt{2\pi D(1-D)n}}$$

i.e.,

$$\frac{2^{nH(D)}}{\sqrt{8D(1-D)n}} \leq \sum_{i=0}^{Dn} \binom{n}{i} \leq 2^{nH(D)}$$

This approximation holds for any $D \leq \frac{1}{2}$. Using this result in equation (4) we get

$$R(D) \geq 1 - H(D)$$

This curve of compression ratio $R(D)$ v/s normalized compression ratio D is plotted in figure 4. If we allow normalized distortion to be $\frac{1}{2}$, we can have a code of zero rate. Distortion of $\frac{1}{2}$ means that given a file of length n, the codeword developed is at most a distance $\frac{n}{2}$ from the file, i.e distance between estimate and actual file is at most $\frac{n}{2}$. If size of file is 2, we will require only 1 bit to write it.

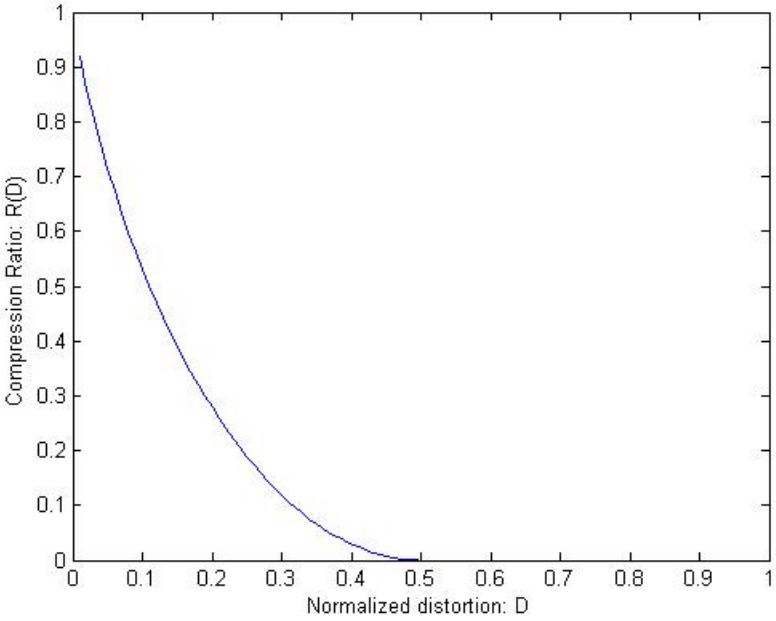


Figure 4: Plot of normalized compression ratio D v/s compression ratio R(D)

EE5585 Data Compression February 19, 2013

Lecture 9

Instructor: Arya Mazumdar *Scribe: Surya Ramachandran*

Rate distortion

The theory of quantization (Lossy compression) is given by Rate Distortion. Consider that we have a binary file, X. The file can take any n-length binary string, where n is any large integer. Assuming that we do not have any information about the statistics, the file can take any value.

$$X = \{0,1\}^n$$

If it is uniform, and every bit is independant and takes values 1 and 0 with equal probability of 1/2, then it cannot be compressed any further. The limit of the compression is given by the binary entropy function of half.

$$\mathrm{h}(1/2) = 1$$

Note: H(X) represents the Entropy of a general random variable; h(x) represents the binary entropy function of bernoulli random variables.

Hence, the file cannot be compressed any further without any loss. In this case, we introduce a distortion in reconstructing the file to further compress it. For binary strings, the distortion measure we use is Hamming Distortion. Let us consider two files-

10101**01**010
10101**10**010

These two files do not agree in two positions. Hence, the distance between them is 2. This distance is called the Hamming distance, a valid metric on the space of all n length binary strings. Hence, it has the following properties:

- It is always non-negative
- It is symmetric
- It satisfies the triangle inequality

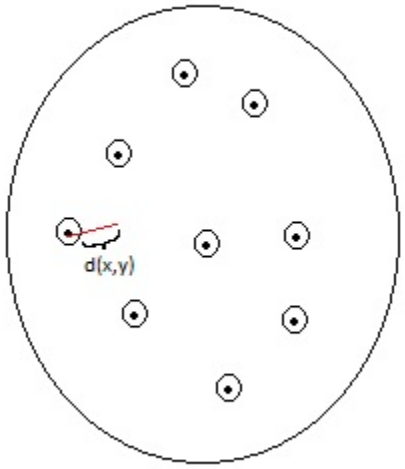
Code

A code, or source code C is a set of n length strings.

$$C \subseteq \{0,1\}^n$$

C represents the centers or cluster points in the subspace. Whenever we are given a file which is a part of the set, we try to map it to the nearest center. Then, we do a binary encoding of the center. The number of codewords in the set is represented by $|C|$. To have $|C|$ codewords, we require $\log|C|$ bits. To encode the entire space, we require n bits, because the size of the space is 2^n. But the size of $|C|$ is much lesser than 2^n, because of which we require fewer bits to encode, due to which we achieve compression. However, while reconstructing, although we know the center point, there is no way of knowing exactly which point in the space got mapped to that center. This gives rise to the distortion. A code C achieves distortion Dn, where $0 \leq D \leq 1$, if for all $y \in \{0,1\}^n$, $\exists x \in C$ such that $d(x,y) \leq nD$. Therefore, for any given point, we can find one codeword in the set, such that the point is atmost distance nD away.

Figure 1: Subspace of points



Let $M(n, D)$ be the size of the smallest code that recieves distortion nD. The compression ratio is given by $\frac{\log M(n,D)}{n}$.

$$R(D) = \lim_{n \to +\infty} \frac{\log M(n, D)}{n}$$

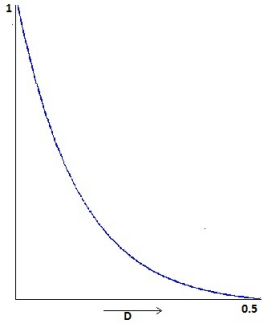
Assuming that this limit exists, to achieve a normalized distortion D, $R(D)$ is the max level of compression that can be done. D is called the *worst case distortion function*, because there is no probability associated with it.

Theorem

$$R(D) = 1 - h(D)$$

When D is $1/2$, the distortion is $n/2$. We can take two n length code words-

Figure 2: Rate Distortion



111...11
000...00

Now whatever binary string is given, the distance between the string and one of these codewords will be less than or equal to $n/2$. We can map the string to the code with the smaller distance. Hence, the worst case distortion is $n/2$. The given string is compressed to just one bit, (either 0 or 1). We can prove the theorem with the help of two claims.

Claim 1

$$R(D) \geq 1 - \mathrm{h}(D)$$

Proof Consider a code of size M. For any given point, we can find a center which has distortion atmost nD from that point. Let us draw spheres of radius nD around every center, covering every point in the space.

$$\begin{aligned} M \times \sum_{i=0}^{nD} \binom{n}{i} &\geq 2^n \\ M &\geq \frac{2^n}{\sum_{i=0}^{nD} \binom{n}{i}} \end{aligned}$$

We know that,

$$\begin{aligned} \sum_{i=0}^{nD} \binom{n}{i} &\leq 2^{n\mathrm{h}(D)} \\ M &\geq 2^{n[1-\mathrm{h}(D)]} \\ \frac{\log M}{n} &\geq 1 - \mathrm{h}(D) \\ R(D) &\geq 1 - \mathrm{h}(D) \end{aligned}$$

Claim 2

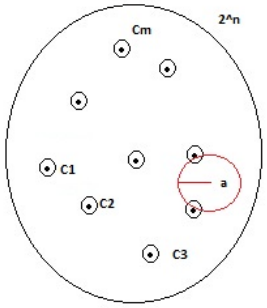
$$R(D) \leq 1 - h(D)$$

Proof: Choose a code C of size M.

$$M \equiv 2n \ln 2 \times \frac{2^n}{\sum_{i=0}^{nD} \binom{n}{i}}$$

This choice of code is random, uniform and independent of each other.

Figure 3: Random Coding



$$C = c_1, c_2, \ldots, c_m$$

A file is called a *Bad File* if there is no point within distance D to any point in C. Given a point a,

$$P(d(c_1, a) \leq nD) = \frac{\sum_{i=0}^{nD} \binom{n}{i}}{2^n}$$

The probability that a is bad w.r.t c_1 is

$$1 - \frac{\sum_{i=0}^{nD} \binom{n}{i}}{2^n}$$

Hence, the probaility that a is a bad file is

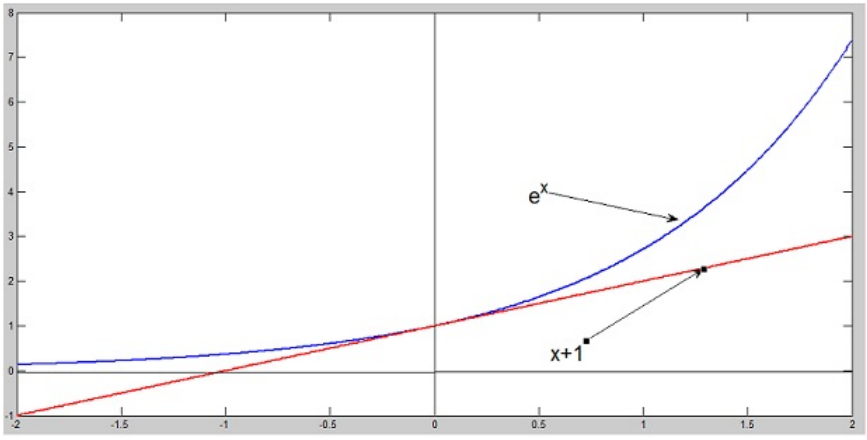
$$(1 - \frac{\sum_{i=0}^{nD} \binom{n}{i}}{2^n})^M$$

The average number of bad files is given by

$$2^n \times (1 - \frac{\sum_{i=0}^{nD} \binom{n}{i}}{2^n})^M$$

From the below graph, we can see that $e^x \geq 1 + x$ Hence, the average number of bad files is

Figure 4: Plot of e^x and $1 + x$



$$\leq 2^n \times (e^{-\frac{\sum_{i=0}^{nD} \binom{n}{i}}{2^n}})^M$$

This can be written as

$$e^{n \ln 2 - M \frac{\sum_{i=0}^{nD} \binom{n}{i}}{2^n}}$$

On expanding,

$$e^{n \ln 2 - 2n \ln 2 \frac{2^n}{\sum_{i=0}^{nD} \binom{n}{i}} \times \frac{\sum_{i=0}^{nD} \binom{n}{i}}{2^n}}$$

$$\begin{aligned} &= e^{-n \ln 2} \\ &= 2^{-n} \end{aligned}$$

2^{-n} is exponentially close to 0 and strictly less than 1. Therefore, the average number of bad files is less than 1. There has to be a code of size $2n \ln 2 \times \frac{2^n}{\sum_{i=0}^{nD} \binom{n}{i}}$ for which there are no bad files. There exists a code that achieves normalized distortion D, and has size $2n \ln 2 \times \frac{2^n}{\sum_{i=0}^{nD} \binom{n}{i}}$.

$$\begin{aligned}
\frac{\log M}{n} &= \frac{1}{n} \log \frac{2n \ln 2 \times 2^n}{\sum_{i=0}^{nD} \binom{n}{i}} \\
\sum_{i=0}^{nD} \binom{n}{i} &\geq \frac{2^{n\mathrm{h}(D)}}{\sqrt{8n(1-D)D}} \\
\frac{\log M}{n} &\leq \frac{1}{n} \log \frac{2^n 2n \ln 2 \sqrt{8n(1-D)D}}{2^{n\mathrm{h}(D)}} \\
&= 1 - \mathrm{h}(D) + \frac{1}{n} \log(2n \ln 2 \sqrt{8n(1-D)D})
\end{aligned}$$

When $n \to \infty$, the last term tends to 0. There exists a code that at a limit of $n \to \infty$, achieves the distortion $1 - \mathrm{h}(D)$. For the smallest code, this function is the rate distortion function.

$$R(D) \leq 1 - \mathrm{h}(D)$$

Hence, this proves Claim 2. Since both Claim 1 and 2 are valid, our Theorem $R(D) = 1 - \mathrm{h}(D)$ is proved.

We proved this theorem for the *worst case distortion*. Now, we shall look at *average case distortion*. In this case, there are certain probability distributions and statistics among the file. Let us consider an n length string, where each bit is a independent Bernoulli random variable. Code C achieves average distortion nD, if

$$E[d(X, C)] \leq nD$$

The distance $d(X, C)$ is the minimum of all the distances $d(X, Y)$, where $Y \in C$.

$$R_{av}(D) = \lim_{n \to +\infty} \frac{M_{av}(n, D)}{n}$$

Theorem

$$R_{av}(D) = 1 - \mathrm{h}(D)$$

For a large file, it does not matter whether it is the worst case distortion or average distortion. The same rate of compression is achievable.

Claim 1

$$R_{av}(D) \leq 1 - \mathrm{h}(D)$$

We have proved that

$$R(D) \leq 1 - \mathrm{h}(D)$$

for the worst case distortion. Because this holds for the worst case distortion D, it has to achieve an average distortion of at most D. This is called the *Direct part/ Achievability*.

Claim 2

$$R_{av}(D) \geq 1 - \mathrm{h}(D)$$

This is called the *Converse/Impossibility*.

Proof Suppose a code achieves average distortion nD.

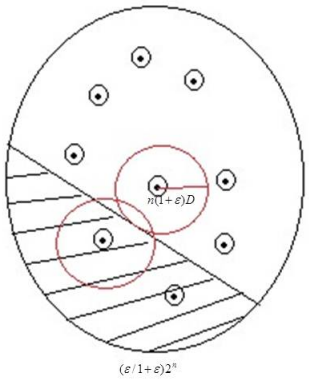
$$E(d(X, C) \leq nD$$

By *Markov's Inequality,*

$$\begin{aligned} P(d(X,C) > n(1+\epsilon)D) &\leq \frac{E[d(X,C)]}{n(1+\epsilon)D} \\ &\leq \frac{1}{1+\epsilon} \\ P(d(X,C) \leq n(1+\epsilon)D) &\geq \frac{\epsilon}{1+\epsilon} \end{aligned}$$

Hence, at least $\frac{\epsilon}{1+\epsilon}$ fraction of all points in $\{0,1\}^n$ has distortion less than or equal to $n(1+\epsilon)D$ for C.

Figure 5: Illustration of average distortion



If we draw spheres around the points in C, with radius $n(1+\epsilon)D$, this fraction of the points has to be covered.

$$\begin{aligned} |C| \sum_{i=0}^{n(1+\epsilon)D} \binom{n}{i} &\geq \frac{\epsilon}{1+\epsilon} 2^n \\ |C| &\geq \frac{\epsilon}{1+\epsilon} \frac{2^n}{\sum_{i=0}^{n(1+\epsilon)D} \binom{n}{i}} \\ \frac{\log|C|}{n} &\geq \frac{1}{n} \log \frac{\epsilon}{1+\epsilon} + 1 - \mathrm{h}((1+\epsilon)D) \end{aligned}$$

As $n \to \infty$,

$$\frac{log|C|}{n} \geq 1 - \mathrm{h}((1+\epsilon)D)$$

for any $\epsilon > 0$

$$\begin{aligned} R_{av}(D) &\geq 1 - \mathrm{h}((1+\epsilon)D) \\ R_{av}(D) &\geq 1 - \mathrm{h}(D) \end{aligned}$$

Therefore, $R_{av}(D) = 1 - \mathrm{h}(D)$ This function is called the *Rate Distortion function.* It can be defined for any source.

EE5585 Data Compression	January 22, 2013
Lecture 10	
Instructor: Arya Mazumdar	*Scribe: Mandy Ding*

Solutions for Homework 1

Problem 1 Suppose, $\mathcal{X} = \{A, B, C, D\}$. A source produces i.i.d. X symbols from this source,with $Pr(X = A) = P_A, Pr(X = B) = P_B, Pr(X = C) = P_C, Pr(X = D) = P_D$. You are given a file $F = AACDDBBBBBCAABCDAABAADCB$ generated by this source.
(a) What is your best guess for p_A, p_B, p_C, p_D? Reason.
(b) What is the entropy(in bits) of the probability distribution you guess?
(c) What is the Huffman code for the probability distribution that you guessed? What is the average number of bits per symbol?
(d) Encode the file F with the Huffman code you have designed. What is the length of the encoded binary file?What is the average number of bits that have been used for a symbol in this file?

Solution
(a)
The length of file F is 24 bits, and it has 8 of A, 8 of B, 4 of C and 4 of D. Hence we have

$$P_A = \frac{8}{24} \quad P_B = \frac{8}{24} \quad P_C = \frac{4}{24} \quad P_C = \frac{4}{24}$$

Next we prove that this guess is valid by solving the following maximization problem.

$$\begin{aligned} \text{maximize} \quad & f = {P_A}^8 {P_B}^8 {P_C}^4 {P_D}^4 \\ \text{subject to} \quad & 0 \leqslant P_A P_B P_C P_D \leqslant 1 \\ & P_A + P_B + P_C + P_D = 1 \\ & P_D = 1 - P_A - P_B - P_C \end{aligned}$$

Thus, rewrite the cost funciton

$$\begin{aligned} f(P_A, P_B, P_C) &= {P_A}^8 (P_B)^8 (P_C)^4 (1 - P_A - P_B - P_C)^4 \\ g(P_A, P_B, P_C) &= \log f(P_A, P_B, P_C) \\ &= 8 \log P_A + 8 \log P_B + 4 \log P_C + 4 \log P_D \end{aligned}$$

Get the partial derivatives of g

$$\begin{aligned} \frac{\partial g}{\partial P_A} &= \frac{8}{P_A} - \frac{4}{1 - P_A - P_B - P_C} = 0 \\ \frac{\partial g}{\partial P_B} &= \frac{8}{P_B} - \frac{4}{1 - P_A - P_B - P_C} = 0 \\ \frac{\partial g}{\partial P_C} &= \frac{8}{P_C} - \frac{4}{1 - P_A - P_B - P_C} = 0 \end{aligned}$$

By solving the above equations, we get the solutions

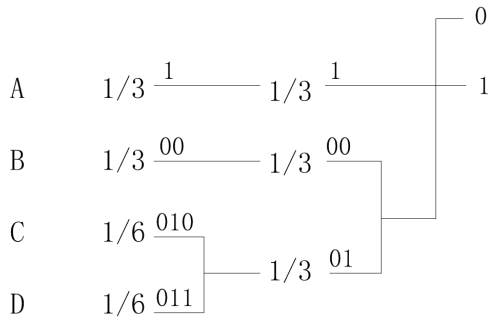
$$P_A = \frac{1}{3} \quad P_B = \frac{1}{3} \quad P_C = \frac{1}{6} \quad P_D = \frac{1}{6}$$

which are the same as we guess.

(b)

$$H(X) = -\frac{1}{3}\log\frac{1}{3} - \frac{1}{3}\log\frac{1}{3} - \frac{1}{6}\log\frac{1}{6} - \frac{1}{6}\log\frac{1}{6} = 1.92 \quad \text{bits/symbol}$$

(c)



Thus, the Huffman code set is

A	1
B	00
C	010
D	011

The average codeword length is $\frac{1}{3}\times 1 + \frac{1}{3}\times 2 + \frac{1}{6}\times 3 + \frac{1}{6}\times 3 = 2 \quad bits/symbol$

(d)
Encode the file with the Huffman code we designed in (c), we get the binary sequence

$$110100110110000000000010110001001111001101101000$$

Length of this binary sequence is $1\times 8 + 2\times 8 + 3\times 4 + 3\times 4 = 48$ bits
Average number of bits per symbol is $\frac{48}{24} = 2$ bits/symbol

Problem 2 Consider the code $0, 01$. Is this code uniquely decodable?Why?Is it instantaneous?

Solution
The code is uniquely decodable but not instantaneous.

Problem 3 Suppose $\chi = \{0, 1\}$. The random variable(source) X takes value in χ, with $Pr(X = 0) = \frac{3}{4}$ and $Pr(X = 0) = \frac{1}{4}$. What is the probability that the source produce a sequence 0000011111?

Solution

$$P = (\frac{3}{4})^{5}\,(\frac{1}{4})^{5}$$

Problem 4 Write the Lempel-Ziv parsing for the file F in Problem 1. What is the number of bits that you need to write the entire compressed file(with LZ algorithm).

Solution
The file F is parsed as following

$$F = \underset{1}{A}|\underset{2}{AC}|\underset{3}{D}|\underset{4}{DB}|\underset{5}{B}|\underset{6}{BB}|\underset{7}{BC}|\underset{8}{AA}|\underset{9}{BCD}|\underset{10}{AAB}|\underset{11}{AAD}|\underset{12}{C}|\underset{13}{B}$$

Therefore, the corresponding codewords are

$$(0, \mathrm{A})(1, \mathrm{C})(0, \mathrm{D})(3, \mathrm{B})(0, \mathrm{B})(5, \mathrm{B})(5, \mathrm{C})(1, \mathrm{A})(7, \mathrm{D})(8, \mathrm{B})(8, \mathrm{D})(0, \mathrm{C})$$

Generate initial dictionary

A	00
B	01
C	10
D	11

By encoding, we get the binary sequence

$$\underset{000000}{(0, A)} \quad \underset{000110}{(1, C)} \quad \underset{000011}{(0, D)} \quad \underset{001101}{(3, B)} \quad \ldots$$

As shown above, each phrase can be decoded into 6 bits, and we have 13 phrases, thus the length of the entire file is $6 \times 13 = 78 \quad bits/symbol$

Rate Distortion Theory

In the previous class, we have proved $R(D) = 1 - h(D)$. If there is a n-bits sequence allowing nD bits distortion, what is the optimal rate of compression?

Information Theory

Entropy

$$H(X) = -\sum_x p(x) \log p(x)$$

where $p(x) = Pr(X = x)$

Conditional Entropy

$$\begin{aligned} H(X|Y) &= \sum_y H(X|Y = y) \\ &= -\sum_x \sum_y p(y)p(x|y) \log p(x|y) \\ &= -\sum_x \sum_y p(x, y) \log p(x|y) \end{aligned}$$

which is the conditional entropy of X given Y

Joint Entropy

$$H(X, Y) = -\sum_x \sum_y p(x, y) \log p(x, y)$$

Claim 1 $H(X,Y) = H(Y) + H(X|Y)$

$$\begin{aligned}
H(Y) + H(X|Y) &= -\sum_y p(y)\log p(y) - \sum_x \sum_y p(x,y)\log p(x|y) \\
&= -\sum_y p(y)\log p(y) - \sum_x \sum_y p(x,y)\log p(x|y) \\
&= -\sum_x \sum_y p(x,y)\log p(y) - \sum_x \sum_y p(x,y)\log p(x|y) \\
&= -\sum_{x,y} p(x,y)\log p(x,y) \\
&= H(X,Y)
\end{aligned}$$

Claim 2 $H(Y) \geqslant H(Y|X)$ with equality when X and Y are independent.

$$\begin{aligned}
H(Y) - H(Y|X) &= -\sum_y p(y)\log p(y) + \sum_x \sum_y p(x,y)\log p(y|x) \\
&= -\sum_x \sum_y p(x,y)\log p(y) + \sum_x \sum_y p(x,y)\log p(y|x) \\
&= \sum_x \sum_y p(x,y)\log \frac{p(x,y)}{p(x)p(y)} \\
&= D(p(x,y)\|p(x)p(y))
\end{aligned}$$

By using the property of divergence, $D(X\|Y) \geqslant 0$ where X,Y are random variable, we get

$$H(Y) \geqslant H(Y|X)$$

Mutual Information

$$I(X,Y) = H(Y) - H(Y|X) = H(Y) - H(X|Y)$$

In the definition of mutual information, $H(Y)$ means the uncertainty in Y and $H(Y|X)$ means the uncertainty in Y if X is known. This equality implies that the amount of information of X gives about Y is the same as the that Y gives about X.

Property 1 $I(X,Y) = I(Y,X)$

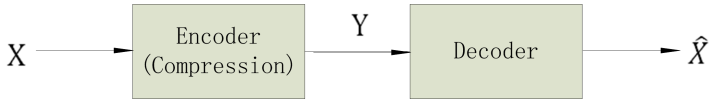
$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

$$H(X) - H(X|Y) = H(Y) - H(Y|X)$$

$$I(X,Y) = I(Y,X)$$

Rate Distortion

Suppose we have $X_1, X_2, X_3 \ldots X_n$ which are *i.i.d* random variables, and then compressed them into $Y_1, Y_2, Y_3 \ldots Y_k$. At the receiver, $Y_1, Y_2, Y_3 \ldots Y_k$ are decoded into $\hat{X_1}, \hat{X_2}, \hat{X_3} \ldots \hat{X_n}$



Let R(D) represent the optimal compression rate given D is the normalized distortion, we have $\frac{1}{n}Ed(X^n, \hat{X^n}) \leqslant D$ and define

$$R(D) = \min_{p(\hat{x}|x):\sum p(\hat{x},x)d(x,\hat{x})\leq D} I(X;\hat{X})$$