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 $\sim$ 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 $\sim$ 4.7 bits/Char
- Gzip $\sim$ 2.7 bits/Char
- Unix compress $\sim$ 3.7 bits/Char
- Most complicated scheme to date $\sim$ 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

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

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,*

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

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 ,*

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

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.

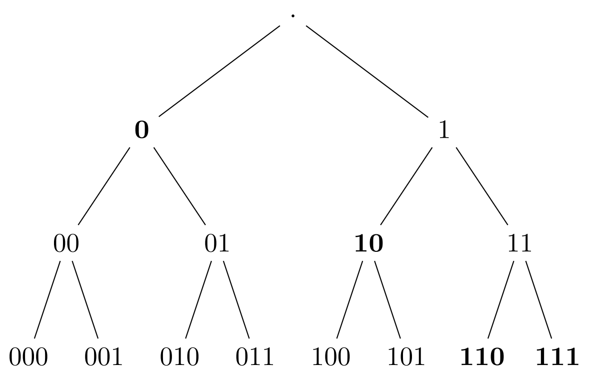
Example: $c(1) = 0, c(2) = 10, c(3) = 110, c(4) = 111$ is an instantaneous code.

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

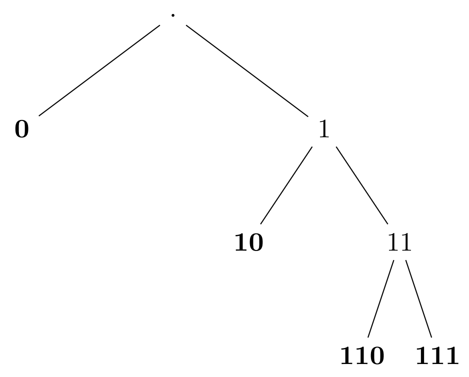
Example: Suppose $\mathcal{X} = \{1,2,3,4\}$ and $p(1) = 0.25, p(2) = 0.25, p(3) = 0.25, p(4) = 0.25$.

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

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,

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

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

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

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

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

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

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

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

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

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

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

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 $|\mathcal{X}| =$ m then say we have $\ell_1, \ell_2, \ldots, \ell_m$. Then we need to

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

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

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,

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

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

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

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

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).