Proof of Kraft-McMillan theorem*

Nguyen Hung Vu†

平成 16 年 2 月 24 日

1 Kraft-McMillan theorem

Let $\mathcal{C}$ be a code with n codewords with lenghth $l_1, l_2, ..., l_N$. If $\mathcal{C}$ is uniquely decodable, then

$$K(\mathcal{C}) = \sum_{i=1}^{N} 2^{-l_i} \leq 1 \quad (1)$$

2 Proof

The proof works by looking at the nth power of $K(\mathcal{C})$. If $K(\mathcal{C})$ is greater than one, then $K(\mathcal{C})^n$ should grw expoentially with n. If it does not grow expoentially with n, then this is proof that $\sum_{i=1}^{N} 2^{-l_i} \leq 1$.

Let n be and arbitrary integer. Then

$$\left[\sum_{i=1}^{N} 2^{-l_i}\right]^n = \\ = \left(\sum_{i_1=1}^{N} 2^{-l_{i_1}}\right)\cdots \\ \left(\sum_{i_2=1}^{N} 2^{-l_{i_2}}\right)\left(\sum_{i_N=1}^{N} 2^{-l_{i_n}}\right)$$

and then

$$\left[\sum_{i=1}^{N} 2^{-l_i}\right]^n = \sum_{i_1=1}^{N}\sum_{i_2=1}^{N}\cdots\sum_{i_n=1}^{N} 2^{-(l_{i_1+i_2+\cdots+i_n})} \quad (2)$$

The exponent $l_{i_1} + l_{i_2} + \cdots + l_{i_n}$ is simply the length of n codewords from the code $\mathcal{C}$. The smallest value this exponent can take is greater than or equal to n, which would be the case if codwords were 1 bit long. If

$$l = max\{l_1, l_2, \cdots, l_N\}$$

then the largest value that the exponent can take is less than or equal to nl. Therefore, we can write this summation as

$$K(C)^n = \sum_{n}^{nl} A_k 2^{-k}$$

where A_k is the number of combinations of n codewords that have a combined length of k. Let's take a look at the size of this coeficient. The number of possibledistinct binary sequences of length k is 2^k. If this code is uniquely decodable, then each sequence can represent one and only one sequence of codewords. Therefore, the number of possible combinations of codewords whose combined length is k cannot be greater than 2^k. In other words,

$$A_k \leq 2^k.$$

This means that

$$K(C)^n = \sum_{k=n}^{nl} A_k 2^{-k} \leq \sum_{k=n}^{nl} 2^k 2^{-k} = nl - n + 1 \quad (3)$$

But $K(\mathcal{C})$ is greater than one, it will grow exponentially with n, while $n(l-1)+1$ can only grow linearly, So if $K(\mathcal{C})$ is greater than one, we can always find an n large enough that the inequality (3) is violated. Therefore, for an uniquely decodable code $\mathcal{C}$, $K(\mathcal{C})$ is less than or equal ot one.

This part of of Kraft-McMillan inequality provides a neccessary condition for uniquely decodable codes. That is, if a code is uniquely decodable, the codeword lengths have to satisfy the inequality.

3 Construction of prefix code

Given a set of integers $l_1, l_2, \cdots, l_N$ that satisfy the inequality

$$\sum_{i=1}^{N} 2^{-l_i} \leq 1 \quad (4)$$

we can always find a prefix code with codeword lengths $l_1, l_2, \cdots, l_N$.

4 Proof of prefix code constructing theorem

We will prove this assertion by developing a procedure for constructing a prefix code with codeword lengths $l_1, l_2, \cdots, l_N$ that satisfy the given inequality. Without loss of generality, we can assume that

$$l_1 \leq l_2 \leq \cdots \leq l_N. \quad (5)$$

Define a sequence of numbers $w_1, w_2, \cdots, w_N$ as follows:

$$\begin{aligned} w_1 &= 0 \\ w_j &= \sum_{i=1}^{j-1} 2^{l_j - l_i} \qquad j > 1. \end{aligned}$$

The binary representation of w_j for $j > 1$ would take up $\lceil \log_2 w_j \rceil$ bits. We will use this binary representation to construct a prefix code. We first note that the number of bits in the binary representation of w_j is less than or equal to l_j. This is obviously true for w_1. For $j > 1$,

$$\begin{aligned} \log_2 w_j &= \log_2 \left[\sum_{i=1}^{j-1} 2^{l_j - l_i}\right] \\ &= \log_2 \left[2^{l_j} \sum_{i=1}^{j-1} 2^{-l_i}\right] \\ &= l_j + \log_2 \left[\sum_{i=1}^{j-1} 2^{-l_i}\right] \\ &\leq l_j. \end{aligned}$$

*Introduction to Data Compression, Khalid Sayood

†email: vuhung@fedu.uec.ac.jp

The last inequality results from the hypothesis of the theorem that $\sum_{i=1}^{N} 2^{-l_i} \leq 1$, which implies that $\sum_{i=1}^{j-1} 2^{l_i} \leq 1$. As the logarithm of a number less than one is negative, $l_j + \log_2 \left[\sum i = 1^{j-1} 2^{-l_i}\right]$ has to be less than l_j.

Using the binary representation of w_j, we can devise a binary code in the following manner: If $\lceil \log_2 w_j \rceil = l_j$, then the jth codeword c_j is the binary representation of w_j. If $\lceil \log_2 w_j \rceil < l_j$, then c_j is the binary representation of w_j, with $l_j - \lceil \log_2 w_j \rceil$ zeros appended to the right. This certainly a code, but is it a prefix code? If we can show that the code $C = \{c_1, c_2, \cdots, c_N\}$ is a prefix, then we will have proved the theorem by construction.

Suppose that our claim is not true.. The for some $j < k$, c_j is a refix of c_k. This means that l_j most significant bits of w_k form the binary representation of w_j. Therefore if we right–shift the binary of w_k by $l_k - l_j$ bits, we should get the binary representation for w_j. We can write this as

$$w_j = \lfloor \frac{w_k}{2^{l_k - l_j}} \rfloor.$$

However,

$$w_k = \sum_{i=1}^{k-1} 2^{l_k - l_i}.$$

Therefore,

$$\begin{aligned} \frac{w_k}{2^{l_k - l_j}} &= \sum_{i=0}^{k-1} 2^{l_j - l_i} \\ &= w_j + \sum_{i=j}^{k-1} 2^{l_j - l_i} \\ &= w_j + 2^0 + \sum_{i=j+1}^{k-1} 2^{l_j - l_k} \\ &\geq w_j + 1 \end{aligned} \tag{6}$$

That is, the smalleset value for $\frac{w_k}{2^{l_k - l_j}}$ is $w_j + 1$. This constradicts the requirement for c_j being the prefix of c_k. Therefore, c_j cannot be the prefix of c_k. As j and k were arbitrary, this means that no codword is a prefix of another codeword, and the code C is a prefix code.

5 Projects and Problems

1. Suppose X is a random variable that takes on values from an M–letter alphabet. Show that $0 \leq H(X) \leq \log_2 M$.
   Hints: $H(X) = -\sum_{i=1}^{n} P(\alpha_i) \log P(\alpha_i)$ where, $M = \{\alpha_1 \times m_1, \alpha_2 \times m_2, \cdots, \alpha_n \times m_n\}, \alpha_i \neq \alpha_j \forall i \neq j, \sum_{i=1}^{n} m_i = M$. $H(X) = -\sum_{i=1}^{n} \frac{m_i}{M} \log \frac{m_i}{M} = -\frac{1}{M} \sum_{i=1}^{n} m_i (\log m_i - \log M) = -\frac{1}{M} \sum m_i \log m_i + \frac{1}{M} \log M \sum m_i = \boxed{-\frac{1}{M} \sum m_i \log m_i + \log M}$

2. Show that for the case where the elements of an observed sequence are *idd* [1] ,the entropy is equal to the first-order entropy.
   Hints: First–order entropy is defined by: $H_1(S) = \lim_{n \to \infty} \frac{1}{n} G_n$, where $G_n = -\sum_{i_1=1}^{i_1=m} \sum_{i_2=1}^{i_2=m} \cdots \sum_{i_n=1} i_n = mP(X_1 = i_1, X_2 = i_2, \cdots, X_n = i_n) \log P(X_1 = i_1, X_2 = i_2, \cdots, X_n = i_n)$. And second–order entropy is defined by $H_2(S) = -\sum P(X_1) \log P(X_1)$. In case of *iid*, $G_n = -n \sum_{i_1=1}^{i_1=m} P(X_1 = i_1) \log P(X_1 = i_1)$. Prove it!

3. Given an alphabet $A = \{a_1, a_2, a_3, a_4\}$, find the first–order entropy in the following cases:
   (a) $P(a_1) = P(a_2) = P(a_3) = P(a_4) = \frac{1}{4}$.
   (b) $P(a_1) = \frac{1}{2}, P(a_2) = \frac{1}{4}, P(a_3) = P(a_4) = \frac{1}{8}$.
   (c) $P(a_1) = 0.505, P(a_2) = \frac{1}{4}, P(a_3) = \frac{1}{8}, P(a_4) = 0.12$.

4. Suppose we have a source with a probability model $P = \{p_0, p_1, \cdots, p_m\}$ and entropy H_P. Suppose we have another sourcewith probability model $Q = \{q_0, q_1, \cdots q_m\}$ and entroy H_Q, where

   $$q_i = p_i \qquad i = 0, 1, \cdots, j-2, j+1, \cdots, m$$

   and

   $$q_j = q_{j-1} = \tfrac{p_j + p_{j-1}}{2}$$

   How is H_Q related to H_P(greater, equal, or less)? Prove your answer.
   Hints: $H_Q - H_P = \sum p_i \log p_i - \sum q_i \log q_i = p_{j-1} \log p_{j-1} + p_j \log p_j - 2\frac{p_{j-1}+p_j}{2} \log \frac{p_{j-1}+p_j}{2} = \phi(p_{j-1}) + \phi(p_j) - 2\phi(\frac{p_{j-1}+p_j}{2})$. Where $\phi(x) = x \log x$ and $\phi''(x) = 1/x > 0\ \forall x > 0$.

5. There are several image and speech files among the accompanying data sets.
   (a) Write a program to compute the first–order entropy of some of the image and speech files.
   (b) Pick one of the image files and compute is second–order entropy. Comment on the difference between the first–order and second–order entropies.
   (c) Compute the entropy of the difference between neighboring pixels for the image you used in part (b). Comment on what you discovered.

6. Conduct an experiemnt to see how well a model can describe a source.
   (a) Write a program that randomly selects letters from a 26–letter alphabet $\{a, b, c, \cdots, z\}$ and forms four–letter words. Form 100 such words and see how many of these words make sence.
   (b) Among the accompanying data sets is a file called `4letter.words` , which contains a list of four–letter words. Using this file, obtain a probability model for the alphabet. Now repeat part (a) generating the words using probability model. To pick letters according to a probability model, construct the cumulative density function $(cdf) F_X(x)$[2]. Using a uniform pseudorandom number generator to generate a value r, where $0 \leq r < 1$, pick the letter x_k if $F_X(x_k - 1) \leq r < F_X(x_k)$. Compare your results with those of part (a).
   (c) Repeat (b) using a single–letter context.
   (d) Repeat (b) using a two–letter context.

7. Determine whether the following codes are uniquely decodable:

[1]iid: independent and identical distributed

[2]*cumulative distribution function* (cdf) $F_X(x) = P(X \leq x)$

(a) $\{0, 01, 11, 111\}$(01 11 = 0 111)

(b) $\{0, 01, 110, 111\}$ (01 110 = 0 111 0)

(c) $\{0, 10, 110, 111\}$, Yes. Prove it!

(d) $\{1, 10, 110, 111\}$, (1 10 = 110)

8. Using a text file compute the probabilities of each letter p_i.

   (a) Assume that we need a codeword of length $\lceil \frac{1}{\log_2 p_i} \rceil$ to encode the letter i. Determine the number of bits needed to encode the file.

   (b) Compute the conditional probability $P(i/j)$ of a letter i given that the previous letter is j. Assume that we need $\lceil \frac{1}{\log_2 P(i/j)} \rceil$ to represent a letter i that follows a letter j. Determine the number of bits needed to encode the file.

6 Huffman coding

6.1 Definition

A minimal variable-length character encoding based on the frequency of each character. First, each character becomes a trivial tree, with the character as the only node. The character's frequency is the tree's frequency. The two trees with the least frequencies are joined with a new root which is assigned the sum of their frequencies. This is repeated until all characters are in one tree. One code bit represents each level. Thus more frequent characters are near the root and are encoded with few bits, and rare characters are far from the root and are encoded with many bits.

6.2 Example of Huffman encoding design

Alphabet $A = \{a_1, a_2, a_3, a_4\}$ with $P(a_1) = P(a_2) = 0.2$ and $P(a_4) = P(a_5) = 0.1$. The entropy of this source is 2.122bits/symbol. To design the Huffman code, we first sort the letters in a descending probability order as shown in table below. Here $c(a_i)$ denotes the codewords as

$$
\begin{aligned}
c(a_4) &= \alpha_1 * 0 \\
c(a_5) &= \alpha_1 * 1
\end{aligned}
$$

where α_1 is a binary string, and * denotes concatnation.

The initial five–letter alphabet

Letter	Probability	Codeword
a_2	0.4	$c(a_2)$
a_1	0.2	$c(a_1)$
a_3	0.2	$c(a_3)$
a_4	0.1	$c(a_4)$
a_5	0.1	$c(a_5)$

Now we define a new alphabe A' with a four–letter $a_1, a_2, a_3, a_4^{'}$ where $a_4^{'}$ is composed of a_4 and a_5 and has a probability $P(a_4^{'}) = P(a_4) + P(a_5) = 0.2$. We sort this new alphabet in descending order to obtain following table.

The reduced four–letter alphabet

Letter	Probability	Codewords
a_2	0.4	$c(a_2)$
a_1	0.2	$c(a_1)$
a_3	0.2	$c(a_3)$
$a_4^{'}$	0.2	α_1

In this alphabet, a_3 and $a_4^{'}$ are the two letters at the bottom of the sorted list. We assign their codewords as

$$
\begin{aligned}
c(a_3) &= \alpha_2 * 0 \\
c(a_4^{'}) &= \alpha_2 * 1
\end{aligned}
$$

but $c(a_4^{'}) = \alpha_1$. Therefore

$$\alpha_1 = \alpha_2 * 1$$

which means that

$$
\begin{aligned}
c(a_4) &= \alpha_2 * 10 \\
c(a_5) &= \alpha_2 * 11.
\end{aligned}
$$

At this stage, we again define a new alphabet A'' that consists of three letters $a_1, a_2, a_3^{'}$, where $a_3^{'}$ is composed of a_3 and $a_4^{'}$ and has a probability $P(a_3^{'}) = P(a_3) + P(a_4^{'}) = 0.4$. We sort this new alphabet in descending order to obtain the following table.

The reduced four–letter alphabet

Letter	Probability	Codewords
a_2	0.4	$c(a_2)$
$a_3^{'}$	0.4	α_2
a_1	0.2	$c(a_1)$

In this case, the two least probable symbols are a_1 and $a_3^{'}$. Therefore,

$$
\begin{aligned}
c(a_3^{'}) &= \alpha_3 * 0 \\
c(a_1) &= \alpha_3 * 1
\end{aligned}
$$

But $c(a_3^{'}) = \alpha_2$. Therefore,

$$\alpha_2 = \alpha_3 * 0$$

which means that.

$$
\begin{aligned}
c(a_3) &= \alpha_3 * 00 \\
c(a_4) &= \alpha_4 * 010 \\
c(a_5) &= \alpha_5 * 011.
\end{aligned}
$$

Again we define a new alphabet, this time with only two letter $a_3^{''}, a_2$. Here $a_3^{''}$ is composed of letters $a_3^{'}$ and a_1 and has probability $P(a_3^{''}) = P(a_3') + P(a_1) = 0.6$. We now have the following table

The reduced four–letter alphabet

Letter	Probability	Codewords
a_3''	0.6	α_3
a_2	0.4	$c(a_2)$

As we have only two letters, the codeword assignment is straghtforward:

$$
\begin{aligned}
c(a_3'') &= 0 \\
c(a_2) &= 1
\end{aligned}
$$

which means that $\alpha_3 = 0$, which in turn means that

$$
\begin{aligned}
c(a_1) &= 01 \\
c(a_3) &= 000 \\
c(a_4) &= 0010 \\
c(a_5) &= 0011
\end{aligned}
$$

and the Huffman finally is given here.

The reduced four–letter alphabet		
Letter	Probability	Codewords
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011

6.3 Huffman code is optimal

The neccessary conditions for an optimal variable binary code are as follows:

1. **Condition 1**: Give any two letter a_j and a_k, if $P(a_j) \geq P(a_k)$, then $l_j \leq l_k$, where l_j is the number of bits in the codeword for a_j.
2. **Condition 2**: The two least probable letters have codewords with the same maximum length l_m.
3. **Condition 3**: In the tree corresponding to the optimum code, there must be two branches stemming[3] from each intermediate node.
4. **Condition 4**: Suppose we change an intermediate node into a leaf node by combining all the leaves descending from it into a composite word of a reduced alphabet. Then, if the original tree was optimal for the original alphabet, the reduced tree is optimal for reduced alphabet.

The Huffman code satisfies all conditions above and therefore be optimal.

6.4 Average codeword length

For a source S with alphabet $A = \{a_1, a_2, \cdots, a_K\}$, and probability model $\{P(a_1), P(a_2), \cdots, P(a_K)\}$, the average codeword length is given by

$$\bar{l} = \sum_{i=1}^{K} P(a_i) l_i.$$

The difference between the entropy ofthe sourcce H(S) and the average length is

$$\begin{aligned} H(S) - \bar{l} &= -\sum_{i=1}^{K} P(a_i) \log_2 P(a_i) - \sum_{i=1}^{K} P(a_i) l_i \\ &= \sum P(a_i) \left(\log \left[\frac{1}{P(a_i)} - l_i \right] \right) \\ &= \sum P(a_i) \left(\log \left[\frac{1}{P(a_i)} \right] - \log_2(2^{l_i}) \right) \\ &= \sum P(a_i) \log \left[\frac{2^{-l_i}}{P(a_i)} \right] \\ &\leq \log_2 \left[\sum 2^{l_i} \right] \end{aligned}$$

6.5 Length of Huffman code

It has been proven that

$$H(S) \leq \bar(l) < H(S) + 1 \tag{7}$$

where H(S) is the entropy of the source S.

[3] 生じる

6.6 Extended Huffman code

Consider an alphabet $A = \{a_1, a_2, \cdots, a_m\}$. Each element of the sequence is generated independently of the other elements in the sequence. The entropyo f this source is give by

$$H(S) = -\sum_{i=1}^{N} P(a_i) \log_2 P(a_i)$$

We know that we can generate a Huffman code for this source with rate r such that

$$H(S) \leq R < H(S) + 1 \tag{8}$$

We have used the looser bound here; the same argument can be made with the tighter bound. Notice thatwe have used "rate R" to denote the number of bits per symbol. This is a standard convention in the data compression literate. However, in the comunication literature, the word "rate" ofter refers to the number of bits per second.

Suppose we now encode the sequence by generating one codeword for every n symbols. As there are m^n combinations of n symbols, we will need m^n codewords in our Huffman code. We could generate this code by viewing m^n symbols as letters of an extended alphabet. $\mathcal{A}^{(n)} = \{\overbrace{a_1 a_1 ... a_1}^{ntimes}, a_1 a_1 ... a2, ..., a2, a_1 a_1, ... a_m, a_1 a_1 ... a_2 a1, ..., a_m a_m ... a_m\}$ from a source $S^{(n)}$. Denote the rate for the new source as $R^{(n)}$. We know

$$H(S^{(n)}) \leq R^{(n)} < H(S^{(n)}) + 1. \tag{9}$$

$$R = \frac{1}{n} R^{(n)}$$

and

$$\frac{H(S^{(n)})}{n} \leq R < \frac{H(S^{(n)})}{n} + \frac{1}{n}$$

It can be proven that

$$H(S^{(n)}) = nH(S)$$

and therefore

$$H(S) \leq R \leq H(S) + \frac{1}{n}$$

6.6.1 Example of Huffman Extended Code

$\mathcal{A} = \{a_1, a_2, a_3\}$ with probabilities model $P(a_1) = 0.8, P(a_2) = 0.02$ and $P(a_3) = 0.18$. The entropy for this source is 0.816 bits per symbol. Huffman code this this source is shown below

a_1	0
a_2	11
a_3	10

and the extended code is

$a_1 a_1$	0.64	0
$a_1 a_2$	0.016	10101
$a_1 a_3$	0.144	11
$a_2 a_1$	0.016	101000
$a_2 a_2$	0.0004	10100101
$a_2 a_3$	0.0036	1010011
$a_3 a_1$	0.1440	100
$a_3 a_2$	0.0036	10100100
$a_3 a_3$	0.0324	1011

6.7 Nonbinary Huffman codes

6.8 Adaptive Huffman Coding

7 Arithmetic coding

[4]Use the mapping

$$X(a_i) = i, \quad a_i \in \mathcal{A} \tag{10}$$

where $\mathcal{A} = \{a_1, a_2, ...a_m\}$ is the alphabet for a discrete source and X is a random variable. This mapping means that given a probability model $mathcalP$ for the source, we also have a probability density function for the random variable

$$P(X = i) = P(a_i)$$

and the curmulative density function can be defined as

$$F_X(i) = \sum_{k=1}^{i} P(X = k).$$

Define $\bar{T}_X(a_i)$ as

$$\bar{T}_X(a_i) = \sum_{k=1}^{i-1} P(X = k) + \frac{1}{2}P(X = i) \tag{11}$$

$$= F_X(i-1) + \frac{1}{2}P(X = i) \tag{12}$$

For each $a_i, \bar{T}_X(a_i)$ will have a unique value. This value can be used as a unique tag for a_i. In general

$$\bar{T}_X^{(m)}(x_i) = \sum_{y<x_i} P(y) + \frac{1}{2}P(x_i) \tag{13}$$

[4]Arithmetic: 算数, 算術