EE376A Information Theory Lecture 7 - 01/29/2015

Lecture 7

Lecturer: Tsachy Weissman *Scribe: Bill Chen, Jessica Taylor*

1 Variable Length Lossless Compression (Continued)

Last lecture, we defined the notions of Uniquely Decodable (UD), and prefix codes. We also found the shortest achievable code length for prefix codes for Dyadic distributed sources as well as for General sources. For Dyadic distributed sources, we found the steps for obtaining such an optimal prefix code scheme.

Today, we will learn how to achieve such optimal prefix code scheme for any source distributions. Before that, let's do a quick review with a Dyadic distributed source example.

Example 1. Prefix code for a 4-symbol dyadic source: Let $\mathcal{U} = \{a, b, c, d\}$.

u	p(u)	c(u)	l(u)
a	1/2	0	1
b	1/4	10	2
c	1/8	110	3
d	1/8	111	3

Here we see the prefix code having lengths $l(u) = -\log P(u)$.

In summary, again, for dyadic sources we know how to construct a prefix code with $l(u) = -\log P(u)$, which is the ideal length function. For general sources we can construct the ideal prefix code with $l(u) = \lceil -\log P(u) \rceil$, i.e., the length function takes the upper integer value of the log of the probably. A code with such length function property is called "Shannon Code".

1.1 Notes on Shannon Code

The expected code length for a Shannon code can be expressed as the following:

$$\bar{l} = \sum_u P(u)l(u) = \sum_u P(u)\lceil -\log P(u) \rceil \leq \sum_u P(u)(-\log P(u) + 1) = H(U) + 1$$

Therefore, the expected code length is always less or equal to the entropy plus 1. This result could be good or bad depending on how large $H(U)$ is to start with. If the extra "1" is too much, alternatively, we can construct a Shannon code for the multi-symbol $u^n = (u_1, u_2, ...u_n)$, where u_i is memoryless. Then,

$$\bar{l}_n \leq H(U^n) + 1 \text{ or } \frac{1}{n}\bar{l}_n \leq \frac{1}{n}H(U^n) + \frac{1}{n} = H(U) + \frac{1}{n}$$

Now we can make it arbitrarily close to the entropy. In the end, there is a trade-off between ideal code length and memory since the code map is essentially a lookup table. If n gets too large, the exponential increase in lookup table size could be a problem.

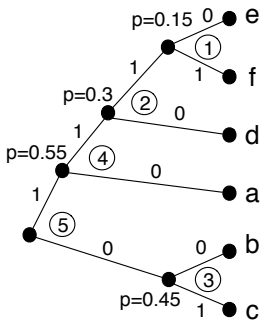
1.2 Best Prefix Code for General Source

The answer to the best prefix code for a general source code distribution is the Huffman Code. It is, however, constructed exactly as for the dyadic source case.

Example 2. Prefix code for a senary source (six letters):

u	p(u)
a	0.25
b	0.25
c	0.2
d	0.15
e	0.1
f	0.05

To find the code c(u), we follow these steps:
① Find 2 symbols with the smallest probability and then merge them to create a new "node" and treat it as a new symbol.
② Then merge the next 2 symbols with the smallest probability to create a new "node"
③ - ⑤ Repeat steps 1 and 2 until there is only 1 symbol left. At this point, we created a binary tree. The paths traversed from the root to the leaves are the prefix codes.



u	c(u)	l(u)
e	1110	4
f	1111	4
d	110	3
a	10	2
b	00	2
c	01	2

2 Optimality of Huffman codes

Theorem 3. *The Huffman code is an optimal prefix code.*

Proof

Assume $U \sim P$, with alphabet $\mathcal{U} = \{1, 2, ..., r\}$. Let c and L be the code and length function for U, respectively, and let $\bar{l} = \mathbb{E}[l(U)]$. Assume without loss of generality that $p(1) \geq p(2) \geq ... \geq p(r)$.

Let U_{r-1} denote the random variable with alphabet $\{1, 2, ..., r-1\}$ obtained from U by merging the two symbols with smallest probabilities (that is, r and $r-1$). At some point, this may require reordering the symbols so that the symbols of U_{r-1} also have descending probabilities, but this aspect can be ignored for now.

Let c_{r-1}, l_{r-1} denote the code and length function for U_{r-1}, respectively, and let $\bar{l}_{r-1} = \mathbb{E}[l_{r-1}(U_{r-1})]$

Similarly, let U_{r-2} denote the random variable with alphabet $\{1, 2, ..., r-2\}$ obtained by merging the 2 least likely symbols of U_{r-1}, etc.

Given a prefix code c_{r-1} for U_{r-1}, we can obtain a prefix code c for U by splitting the code word for the symbol of U_{r-1} created by merging r with $r-1$. Specifically:

$$\begin{aligned} c(i) &= c_{r-1}(i) && \text{for } 1 \leq i \leq r-2 \\ c(r-1) &= c_{r-1}(r-1)0 \\ c(r) &= c_{r-1}(r-1)1 \end{aligned}$$

where juxtaposition with 0 or 1 indicates concatenation.

Lemma 4. *Let c_{r-1}^{opt} be an optimal prefix code for U_{r-1}. Then c, formed by splitting c_{r-1}^{opt}, is an optimal prefix code for U.*

Proof Note there is an optimal prefix code for U satisfying:

1. $l(1) \leq l(2) \leq ... \leq l(r)$. Otherwise, we could rearrange the codes to satisfy this property, and the result would be at least as good.

2. $l(r-1) = l(r) = l_{max}$. This is a result of the fact that there is no use of having a code that is longer than all of the other codes: it could be shortened without preventing the code from being a prefix code.

3. $c(r-1)$ and $c(r)$ differ only in the last bit. If this was not the case, then we could swap the prefixes of $c(r-1)$ and $c(r)$ up to the last bit with other codes in order to satisfy the property.

The rest of the proof will follow in the next lecture. □

Assuming the above lemma, the argument is as follows: by construction, the Huffman code for U is obtained from the Huffman code for U_{r-1} by splitting. By our lemma, we now only need to prove that the Huffman code for U_{r-1} is optimal. But the Huffman code for U_{r-1} is obtained from the Huffman code for U_{r-2} by splitting, so by our lemma we now only need to prove that the Huffman code for U_{r-2} is optimal. The argument can be applied recursively until we need only to prove that the Huffman code for U_2 is optimal, which is trivially true. □