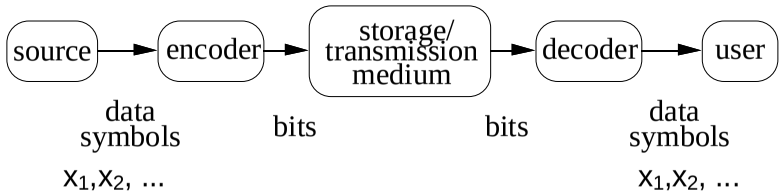
EECS 651 Lossless Source Coding Winter 2001

Formerly called noiseless coding, or coding for a noiseless channel.

References: Sections 9.1-9.5, G&G, Sayood: 3.1-3.3
See also books on information or communication theory.

Block diagram of source coding system



Goal

encode the data symbols with bits in such a way that a reproduction can be created from the bis

Performance

accuracy: in lossless coding demand perfect reproduction;
i.e. decoder produces same data symbols as source

efficiency: rate = R = avg. no.of bits per source symbol
should be as small as possible,

First-order lossless codes

Encode one source symbol at a time, independently of other symbols. We focus first on these

Other types of lossless codes: to be considered later

Block codes:

- Fixed-length to fixed-length
- Fixed-to-variable length
- Variable-to-fixed length
- Variable-to-variable length

Non-block codes:

- conditional codes
- arithmetic codes
- Lempel-Ziv

Examples: Source alphabet: A = {1,2,3,4} , Probabilities: P = {.5, .25, .125, .125}

Several codes: codebooks and encoding rules

p(x)	x		code a	code b	code c	code d	code e	code f	code g
.5	1	$\underline{v}_1$ =	00	0	10	0	0	0	0
.25	2	$\underline{v}_2$ =	01	10	110	10	10	1	01
.125	3	$\underline{v}_3$ =	10	110	111	10	11	01	011
.125	4	$\underline{v}_4$ =	11	111	0	11	111	10	0111
	rate		2	1.75	2.25	1.5	1.625	1.25	1.875
			FL	VL		not UD	not UD	not UD	UD, not prefix

code a: fixed-length

code b: variable-length; smaller rate than code a

code c: same codewords as code b; different assignment; larger rate.

code d: better rate than b; but doesn't distinguish 2 and 3

code e: good rate, but can't distinguish 32 from 41. $32 \rightarrow 1110$, $41 \rightarrow 1110$

code f: good rate, can't distinguish $3 \rightarrow 01$ from $1\ 2 \rightarrow 01$.

code g: simple to encode and decode, not as good as b.

How do we know codes a, b and c are UD? Note that a and b have the property that no codeword is prefix of another. Describe their decoding.

Source

- Modeled as a sequence of random variables $X_1, X_2, \ldots,$ where X_i is a discrete random variable representing the ith source symbol, i.e. a discrete-time, discrete-valued random process.
- Unless otherwise stated we assume the random variables are identically distributed. That is, each has the same distribution as some generic discrete random variable X.
- The performance of first-order lossless codes depends only on the probability distribution of the individual X_i's, i.e. we don't need joint distributions.
- Let A denote the possible outcomes of X, i.e. the possible data symbols. Let M denote the cardinality of A, i.e. the number of distinct data symbols. There are two cases:

  (a) $M < \infty$, i.e. A is a finite alphabet. We write $A = \{a_1,\ldots,a_M\}$, $M < \infty$; or wlog $A = \{1, 2, \ldots, M\}$.

  (b) $M = \infty$, i.e. A is a countably infinite alphabet. We write $A = \{a_1,a_2, \ldots\}$ or wlog $A = \{1, 2, 3, \ldots\}$

  Let $P_i = Pr(X=a_i)$ or $Pr(X=i)$.

  Let $\underline{P} = (P_1, P_2,\ldots)$ denote the ordered set of probablities.
- Note: Lossless source codes do not exist for continuous-time or continuous-valued sources.

First-order, variable-length codes (aka variable-length codes)

Described by a binary codebook $C = \{\underline{v}_1, \underline{v}_2, \ldots\}$ of binary sequences of varying lengths.

Encoding rule: $e(a_i) = \underline{v}_i$ or $e(i) = \underline{v}_i$

That is, we assume the ordering of the codewords determines the assignment of codewords to symbols in the source alphabet.

Decoding rule: specific form depends on encoding rule

Rate: R = average length = $L = \sum_{i=1}^{M} l_i P_i$

where $l_i = l(\underline{v}_i)$ = length of $\underline{v}_i$

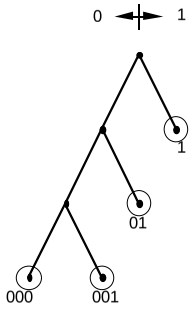
Visualize codebook with a binary tree

Example: Codebook {1, 01, 000, 001}

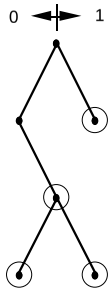
A tree has *nodes*, *branches*, *a root node, paths*, *terminal nodes, interior nodes, codewords assigned to some of the nodes.*

It's easy to recognized a prefix or nonprefix code.

Is {1, 01, 011, 010} a prefix code?



Can use the tree for decoding; e.g. store the codewords in a linked list and use the tree to guide decoding.



Unique Decodability (codes need to have this property)

Defn: A code is uniquely decodable (UD) if any finite sequence of its codewords corresponds to only one message sequence.

Equivalently, the concatenation of any finite set of codewords is different than the concatenation of some other finite set of codewords.

Equivalently, the concatenation of any given sequence of codewords cannot be parsed in a different way into codewords.

Codes d, e, f are not UD

UD codes also called *separable* or *uniquely decipherable*.

Tests for UD

- In general it can be hard to tell if a code is UD.
- Examples: {00, 001, 1010, 0101}
- Example :

  {010, 0001, 0110, 1100, 00011, 00110, 11110, 101011}
- There is a systematic procedure due to Sardinas and Patterson for determining if a code is UD in finitely many steps. See, *Information Theory,* R. Ash.

Defn: A code is ***prefix*** (prefix-free) if no codeword is the prefix of another.

- Prefix codes are UD, because the concatenation of any sequence of codewords can be parsed in only one way: If z is such a sequence of codewords, then it must be prefixed by some codeword, say v. It can't be prefixed by another codeword, say w, because then either v would prefix w or vice versa, both of which are impossible. Therefore, v is the only codeword that prefixes z. Now remove v from the beginning of z and repeat the argument. Continuing in this way will produce the one and only parsing of z into codewords.
- Codes a and b are prefix. The rest are not.
- Prefix codes are *instantaneously decodable*. (They are sometimes called instantaneous codes.).
- There are UD codes that are not prefix, for example, Code f. Notice that it is not instantaneously decodable.
- Some UD codes that are not prefix cannot be decoded unless and until we stop using them.

  Example: {00, 001, 1010, 0101}.

  This is UD since it is suffix free. However, there can be problems with UD codes that are not prefix. For example, how to decode the infinite sequence

  0 0 1 0 1 0 1 0 1 0 1 0 1 0 ...

From now on, we usually restrict attention to prefix codes, because they are easy to recognize, easy to design, easy to decode, instantaneously decodable, and because we will show there is no loss of performance.

Key Questions

How to design a prefix code to minimize rate for a given set of probabilities?

What is the resulting smallest possible average length, i.e. rate? Let L^* and R^* denote these quantities?

Basic idea: short codewords for large P_i's

Can't have all lengths small: (illustrate with tree)

What lengths can a prefix code have??

Answer given by the Kraft Inequality Theorem.

Kraft Inequality Theorem (ref. Chap 2, Section 2.7, Neuhoff & Forney)

Positive statement: If integers $l_1,...,l_M$ satisfy $\sum_{i=1}^{M} 2^{-l_i} \leq 1$, which is called the Kraft inequality, then there exists a prefix code with these lengths.

Negative statement: The lengths $l_1,...,l_M$ of any UD code (prefix or not) satisfy the Kraft inequality, i.e. $\sum_{i=1}^{M} 2^{-l_i} \leq 1$

Combined statement: There exists a UD code with lengths $l_1,...,l_M$ iff $\sum_{i=1}^{M} 2^{-l_i} \leq 1$.

Alternatively, there exists a prefix code with lengths $l_1,...,l_M$ iff $\sum_{i=1}^{M} 2^{-l_i} \leq 1$.

Corollary: For any UD code, there is a prefix code with the same lengths. Thus restricting to prefix codes entails no loss in performance.

Infinite alphabets: The Kraft inequality theorem also holds when $|A| = M = \infty$

Code Design Algorithms

For a given list of probabilities $\underline{P} = \{P_1,\ldots,P_2\}$ we want to find a prefix code with small average length. Equivalently we want to find a set of lengths that satisfy the Kraft inequality and have small average length. The basic idea is that highest probability symbols should have the shortest codewords. But how short should they be?

Shannon Codes (aka Shannon-Fano codes)

Basic idea: use lengths

$$l_i \cong -\log_2 P_i$$

Is this possible? Lengths need to be integers; try

$$l_i = \lceil -\log_2 P_i \rceil \qquad \text{(round up to be conservative)}$$

and check Kraft inequality:

$$\sum_i 2^{-l_i} = \sum_i 2^{-\lceil -\log_2 P_i \rceil} \le \sum_i 2^{-(-\log_2 P_i)} = \sum_i P_i = 1$$

Since Kraft inequality is satisfied, there are codes with these lengths.

These codes are called Shannon codes. The rate is

$$R_{Sh} = \sum_i P_i \lceil -\log_2 P_i \rceil$$

How good are Shannon codes?

A. Upper bound

$$R_{Sh} = \sum_i P_i \lceil -\log_2 P_i \rceil$$

$$< -\sum_i P_i \log_2 P_i + 1, \qquad \text{because } \lceil -\log_2 P_i \rceil < -\log_2 P_i + 1$$

B. Lower bound: For any prefix or UD code whatsoever

$$R \ge -\sum_i P_i \log_2 P_i$$

Derivation: show $R - (-\sum_i P_i \log_2 P_i) \ge 0$:

$$R - (-\sum_i P_i \log_2 P_i) = \sum_i P_i l_i + \sum_i P_i \log_2 P_i = \sum_i P_i (l_i + \log_2 P_i)$$

$$= -\sum_i P_i \log_2 \frac{2^{-l_i}}{P_i}$$

$$\ge -\sum_i P_i \left(\frac{2^{-l_i}}{P_i} - 1\right) \frac{1}{\ln 2} \qquad \text{because } \ln x \le x-1 \Rightarrow \log_2 x = \frac{\ln x}{\ln 2} \le \frac{x-1}{\ln 2}$$

$$= -\left(\sum_i 2^{-l_i} - \sum_i P_i\right) \frac{1}{\ln 2}$$

$$\ge -(1 - 1)\frac{1}{\ln 2} = 0 \qquad \text{because UD nature of code implies that its lengths satisfy the Kraft inequality}$$

C. A and B show that no code could have rate less than R_{sh}-1

Definitions:

$H = H(X) = H(\underline{P}) = -\sum_i P_i \log_2 P_i$ = *entropy* of X or $\underline{P}$

R^* = least rate of any prefix (or UD) code.

In summary, we've established a

Lossless Source Coding Theorem: For any set of probabilities,

$$H \le R^* \le R_{Sh} < H + 1 \ ,$$

That is, there exists a code (e.g. a Shannon code) with $R < H+1$, and every prefix code has $R \ge H$.

Notes:

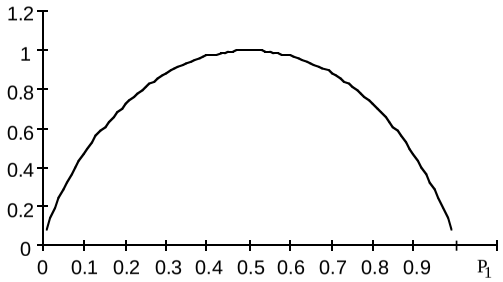
- Entropy is obviously an important quantity. It can be interpreted as the amount of randomness in X or the uncertainty in X.
- Notice the three alternative notations for entropy.
- The theorem doesn't actually find the best code or R^*. But we've sandwiched R^* between H and $H+1$.
- The Huffman design algorithm will enable us to find R^* but it won't tell us how it depends on the probabilities.
- We already know that Shannon codes are within 1 bit of optimal.

Examples:

- Binary random variable: $P = \{P_1, 1-P_1\}$,

  $H = -P_1 \log_2 P_1 - (1-P_1) \log_2 P_1$

  $\triangleq H(P_1)$



- 4-ary random variable: $P = \{.5, .25, .125, .125\}$

  $H = 1.75$, $R_{Sh} = 1.75$.

- More generally, whenever all probilities are powers of 2,

  $R_{Sh} = H$ (left as an excercise)

- English text: $M = 27$, $\log_2 27 = 4.75$, $H \cong 4.0$.

  Table of English letter probabilities given in Chap 2, N&F.

Properties of Entropy: $H = -\sum_i -P_i \log_2 P_i$

0. $0 \log_2 0 \triangleq 0$, because $\lim_{p \to 0} p \log_2 p = 0$

1. "Fundamental inequality" of information theory:

   $\ln p \le p-1$ and $\log_2 p = \frac{\ln p}{\ln 2} \le (p-1)\frac{1}{\ln 2}$

   Equality holds iff $p = 1$.

2. $H \ge 0$, and $H = 0$ iff $P_i = 1$ for some i.

   Proof: $H \ge 0$ because $-P_i \log_2 P_i \ge 0$ each i.

   $(H = 0) \Leftrightarrow (-P_i \log_2 P_i = 0$ each i$) \Leftrightarrow (P_i = 0$ or 1 each i$) \Leftrightarrow P_i = 1$ for some i.

3. If $|A| = M < \infty$, then $H \le \log_2 M$. $H = \log_2 M$ iff $P_i = \frac{1}{M}$ all i

   Proof: $\log_2 M - H = \sum_i P_i (\log_2 M + \log_2 P_i) = -\sum_i P_i \log_2 \frac{1}{MP_i}$

   $$\ge -\sum_i P_i \left(\frac{1}{MP_i} - 1\right)\frac{1}{\ln 2} = \left(-\sum_i \frac{1}{M} + \sum_i P_i\right)\frac{1}{\ln 2} = 0$$

   with equality iff $\frac{1}{MP_i} = 1$ for each i

4. If $|A| = \infty$, then H can be finite or infinite.

5. Suppose we replace P_1, P_2 by Q_1 and Q_2 where Q_1 and Q_2 are more nearly equiprobable. Then

   $H(P_1,P_2,P_3, \ldots) < H(Q_1,Q_2,P_3,\ldots)$

   This shows that entropy increases as the probabilities become more equal and decreases as they become more dissimilar.

   Proof: WLOG assume $P_1 > P_2$. Then $P_1 - \delta = Q_1 \ge Q_2 = P_2 + \delta$. Then

   $H(Q_1,Q_2,P_3,\ldots) - H(P_1,P_2,P_3, \ldots)$

   $$= -Q_1 \log_2 Q_1 - Q_2 \log_2 Q_2 + P_1 \log_2 P_1 + P_2 \log_2 P_2$$

   $$= -(P_1-\delta)\log_2(P_1-\delta) - (P_2+\delta)\log_2(P_2+\delta) + P_1 \log_2 P_1 + P_2 \log_2 P_2$$

   $$= -P_1 \log_2 \frac{P_1-\delta}{P_1} - P_2 \log_2 \frac{P_2+\delta}{P_2} + \delta \log_2 \frac{P_1-\delta}{P_2+\delta}$$

   $$\ge -P_1 \left(\frac{P_1-\delta}{P_1} - 1\right) - P_2 \left(\frac{P_2+\delta}{P_2} - 1\right) + \delta \log_2 \frac{P_1-\delta}{P_2+\delta}$$

   $$= \delta(P_1-P_2) + \delta \log_2 \frac{P_1-\delta}{P_2+\delta} > \delta \times 0 + \delta \log_2 1 = 0.$$

6. Properties 2,3,5 suggest that H may be interpreted as measure of "uncertainty" or "randomness" inherent in X or P. But its real quantitative significance comes from coding theorems, like the one stated previously that says that the least rate of any variable-length code is approximately H.

Huffman's Code Design Algorithm -- Huffman Codes

For a finite set of probabilities $\{P_1,\ldots,P_M\}$, Huffman's algorithm finds an optimal prefix code, i.e. one with smallest average length, i.e. one whose average length is R^*. The resulting codes are called Huffman codes.

There are usually many optimal codes. They can even have different sets of lengths.

See N&F Chap. 2 for a description of Huffman's algorithm and a proof that it generates optimal codes.

Other Code Designs

Binary code:

For $n \in A = \{0,\ldots,2^m-1\}$,

$b_m(n) = m$ bit binary representation of n,

$l(n) = m$

Codes for infinite alphabets:

$M = \infty$, $A = \{1, 2, 3, \ldots\}$, $P_1 > P_2 > \ldots$

Can't store an infinite number of codewords, so need a systematic encoding rule.

Examples of Systematic Codes for an Infinite Alphabet

	unary	gamma	delta	golumb b = 3	golumb b=4
1	0	0*	0	00	0 00
2	10	10 0	100 0	0 10	0 01
3	110	10 1	100 1	0 11	0 10
4	1110	110 00	101 00	10 0	0 11
5	11110	110 01	101 01	10 10	10 00
6	111110	110 10	101 10	10 11	10 01
7	1111110	110 11	101 11	110 0	10 10
8	11111110	1110 000	11000 000	110 10	10 11
9	111111110	1110 001	11000 001	110 11	110 00
10	11111111110	1110 010	11000 010	1110 0	110 01

***** denotes special case

Reference: Witten, Moffat, Bell, *Managing Gigabytes*

unary code: $u(n) = (1^{n-1}\ 0)$ (n-1 ones followed by one zero), $l(n) = n$

gamma code: $\gamma(n) = (u(m+1)\ b_m(r))$, $l(n) = m+1+m = 2\lfloor \log_2 n \rfloor + 1$

where $m = \lfloor \log_2 n \rfloor$ and $r = n - 2^m$, equiv'ly, $n = 2^m + r$, $0 \le r < 2^m$

Unary code specifies how many bits are used by binary code

Groups of 2^m integers have same prefix. This is similar to code used in JPEG, except that JPEG uses a better prefix than unary.

delta code: $\delta(n) = (\gamma(m+1)\ b_m(r))$, $l(n) = 2\lfloor \log_2 (\lfloor \log_2 n \rfloor + 1) \rfloor + 1 + \lfloor \log_2 n \rfloor$

where $m = \lfloor \log_2 n \rfloor$ and $r = n - 2^m$

Gamma code specifies how many bits are used by binary code.

Golumb code: Fix b, e.g. b = 3,

$$G_b(n) = (u(m+1)\ b^*_{\log b}(r)), \quad l(n) = m+2+\log_2 b$$

where $m = \lfloor \frac{n-1}{b} \rfloor$ and $r = n-1-mb$, equiv'ly, $n-1 = mb+r$, $0 \le r < b$,

and $b^*_{\log b}(r)$ is modified bin'y code, using $\lfloor \log_2 b \rfloor$ or $\lceil \log_2 b \rceil$ bits

Groups of b symbols have same prefix, followed by binary code.

Rice code: A Golumb code with b a power of 2.

Golumb & Rice are good for geometric (exponential) probabilities: $P_n = c\,Q^n$, n = 1, 2, ...