15-750:Algorithms in the Real World

Data Compression

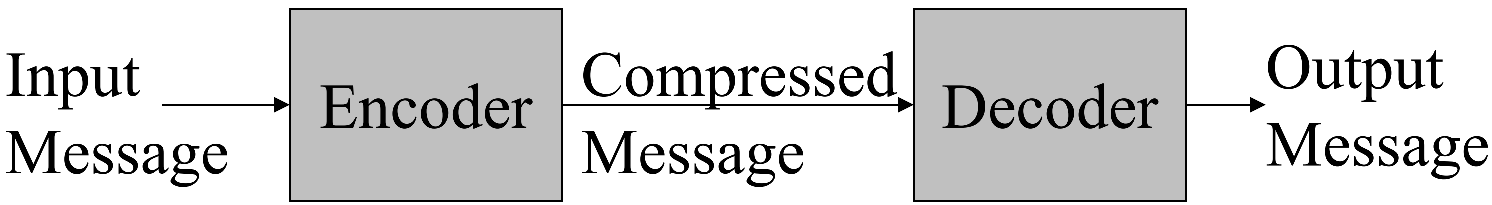
Compression in the Real World

Ubiquitous usage. Examples:

- Data storage: file systems, large-scale storage systems (e.g. cloud storage)
- Communication
- Media: Video, audio, images
- Data structures: Graphs, indexes
- Newer: Neural network compression

Encoding/Decoding

"**Message**" refers to the data to be compressed



The encoder and decoder need to understand common compressed format.

Lossless vs. Lossy

Lossless: Input message = Output message

Lossy: Input message ≈ Output message

Quality of Compression:

For Lossless?

Runtime vs. Compression vs. Generality

For Lossy?

Loss metric (in addition to above)

How much can we compress?

Q: Can we (lossless) compress any kind of messages?

No!

For lossless compression, assuming all input messages are valid, if one string is compressed, some other must expand.

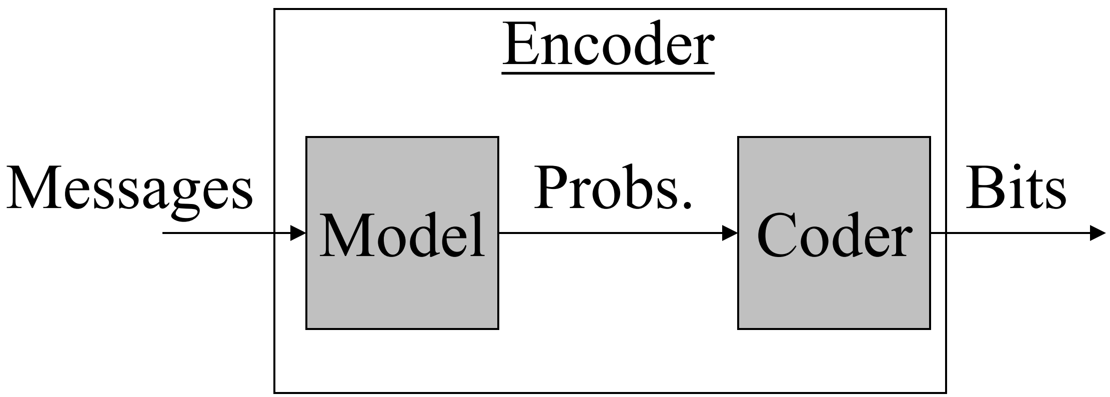
Q: So what we do need in order to be able to compress?

Can compress only if some messages are more likely than other.

That is, there needs to be **bias** in the probability distribution.

Model vs. Coder

To compress we need a bias on the probability of messages. The **model** determines this bias



Example models:

- Simple: Character counts, repeated strings
- Complex: Models of a human face

INFORMATION THEORY BASICS

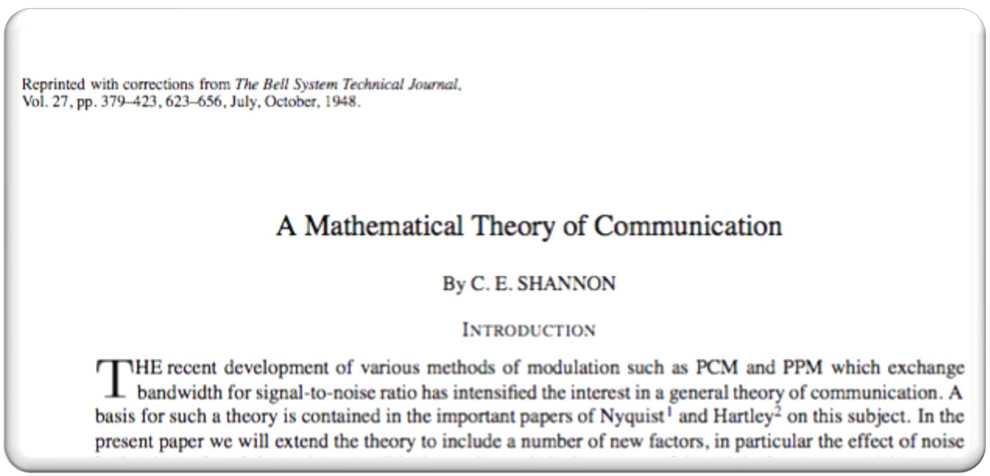
Information Theory

- Quantifies and investigates "information"
- Fundamental limits on representation and transmission of information
  - What's the minimum number of bits needed to **represent** data?
  - What's the minimum number of bits needed to **communicate** data?
  - What's the minimum number of bits needed to **secure** data?

Information Theory

Claude E. Shannon

- Landmark 1948 paper: mathematical framework
- Proposed and solved key questions
- Gave birth to information theory

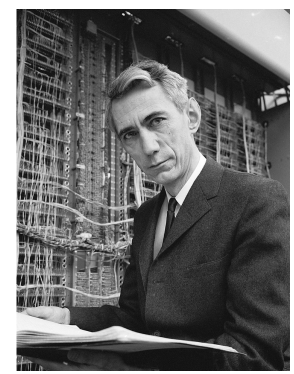

Reprinted with corrections from *The Bell System Technical Journal*,
Vol. 27, pp. 379–423, 623–656, July, October, 1948.

A Mathematical Theory of Communication

By C. E. SHANNON

INTRODUCTION

THE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist[1] and Hartley[2] on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise



Information Theory

In the context of compression:

An interface between modeling and coding

Entropy

– A measure of information content

Suppose a message can take **n** values from $S = \{s_1, \ldots, s_n\}$ with a probability distribution ***p(s)***.

One of the n values will be chosen.

“How much choice” is involved? OR

“How much information is needed to convey the value chosen?

Entropy

Q: Should it depend on the values $\{s_1, \ldots, s_n\}$?

(e.g., American names vs. European names)

No.

Q: Should it depend on p(s)?

Yes.

If P(s_1)=1 and rest are all 0?

No choice. Entropy = 0

More the bias lower the entropy

Entropy

Shannon (1948 paper) lists key properties that an entropy function should satisfy and *shows that "log" is the only function*.

Intuition for the log function:

- When p(s) is low, entropy should be high
- Suppose two independent messages are being picked then entropy should add up

Entropy

For a set of messages *S* with probability *p(s), s* $\in$ *S*, the **self information** of *s* is:

$$i(s) = \log \frac{1}{p(s)} = -\log p(s)$$

Measured in **bits if the log is base 2**.

Entropy is the weighted average of self information.

$$H(S) = \sum_{s \in S} p(s) \log \frac{1}{p(s)}$$

Entropy Example

Binary random variable (i.e., taking two values) with probability p and 1-p

Denoted as $H_2(p)$:

<board>

Highest entropy when equiprobable

(true for n >2 as well)

Entropy Example

$$p(S) = \{.25,.25,.25,.125,.125\}$$

$$H(S) = 3\times.25\log 4 + 2\times.125\log 8 = 2.25$$

$$p(S) = \{.5,.125,.125,.125,.125\}$$

$$H(S) = .5\log 2 + 4\times.125\log 8 = 2$$

$$p(S) = \{.75,.0625,.0625,.0625,.0625\}$$

$$H(S) = .75\log(4/3) + 4\times.0625\log 16 = 1.3$$

Conditional Entropy

Conditional entropy: Information content based on a context

The **conditional probability** *p(s|c)* is the probability of *s* in a context *c*.

The **conditional entropy** is the weighted average of the conditional self information

$$H(S \mid C) = \sum_{c \in C} \left(p(c) \sum_{s \in S} p(s \mid c) \log \frac{1}{p(s \mid c)} \right)$$

Types of “sources”

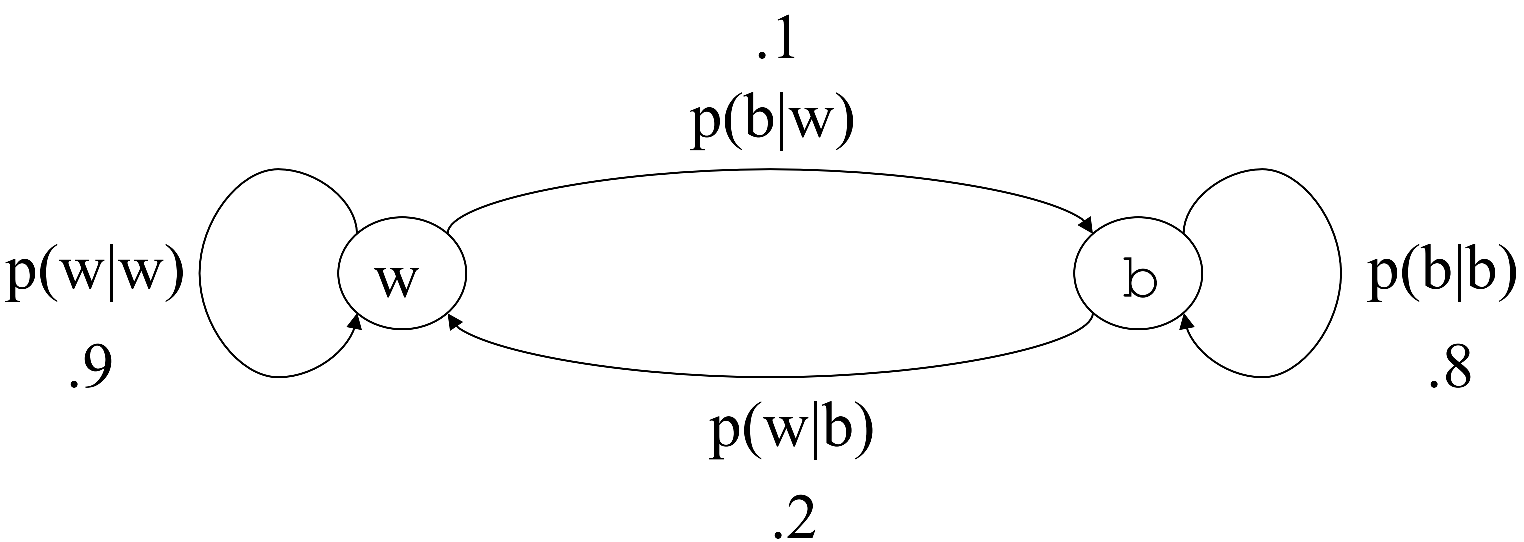
- Sources generate the messages (to be compressed)

- Sources can be modelled in multiple ways

- Independent and identically distributed (i.i.d) source
  - Prob. of each msg is independent of the previous msg

- Markov source
  - message sequence follows a Markov model (specifically Discrete Time Markov Chain, aka DTMC)

Example of a Markov Chain



Shannon's experiment

Asked people to predict the next character given the whole previous text. He used these as conditional probabilities to estimate the entropy of the English Language.

The number of guesses required for right answer:

# of guesses	1	2	3	4	5	> 5
Probability	.79	.08	.03	.02	.02	.05

From the experiment he predicted

H(English) = .6 - 1.3

PROBABILITY CODING

Assumptions and Definitions

Communication (or a file) is broken up into pieces called **messages**.

Each message come from a **message set** $S = \{s_1, \ldots, s_n\}$ with a **probability distribution** $p(s)$.

Code $C(s)$: A mapping from a message set to **codewords**, each of which is a string of bits

Message sequence: a sequence of messages

Variable length codes and Unique Decodability

A **variable length code** assigns a bit string (codeword) of variable length to every message value

e.g. `a = 1, b = 01, c = 101, d = 011`

What if you get the sequence of bits `1011` ?

Is it `aba, ca,` or, `ad`?

A **uniquely decodable code** is a variable length code in which bit strings can always be uniquely decomposed into its codewords.

Prefix Codes

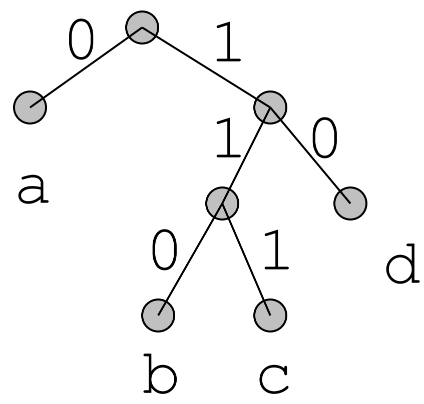
A **prefix code** is a variable length code in which no codeword is a prefix of another word.

e.g., a = 0, b = 110, c = 111, d = 10

All prefix codes are uniquely decodable

Prefix Codes: as a tree

Prefix codes can be viewed as a binary tree with 0s or 1s on the edges and message values at the leaves:



a = 0, b = 110, c = 111, d = 10

Average Length

For a code C with associated probabilities $p(c)$ the **average length** is defined as

$$l_a(C) = \sum_{c \in C} p(c) l(c)$$

$l(c)$ = length of the codeword c (a positive integer)

We say that a prefix code C is **optimal** if for all prefix codes C', $l_a(C) \leq l_a(C')$

Relationship to Entropy

Theorem (lower bound): For any probability distribution p(S) with associated uniquely decodable code C,

$$H(S) \leq l_a(C)$$

Theorem (upper bound): For any probability distribution p(S) with associated <u>optimal</u> prefix code C,

$$l_a(C) \leq H(S) + 1$$

Kraft McMillan Inequality

Theorem (Kraft-McMillan): For any uniquely decodable code C,

$$\sum_{c \in C} 2^{-l(c)} \leq 1$$

Conversely, for any set of lengths L such that $\sum_{l \in L} 2^{-l} \leq 1$

there is a prefix code C such that

$$l(c_i) = l_i (i = 1, \ldots, | L |)$$

Proof of the Upper Bound (Part 1)

Assign each message a length: $l(s) = \lceil \log(1/p(s)) \rceil$

We then have

$$\begin{aligned}\sum_{s \in S} 2^{-l(s)} &= \sum_{s \in S} 2^{-\lceil \log(1/p(s)) \rceil} \\ &\leq \sum_{s \in S} 2^{-\log(1/p(s))} \\ &= \sum_{s \in S} p(s) \\ &= 1\end{aligned}$$

So by the Kraft-McMillan inequality there is a prefix code with lengths *l(s)*.

Proof of the Upper Bound (Part 2)

Now we can calculate the average length given l(s)

$$
\begin{aligned}
l_a(S) &= \sum_{s \in S} p(s) l(s) \\
&= \sum_{s \in S} p(s) \cdot \left\lceil \log\left(1 / p(s)\right) \right\rceil \\
&\le \sum_{s \in S} p(s) \cdot (1 + \log(1 / p(s))) \\
&= 1 + \sum_{s \in S} p(s) \log(1 / p(s)) \\
&= 1 + H(S)
\end{aligned}
$$

Another property of optimal codes

Theorem: If C is an optimal prefix code for the probabilities $\{p_1, \ldots, p_n\}$, then $p_i > p_j$ implies $l(c_i) \leq l(c_j)$

Proof: (by contradiction)

Assume $l(c_i) > l(c_j)$. Consider switching codes c_i and c_j.

If l_a is the average length of the original code, the length of the new code is

$$\begin{aligned} l_a' &= l_a + p_j(l(c_i) - l(c_j)) + p_i(l(c_j) - l(c_i)) \\ &= l_a + (p_j - p_i)(l(c_i) - l(c_j)) \\ &< l_a \end{aligned}$$

This is a contradiction since l_a is not optimal

Huffman Codes

Invented by Huffman as a class assignment in 1950.

Used in many, if not most, compression algorithms

Properties:

- Generates optimal prefix codes
- Cheap to generate codes
- Cheap to encode and decode
- $l_a = H$ if probabilities are powers of 2

Huffman Codes

Huffman Algorithm:

Start with a forest of trees each consisting of a single vertex corresponding to a message *s* and with weight *p(s)*

Repeat until one tree left:

- Select two trees with minimum weight roots p_1 and p_2
- Join into single tree by adding root with weight $p_1 + p_2$

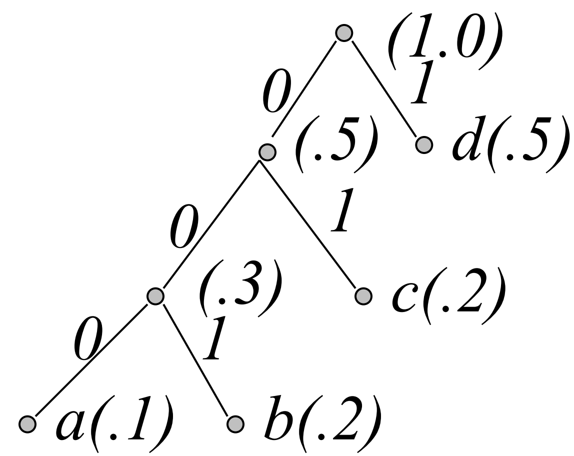
Example

$p(a) = .1,\ \ p(b) = .2,\ \ p(c\,) = .2,\ \ p(d) = .5$

$a(.1)$ $b(.2)$ $c(.2)$ $d(.5)$

Encoding and Decoding

Encoding: Start at leaf of Huffman tree and follow path to the root. Reverse order of bits and send.



$a=000, \ b=001, \ c=01, d=1$

Decoding: Start at root of Huffman tree and take branch for each bit received. When at leaf can output message and return to root.

Huffman codes are "optimal"

Theorem: The Huffman algorithm generates an optimal *prefix* code.

Proof outline:

Induction on the number of messages n.

Consider a message set S with n+1 messages

1. Can make it so that least probable messages of S are neighbors in the Huffman tree
2. Replace the two messages with one message with probability $p(m_1) + p(m_2)$ making S'
3. Show that if S' is optimal, then S is optimal
4. S' is optimal by induction

Problem with Huffman Coding

Consider a message with probability .999. The self information of this message is

$$-\log(.999) = .00144$$

If we were to send a 1000 such message we might hope to use 1000*.0014 = 1.44 bits.

Using Huffman codes we require at least one bit per message, so we would require 1000 bits.

Need to “blend” bits among message symbols!

Discrete or Blended

Discrete: each message is a fixed set of bits

- E.g., Huffman coding, Shannon-Fano coding

01001	11	0001	011
1	2	3	4

message: 1 2 3 4

Blended: bits can be "shared" among messages

- E.g., Arithmetic coding

010010111010

message: 1,2,3, and 4

Arithmetic Coding: Introduction

Allows “blending” of bits in a message sequence.

Can bound total bits required based on sum of self information (s_i):

$$l < 2 + \sum_{i=1}^{n} s_i$$

More expensive than Huffman coding

Used in many compression algorithms: E.g., JPEG/MPEG