A Mathematical Theory of Communication

Ben Eggers

Abstract

This paper defines information-theoretic entropy and proves some elementary results about it. Notably, we prove that given a few basic assumptions about a "reasonable" measure of information, there is exactly one function that satisfies them all. Then, as an application, we study the well-known Huffman compression algorithm.

Contents

1 Introduction

Information theory is a new field, but is rapidly becoming fundamentally important in fields as diverse as Computer Science, Linguistics, and Physics. In information theory, the basic unit is a called a *bit* (shorthand for *binary digit*) which represents a single yes-no decision. While there exist continuous versions of the theory as well, in this paper I will only be expositing discrete information theory. From now on, "information theory" is meant to denote "discrete noiseless information theory."

Sections 1.2 and 1.3 explain the information-theoretic model of communication, and Section 2 defines it formally, as well as defining the Huffman compression algorithm. Section 3 contains optimality results of entropy and analyzes the efficiency of Huffman compression as an application of the theory.

1.1 Historical Background

In 1948, Claude Shannon of Bell Labs published his seminal paper on information theory whose title shares this paper's. In it, he defines precisely the problem that information theory is meant to model and solve, and derives many important mathematical results. More importantly, his work paved the way for other information theory researchers to expand on his theory, giving us the mature and usable mathematics we have today. Shannon's paper did not come a moment too soon: this was an age in which the atomic bomb had just been developed, we were still a decade from the polio vaccine and Sputnik, and most pertinent, transistors were only just beginning to replace vacuum tubes in the design of digital machines (in fact, this is also thanks to Shannon). Information theory gave engineers, mathematicians, and scientists the necessary tools to analyze how well their machines were transmitting data to and from one another.

1.2 Problem Statement

Information theory seeks to study how information is transmitted between an *information source* and an *information receiver*. For example, a computer sending an email over the Internet could be viewed as an information source, with the receiving computer acting as information receiver. This example already illustrates an important point: there are not dedicated information sources and information receivers, but rather, most machines can act as either one. In addition to the information source and information receiver, there is an *encoding* mechanism, a *channel* on which the information is transmitted, and a *decoding* mechanism.

Often, the information being transmitted has some semantic meaning (otherwise, why bother to transmit it?). However, in information theory we do not worry about meaning, and simply worry about measuring the efficacy of our encoding. For example, a text message between friends might contain the string, "wot r u up 2 m8?". Information theory views such a string simply as one chosen out of the finite number of strings of length 15. Some readers may object to this. Indeed, there exist compression algorithms which exploit the semantic structure of the text which they are compressing, but such algorithms are not widely used or understood, and are beyond the scope of this paper. Another possible objection is that the string, "I'm excited!!!!!!!!!!!!" carries little

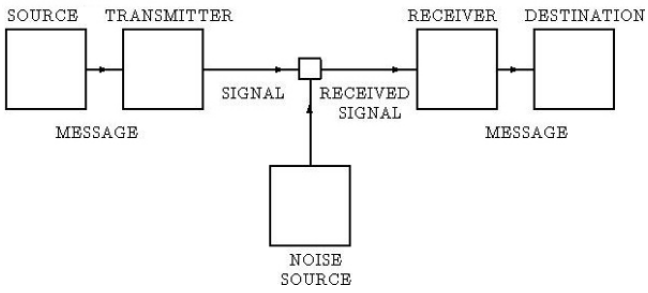

Figure 1: Model of information transmission. [1]

more information than, "I'm excited!!". The point, though, is that the former was chosen out of a much larger *universe* of possible strings of equal length, whereas latter had fewer possibilities. As stated before, the fundamental unit in information theory is a choice, usually a yes-no choice, of the form, "which character comes next in this string?", or more precisely, "given the frequencies that characters tend to appear, how much information did I just get by receiving this particular one?". These are the questions we seek to answer.

1.3 Information Sources as Markov Processes

In information theory, we are primarily concerned with the effectiveness of our encoding, not with the information source. However, having a model for the information source will be convenient as we develop our intuition and prove results. The primary model for an information source is a *Markov chain*. A Markov chain is a weighted directed graph where each vertex represents the state of the system, and each edge represents a transition to another state, weighted by the probability of making such a transition. Figure 2 is a Markov chain which has two states: E and A. If the chain is in state E, the probability of it transitioning to state E again is 0.3, and the probability of transitioning to state A is 0.7. If it is in state A, it transitions to state E with probability 0.4 and to state A with probability 0.6.

An important note: a sensible interpretation of Figure 2 as an information source is that when it transitions to a state, it produces the character labeling that state. We may also have Markov chains that produce characters on *transitions*, however. For example, a Markov chain to select an English vowel uniformly at random may have one state with 5 transitions back to itself, each with probability 0.2 and each representing a different choice of vowel. These two representations of Markov chains are isomorphic, so we will use whichever is most suited to our purposes.

It is convenient to view information sources as Markov processes, with states/transitions representing bits, characters, or even words or sentences in a natural language.

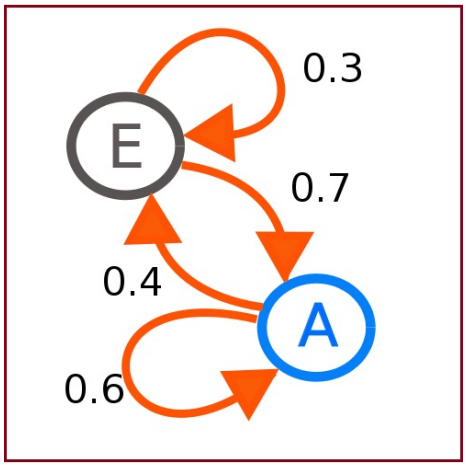

Figure 2: Markov Chain example [3]

2 Definitions

2.1 Entropy

Suppose we have n possible events, with the ith event having probability p_i such that $\Sigma_i p_i = 1$. We will use "entropy" to denote the total information encoded in a system, denoted by $H(p_1, ..., p_n)$. Alternatively, one can think of entropy as the amount of randomness present in a system. A system that is in state 1 with probability 0.99 and state 2 with probability 0.01 will not have much entropy, then, but a system which has two states with equal probability will have more. In other words, $H(0.99, 0.01) < H(\frac{1}{2}, \frac{1}{2})$. Let us formalize this notion a bit more.

There are three properties we would like our measure of information to have:

1. It should be a continuous function of the p_is.
2. If $p_1 = ... = p_n = \frac{1}{n}$, then H should be a monotonically increasing function of n. When there are more choices, there is more uncertainty, and our function should reflect that.
3. It should be possible to break a choice down into sub-choices, with the information of the final choice being the weighted sum of the information from the sub-choices. See Figure 3 for an example from Shannon's own paper.

We show in §3.1 that there is only one possible function suiting our needs, and it is given in the following definition:

Definition 1 *The **entropy** of a set of discrete events 1, 2, ..., n, with probabilities p_1, p_2, ..., p_n such that $\Sigma p_i = 1$ is:*
$H = -K\Sigma p_i log p_i$.
Here, K is arbitrary and can be thought of as the units.

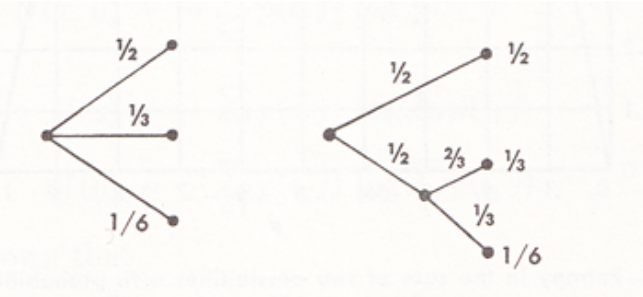

Figure 3: Breaking a choice into sub-choices. In this example, instead of making a three-way choice with probabilities $\frac{1}{2}, \frac{1}{3}, \frac{1}{6}$ (left), we first make a yes-no choice with probability $\frac{1}{2}$ for either option, then make another yes-no choice with probabilities $\frac{2}{3}, \frac{1}{3}$ (right). In this case, we want $H(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}) = H(\frac{1}{2}, \frac{1}{2}) + \frac{1}{2}H(\frac{2}{3}, \frac{1}{3})$. [1]

2.2 Huffman Coding

Huffman coding was invented in 1952 by David Huffman of MIT. In typical ASCII text encoding on a computer, each character uses 8 bits, giving $2^8 = 256$ possible characters, each with equal-length bit encodings. In ASCII, the character 'a' has the encoding 01100001, 'b' is 01100010, and so on. In Huffman coding, more frequent characters are given shorter encodings, while less frequent characters are given longer encodings. For example, the English character "e" is very common, and may be encoded by "001". The character "z" is very infrequent, however, so may be encoded by "010001110010". The key thing to notice is the *variable length*.

Informally, the Huffman Coding problem may be stated as follows: Given a set of symbols and their probabilities, construct a prefix-free binary code with minimum expected codeword length. By prefix-free, we mean that no code word for a symbol is a prefix of another code word. For concreteness, suppose 001 is the code word for some symbol s. Then for every symbol $t \neq s$, t's code word is not of the form $001c$ where c is any binary sequence.

The formal problem statement and algorithm are given below. The algorithm constructs an object known as a *Huffman Tree*. It is a binary (each node has 0, 1, or 2 children) tree in which each leaf node (node with no children) corresponds to one character. The code for a character can be obtained by tracing the path from the root of the tree to the character's leaf node, adding a '0' to the code if the path takes the left child of a node, and a '1' if the path takes the right child. For example, if the path to some character c goes left, then right, then left from the root, the code word for c would be 010. Figure 4 provides an example of a tree's construction and meaning.

It can be proven using standard computer science techniques that Huffman Coding produces the optimal code for symbol-by-symbol encoding, but such a proof lies beyond this paper. Here, we simply introduce Huffman Coding because it is amenable to analysis via our definition of entropy.

Algorithm 1: Huffman Code

Input: An alphabet $A = \{a_1, a_2, ..., a_n\}$ and a set of probabilities $P = \{p_1, ..., p_n\}$ such that $\Sigma_P = 1$.

Output: A code $C = \{c_1, ..., c_n\}$ where c_i is the code word for a_i and no c_i is the prefix of any c_j. There are many such codes, and we wish to find a (not necessarily unique) code which minimizes expected code word length $\Sigma[p_i \times length(c_i)]$.

Algorithm :

1 Create a tree node for each symbol. Tree node n_i corresponds to character a_i which has probability p_i. Call this set of nodes T.

2 While $|T| > 1$:

- Remove the two lowest-probability nodes from T. Call them n_i and n_j.
- Construct a new node, n_k, with children n_i and n_j. It's "probability" will be $p_i + p_j$.

3 The one remaining node in T is the root of the Huffman Tree.

3 Results

3.1 $H = -K\Sigma p_i log p_i$ is Unique

In this section, we show that $H = -K\Sigma p_i log p_i$ is the only function which satisfies the three desired properties stated earlier. The properties are reproduced here:

1. It should be a continuous function of the p_is.
2. If $p_1 = ... = p_n = \frac{1}{n}$, then H should be a monotonically increasing function of n. When there are more choices, there is more uncertainty, and our function should reflect that.
3. It is possible to break a choice down into sub-choices, with the information of the final choice being the weighted sum of the information from the sub-choices. See Figure 3 for an example from Shannon's own paper.

The following proof is borrowed almost directly from Shannon [1].

Let $A(n) = H(\frac{1}{n}, ..., \frac{1}{n})$. By condition (3) above, we can decompose a choice of s^m equally likely choices into a sequence of m choices from s equally likely elements. In other words, instead of just choosing an element from the s^m possible choices, we can construct a "choice tree" that is m levels deep, with each node on the $i \neq n$ level having s children. Therefore,

$$A(s^m) = mA(s)$$

For some other equally likely t^n events, we also have

$$A(t^n) = nA(t).$$

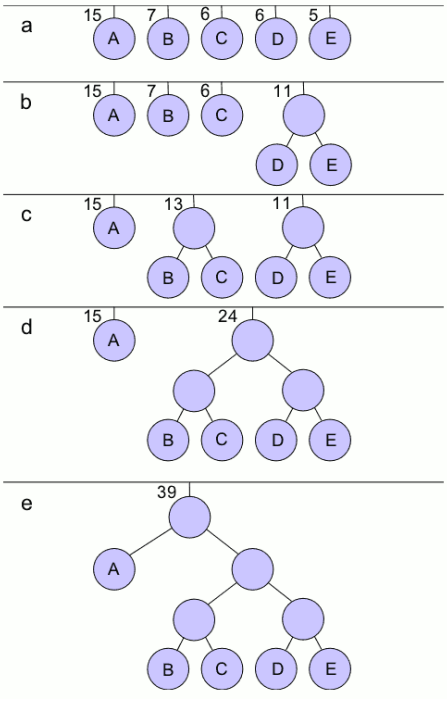

Figure 4: An example run of the Huffman Tree algorithm. We start with 5 characters. In this image, the numbers represent number of occurences in the text to be compressed, not probabilities. Probabilities can be obtained by normalizing the numbers so they sum to one. We start by taking the two lowest-frequency characters, 'D' and 'E', and creating a node with them as children. We continue this process until we have a tree with all the initial nodes. To get the code word for a character, we trace the path from the root to the node, adding a '0' to the code if we go left, and a '1' if we go right. In this example, the code for 'A' would be '0', the code for 'B' would be '100', etc. [4]

Now, for fixed s, t we take n arbitrarily large and find an m such that

$$s^m \leq t^n < s^{m+1}$$

By taking logarithms and dividing everything by $n\log s$,

$$\frac{m}{n} \leq \frac{logt}{logs} < \frac{m}{n} + \frac{1}{n}$$

Since n is arbitrarily large, this gives us

$$\left| \frac{m}{n} - \frac{logt}{logs} \right| < \epsilon \tag{1}$$

for any $\epsilon > 0$. We will revisit this inequality in a moment, but first must prove another useful inequality. With our same choices of s, t, n, m above, by the monotonicity condition we have

$$A(s^m) \leq A(t^n) < A(s^{m+1})$$

and

$$mA(s) \leq nA(t) < (m+1)A(s).$$

Dividing the second inequality by $nA(s)$, we obtain

$$\frac{m}{n} \leq \frac{A(t)}{A(s)} < \frac{m}{n} + \frac{1}{n}.$$

Once again, since n is arbitrarily large, we have

$$\left| \frac{m}{n} - \frac{A(t)}{A(s)} \right| < \epsilon \tag{2}$$

for any $\epsilon > 0$. Combining inequalities (1) and (2), we get our initial result:

$$\left| \frac{A(t)}{A(s)} - \frac{logt}{logs} \right| < 2\epsilon$$

and so

$$A(x) = Klogx \tag{3}$$

where K must be positive to satisfy the monotonicity condition. This shows that for n equally likely choices, the information content grows logarithmically in n. Now, we will generalize our entropy function to non-uniform probability distributions.

Suppose we want to make a choice from n possibilities, each with probability $p_i = \frac{n_i}{\Sigma n_i}$ where the n_i are integers. By condition (3), we can convert a choice from Σn_i possibilities into a choice from n possibilites with probabilities $p_1, ..., p_n$, then if the ith was chosen, a choice from n_i with equal probabilites. Putting this into math,

$$Klog(\Sigma n_i) = H(p_1, ..., p_n) + K\Sigma p_i log p_i.$$

Therefore

$$H(p_1, ..., p_n) = K[\Sigma p_i log \Sigma n_i - \Sigma p_i log n_i = -K\Sigma p_i log \frac{n_i}{\Sigma n_i} = -K\Sigma p_i log p_i$$

If the n_i are rational, we can use the same method. If they are irrational, they can be approximated arbitrarily closely by rational numbers since condition (1) states that the function need be continuous (and indeed, our function is). So $H = -K\Sigma p_i log p_i$ holds in general.

English letter	Frequency of use	English letter	Frequency of use
E	11.1607%	M	3.0129%
A	8.4966%	H	3.0034%
R	7.5809%	G	2.4705%
I	7.5448%	B	2.0720%
O	7.1635%	F	1.8121%
T	6.9509%	Y	1.7779%
N	6.6544%	W	1.2899%
S	5.7351%	K	1.1016%
L	5.4893%	V	1.0074%
C	4.5388%	X	0.2902%
U	3.6308%	Z	0.2722%
D	3.3844%	J	0.1965%
P	3.1671%	Q	0.1962%

Figure 5: Frequencies of English characters in an "average" piece of text. For more details, see [5]

3.2 Information-Theoretic Evaluation of Huffman Coding

4 Conclusion

In Section 1, we introduced the problem that information theory attempts to solve, and gave a brief explanation of its history. In Section 2, we defined information-theoretic entropy, and we saw that it was exactly the same formula as entropy in statistical mechanics, justifying our choice of name. In Section 2 we also defined the Huffman compression algorithm. In Section 3 we proved optimality results–our choice of entropy equation is the only one that suits our demands, and Huffman coding is the optimal byte-by-byte encoding scheme. In Section 4 we used our definition of entropy to analyze how well Huffman coding compresses an average English text, and found that it gives us a compression factor of approximately ??.

References

[1] Shannon, Claude E. (July–October 1948). "A Mathematical Theory of Communication". Bell System Technical Journal 27 (3): 379–423. doi:10.1002/j.1538-7305.1948.tb01338.x.

[2] Huffman, D. (1952). "A Method for the Construction of Minimum-Redundancy Codes". Proceedings of the IRE 40 (9): 1098–1101. doi:10.1109/JRPROC.1952.273898

[3] "Markovkate 01" Joxemai4 - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons

[4] http://commons.wikimedia.org/wiki/File:HuffmanCodeAlg.png Licensed under the GNU Free Documentation License

[5] Method for inputting words in an electronic appliance with buttons of inputting words Eun, I. http://www.google.com/patents/WO2007046567A1?cl=en 2007 Google Patents WO Patent App. PCT/KR2005/003,525