CHAPTER 2
Information, Entropy, and the Motivation for Source Codes

The theory of *information* developed by Claude Shannon (MIT SM '37 & PhD '40) in the late 1940s is one of the most impactful ideas of the past century, and has changed the theory and practice of many fields of technology. The development of communication systems and networks has benefited greatly from Shannon's work. In this chapter, we will first develop the intution behind information and formally define it as a mathematical quantity, then connect it to a related property of data sources, *entropy*.

These notions guide us to efficiently *compress* a data source before communicating (or storing) it, and recovering the original data without distortion at the receiver. A key underlying idea here is *coding*, or more precisely, *source coding*, which takes each "symbol" being produced by any source of data and associates it with a *codeword*, while achieving several desirable properties. (A message may be thought of as a sequence of symbols in some alphabet.) This mapping between input symbols and codewords is called a **code**. Our focus will be on *lossless* source coding techniques, where the recipient of any uncorrupted message can recover the original message exactly (we deal with corrupted messages in later chapters).

2.1 Information and Entropy

One of Shannon's brilliant insights, building on earlier work by Hartley, was to realize that *regardless of the application and the semantics of the messages involved*, a general definition of information is possible. When one abstracts away the details of an application, the task of communicating something between two parties, S and R, boils down to S picking one of several (possibly infinite) messages and sending that message to R. Let's take the simplest example, when a sender wishes to send one of two messages—for concreteness, let's say that the message is to say which way the British are coming:

- "1" if by land.
- "2" if by sea.

(Had the sender been a computer scientist, the encoding would have likely been "0" if by land and "1" if by sea!)

Let's say we have no prior knowledge of how the British might come, so each of these choices (messages) is equally probable. In this case, the amount of information conveyed by the sender specifying the choice is **1 bit**. Intuitively, that bit, which can take on one of two values, can be used to *encode* the particular choice. If we have to communicate a sequence of such independent events, say 1000 such events, we can encode the outcome using 1000 bits of information, each of which specifies the outcome of an associated event.

On the other hand, suppose we somehow knew that the British were far more likely to come by land than by sea (say, because there is a severe storm forecast). Then, if the message in fact says that the British are coming by sea, much more information is being conveyed than if the message said that that they were coming by land. To take another example, far more information is conveyed by my telling you that the temperature in Boston on a January day is 75°F, than if I told you that the temperature is 32°F!

The conclusion you should draw from these examples is that any quantification of "information" about an event should depend on the *probability* of the event. The greater the probability of an event, the smaller the information associated with knowing that the event has occurred. It is important to note that one does not need to use any semantics about the message to quantify information; only the probabilities of the different outcomes matter.

■ 2.1.1 Definition of information

Using such intuition, Hartley proposed the following definition of the information associated with an event whose probability of occurrence is p:

$$I \equiv \log(1/p) = -\log(p). \tag{2.1}$$

This definition satisfies the basic requirement that it is a decreasing function of p. But so do an infinite number of other functions, so what is the intuition behind using the logarithm to define information? And what is the base of the logarithm?

The second question is easy to address: you can use any base, because $\log_a(1/p) = \log_b(1/p)/\log_b a$, for any two bases a and b. Following Shannon's convention, *we will use base 2*,[1] in which case the unit of information is called a **bit**.[2]

The answer to the first question, why the logarithmic function, is that the resulting definition has several elegant resulting properties, and it is the simplest function that provides these properties. One of these properties is **additivity**. If you have two independent events (i.e., events that have nothing to do with each other), then the probability that they both occur is equal to the *product* of the probabilities with which they each occur. What we would like is for the corresponding information to *add up*. For instance, the event that it rained in Seattle yesterday and the event that the number of students enrolled in 6.02 exceeds 150 are independent, and if I am told something about both events, the amount of information I now have should be the sum of the information in being told individually of the occurrence of the two events.

[1] And we won't mention the base; if you see a $\log$ in this chapter, it will be to base 2 unless we mention otherwise.

[2] If we were to use base 10, the unit would be *Hartleys*, and if we were to use the natural log, base e, it would be *nats*, but no one uses those units in practice.

The logarithmic definition provides us with the desired additivity because, given two independent events A and B with probabilities p_A and p_B,

$$I_A + I_B = \log(1/p_A) + \log(1/p_B) = \log \frac{1}{p_A p_B} = \log \frac{1}{P(A \text{ and } B)}.$$

■ 2.1.2 Examples

Suppose that we're faced with N equally probable choices. What is the information received when I tell you which of the N choices occurred? Because the probability of each choice is $1/N$, the information is $\log_2(1/(1/N)) = \log_2 N$ bits.

Now suppose there are initially N equally probable and mutually exclusive choices, and I tell you something that narrows the possibilities down to one of M choices from this set of N. How much information have I given you about the choice?

Because the probability of the associated event is M/N, the information you have received is $\log_2(1/(M/N)) = \log_2(N/M)$ bits. (Note that when $M = 1$, we get the expected answer of $\log_2 N$ bits.)
Some examples may help crystallize this concept:

One flip of a fair coin

Before the flip, there are two equally probable choices: heads or tails. After the flip, we've narrowed it down to one choice. Amount of information = $\log_2(2/1) = 1$ bit.

Simple roll of two dice

Each die has six faces, so in the roll of two dice there are 36 possible combinations for the outcome. Amount of information = $\log_2(36/1) = 5.2$ bits.

Learning that a randomly chosen decimal digit is even

There are ten decimal digits; five of them are even (0, 2, 4, 6, 8). Amount of information = $\log_2(10/5) = 1$ bit.

Learning that a randomly chosen decimal digit ≥ 5

Five of the ten decimal digits are greater than or equal to 5. Amount of information = $\log_2(10/5) = 1$ bit.

Learning that a randomly chosen decimal digit is a multiple of 3

Four of the ten decimal digits are multiples of 3 (0, 3, 6, 9). Amount of information = $\log_2(10/4) = 1.322$ bits.

Learning that a randomly chosen decimal digit is even, ≥ 5, and a multiple of 3

Only one of the decimal digits, 6, meets all three criteria. Amount of information = $\log_2(10/1) = 3.322$ bits.

Learning that a randomly chosen decimal digit is a prime

Four of the ten decimal digits are primes—2, 3, 5, and 7. Amount of information = $\log_2(10/4) = 1.322$ bits.

Learning that a randomly chosen decimal digit is even and prime

Only one of the decimal digits, 2, meets both criteria. Amount of information = $\log_2(10/1) = 3.322$ bits.

To summarize: more information is received when learning of the occurrence of an unlikely event (small p) than learning of the occurrence of a more likely event (large p). The information learned from the occurrence of an event of probability p is defined to be $\log(1/p)$.

2.1.3 Entropy

Now that we know how to measure the information contained in a given event, we can quantify the *expected information* in a set of possible outcomes or mutually exclusive events. Specifically, if an event i occurs with probability $p_i, 1 \leq i \leq N$ out of a set of N mutually exclusive events, then the average or expected information is given by

$$H(p_1, p_2, \ldots p_N) = \sum_{i=1}^{N} p_i \log(1/p_i). \tag{2.2}$$

H is also called the **entropy** (or *Shannon entropy*) of the probability distribution. Like information, it is also measured in bits. It is simply the sum of several terms, each of which is the information of a given event weighted by the probability of that event occurring. It is often useful to think of the entropy as *the average or expected* **uncertainty** *associated with this set of events*.

In the important special case of two *mutually exclusive* events (i.e., exactly one of the two events can occur), occuring with probabilities p and $1-p$, respectively, the entropy

$$H(p, 1-p) = -p \log p - (1-p) \log(1-p). \tag{2.3}$$

We will be lazy and refer to this special case, $H(p, 1-p)$ as simply $H(p)$.

This entropy as a function of p is plotted in Figure 2-1. It is symmetric about $p = 1/2$, with its maximum value of 1 bit occuring when $p = 1/2$. Note that $H(0) = H(1) = 0$; although $\log(1/p) \to \infty$ as $p \to 0, \lim_{p \to 0} p \log(1/p) \to 0$.

It is easy to verify that the expression for H from Equation (2.2) is always non-negative. Moreover, $H(p_1, p_2, \ldots p_N) \leq \log N$ always.

2.2 Source Codes

We now turn to the problem of *source coding*, i.e., taking a set of messages that need to be sent from a sender and *encoding* them in a way that is efficient. The notions of information and entropy will be fundamentally important in this effort.

Many messages have an obvious encoding, e.g., an ASCII text file consists of sequence of individual characters, each of which is independently encoded as a separate byte. There are other such encodings: images as a raster of color pixels (e.g., 8 bits each of red, green and blue intensity), sounds as a sequence of samples of the time-domain audio waveform, etc. These encodings are ubiquitously produced and consumed by our computer's peripherals—characters typed on the keyboard, pixels received from a digital camera or sent to a display, and digitized sound samples output to the computer's audio chip.

All these encodings involve a sequence of fixed-length symbols, each of which can be easily manipulated independently. For example, to find the 42^{nd} character in the file, one

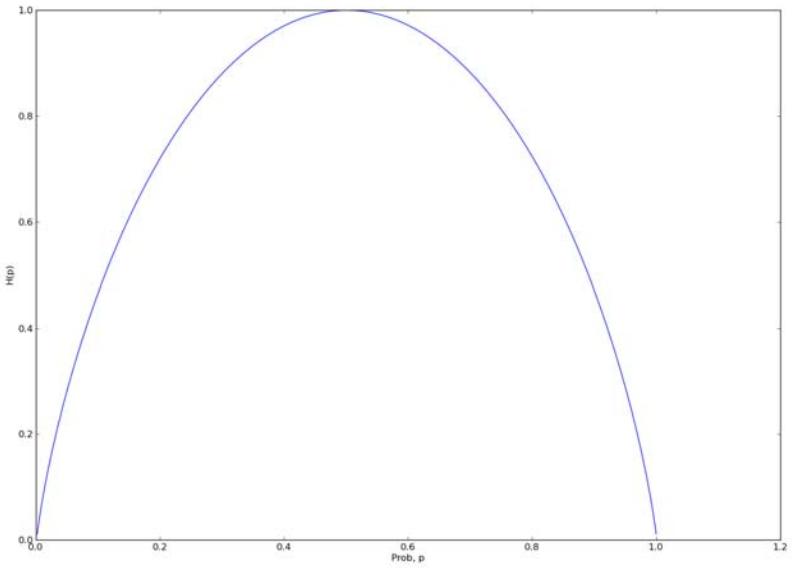

Figure 2-1: $H(p)$ **as a function of** p**, maximum when** $p = 1/2$**.**

just looks at the 42^{nd} byte and interprets those 8 bits as an ASCII character. A text file containing 1000 characters takes 8000 bits to store. If the text file were HTML to be sent over the network in response to an HTTP request, it would be natural to send the 1000 bytes (8000 bits) exactly as they appear in the file.

But let's think about how we might compress the file and send fewer than 8000 bits. If the file contained English text, we'd expect that the letter e would occur more frequently than, say, the letter x. This observation suggests that if we encoded e for transmission using *fewer* than 8 bits—and, as a trade-off, had to encode less common characters, like x, using more than 8 bits—we'd expect the encoded message to be shorter *on average* than the original method. So, for example, we might choose the bit sequence 00 to represent e and the code 100111100 to represent x.

This intuition is consistent with the definition of the amount of information: commonly occurring symbols have a higher p_i and thus convey less information, so we need fewer bits to encode such symbols. Similarly, infrequently occurring symbols like x have a lower p_i and thus convey more information, so we'll use more bits when encoding such symbols. This intuition helps meet our goal of matching the size of the transmitted data to the information content of the message.

The mapping of information we wish to transmit or store into bit sequences is referred to as a **code**. Two examples of codes (fixed-length and variable-length) are shown in Figure 2-2, mapping different grades to bit sequences in one-to-one fashion. The fixed-length code is straightforward, but the variable-length code is not arbitrary, and has been carefully designed, as we will soon learn. Each bit sequence in the code is called a **codeword**.

When the mapping is performed at the source of the data, generally for the purpose of *compressing* the data (ideally, to match the expected number of bits to the underlying

entropy), the resulting mapping is called a **source code**. Source codes are distinct from **channel codes** we will study in later chapters. Source codes *remove redundancy* and compress the data, while channel codes *add redundancy* in a controlled way to improve the error resilience of the data in the face of bit errors and erasures caused by imperfect communication channels. This chapter and the next are about source codes.

This insight about encoding common symbols (such as the letter e) more succinctly than uncommon symbols can be generalized to produce a strategy for *variable-length codes*:

> Send commonly occurring symbols using shorter codewords (fewer bits), and send infrequently occurring symbols using longer codewords (more bits).

We'd expect that, on average, encoding the message with a variable-length code would take fewer bits than the original fixed-length encoding. Of course, if the message were all x's the variable-length encoding would be longer, but our encoding scheme is designed to optimize the expected case, not the worst case.

Here's a simple example: suppose we had to design a system to send messages containing 1000 6.02 grades of A, B, C and D (MIT students rarely, if ever, get an F in 6.02 $\ddot\smile$). Examining past messages, we find that each of the four grades occurs with the probabilities shown in Figure 2-2.

Grade	Probability	Fixed-length Code	Variable-length Code
A	$1/3$	00	10
B	$1/2$	01	0
C	$1/12$	10	110
D	$1/12$	11	111

Figure 2-2: Possible grades shown with probabilities, fixed- and variable-length encodings

With four possible choices for each grade, if we use the fixed-length encoding, we need 2 bits to encode a grade, for a total transmission length of 2000 bits when sending 1000 grades.

Fixed-length encoding for $BCBAAB$: 01 10 01 00 00 01 (12 bits)

With a fixed-length code, the size of the transmission doesn't depend on the actual message—sending 1000 grades always takes exactly 2000 bits.

Decoding a message sent with the fixed-length code is straightforward: take each pair of received bits and look them up in the table above to determine the corresponding grade. Note that it's possible to determine, say, the 42^{nd} grade without decoding any other of the grades—just look at the 42^{nd} pair of bits.

Using the variable-length code, the number of bits needed for transmitting 1000 grades depends on the grades.

Variable-length encoding for $BCBAAB$: 0 110 0 10 10 0 (10 bits)

If the grades were all B, the transmission would take only 1000 bits; if they were all C's and D's, the transmission would take 3000 bits. But we can use the grade probabilities

given in Figure 2-2 to compute the *expected length of a transmission* as

$$1000[(\frac{1}{3})(2) + (\frac{1}{2})(1) + (\frac{1}{12})(3) + (\frac{1}{12})(3)] = 1000[1\frac{2}{3}] = 1666.7 \text{ bits}$$

So, on average, using the variable-length code would shorten the transmission of 1000 grades by 333 bits, a savings of about 17%. Note that to determine, say, the 42^{nd} grade, we would need to decode the first 41 grades to determine where in the encoded message the 42^{nd} grade appears.

Using variable-length codes looks like a good approach if we want to send fewer bits on average, but preserve all the information in the original message. On the downside, we give up the ability to access an arbitrary message symbol without first decoding the message up to that point.

One obvious question to ask about a particular variable-length code: is it the best encoding possible? Might there be a different variable-length code that could do a better job, i.e., produce even shorter messages on average? How short can the messages be on the average? We turn to this question next.

■ 2.3 How Much Compression Is Possible?

Ideally we'd like to design our compression algorithm to produce as few bits as possible: just enough bits to represent the information in the message, but no more. Ideally, we will be able to use no more bits than the amount of information, as defined in Section 2.1, contained in the message, at least on average.

Specifically, the entropy, defined by Equation (2.2), tells us the expected amount of information in a message, when the message is drawn from a set of possible messages, each occurring with some probability. The entropy is a lower bound on the amount of information that must be sent, on average, when transmitting data about a particular choice.

What happens if we violate this lower bound, i.e., we send fewer bits on average than called for by Equation (2.2)? In this case the receiver will not have sufficient information and there will be some remaining ambiguity—exactly what ambiguity depends on the encoding, but to construct a code of fewer than the required number of bits, some of the choices must have been mapped into the same encoding. Thus, when the recipient receives one of the overloaded encodings, it will not have enough information to unambiguously determine which of the choices actually occurred.

Equation (2.2) answers our question about how much compression is possible by giving us a lower bound on the number of bits that must be sent to resolve all ambiguities at the recipient. Reprising the example from Figure 2-2, we can update the figure using Equation (2.1).

Using Equation (2.2) we can compute the expected information content when learning of a particular grade:

$$\sum_{i=1}^{N} p_i \log_2(\frac{1}{p_i}) = (\frac{1}{3})(1.58) + (\frac{1}{2})(1) + (\frac{1}{12})(3.58) + (\frac{1}{12})(3.58) = 1.626 \text{ bits}$$

So encoding a sequence of 1000 grades requires transmitting 1626 bits on the average. The

Grade	p_i	$log_2(1/p_i)$
A	$1/3$	1.58 bits
B	$1/2$	1 bit
C	$1/12$	3.58 bits
D	$1/12$	3.58 bits

Figure 2-3: Possible grades shown with probabilities and information content.

variable-length code given in Figure 2-2 encodes 1000 grades using 1667 bits on the average, and so doesn't achieve the maximum possible compression. It turns out the example code does as well as possible when encoding one grade at a time. To get closer to the lower bound, we would need to encode sequences of grades—more on this idea below.

Finding a good code—one where the length of the encoded message matches the information content (i.e., the entropy)—is challenging and one often has to think "outside the box". For example, consider transmitting the results of 1000 flips of an unfair coin where the probability of a head is given by p_H. The expected information content in an unfair coin flip can be computed using Equation (2.3):

$$p_H \log_2(1/p_H) + (1 - p_H) \log_2(1/(1 - p_H))$$

For $p_H = 0.999$, this entropy evaluates to .0114. Can you think of a way to encode 1000 unfair coin flips using, on average, just 11.4 bits? The recipient of the encoded message must be able to tell for each of the 1000 flips which were heads and which were tails. Hint: with a budget of just 11 bits, one obviously can't encode each flip separately!

In fact, some effective codes leverage the context in which the encoded message is being sent. For example, if the recipient is expecting to receive a Shakespeare sonnet, then it's possible to encode the message using just 8 bits if one knows that there are only 154 Shakespeare sonnets. That is, if the sender and receiver both know the sonnets, and the sender just wishes to tell the receiver which sonnet to read or listen to, he can do that using a very small number of bits, just $\log_2 154$ bits if all the sonnets are equi-probable!

■ 2.4 Why Compression?

There are several reasons for using compression:

- Shorter messages take less time to transmit and so the complete message arrives more quickly at the recipient. This is good for both the sender and recipient since it frees up their network capacity for other purposes and reduces their network charges. For high-volume senders of data (such as Google, say), the impact of sending half as many bytes is economically significant.

- Using network resources sparingly is good for *all* the users who must share the internal resources (packet queues and links) of the network. Fewer resources per message means more messages can be accommodated within the network's resource constraints.

- Over error-prone links with non-negligible bit error rates, compressing messages before they are channel-coded using error-correcting codes can help improve throughput because all the redundancy in the message can be designed in to improve error resilience, after removing any other redundancies in the original message. It is better to design in redundancy with the explicit goal of correcting bit errors, rather than rely on whatever sub-optimal redundancies happen to exist in the original message.

Compression is traditionally thought of as an *end-to-end function*, applied as part of the application-layer protocol. For instance, one might use lossless compression between a web server and browser to reduce the number of bits sent when transferring a collection of web pages. As another example, one might use a compressed image format such as JPEG to transmit images, or a format like MPEG to transmit video. However, one may also apply compression at the link layer to reduce the number of transmitted bits and eliminate redundant bits (before possibly applying an error-correcting code over the link). When applied at the link layer, compression only makes sense if the data is inherently compressible, which means it cannot already be compressed and must have enough redundancy to extract compression gains.

The next chapter describes two compression (source coding) schemes: Huffman Codes and Lempel-Ziv-Welch (LZW) compression.

■ Acknowledgments

Thanks to Yury Polyanskiy for several useful comments and suggestions.

■ Exercises

1. Several people at a party are trying to guess a 3-bit binary number. Alice is told that the number is odd; Bob is told that it is not a multiple of 3 (i.e., not 0, 3, or 6); Charlie is told that the number contains exactly two 1's; and Deb is given all three of these clues. How much information (in bits) did each player get about the number?

2. After careful data collection, Alyssa P. Hacker observes that the probability of "HIGH" or "LOW" traffic on Storrow Drive is given by the following table:

	HIGH traffic level	LOW traffic level
If the Red Sox are playing	P(HIGH traffic) = 0.999	P(LOW traffic) = 0.001
If the Red Sox are not playing	P(HIGH traffic) = 0.25	P(LOW traffic) = 0.75

   (a) If it is known that the Red Sox are playing, then how much information in bits is conveyed by the statement that the traffic level is LOW. Give your answer as a mathematical expression.

   (b) Suppose it is known that the Red Sox are **not** playing. What is the entropy of the corresponding probability distribution of traffic? Give your answer as a mathematical expression.

3. X is an unknown 4-bit binary number picked uniformly at random from the set of all possible 4-bit numbers. You are given another 4-bit binary number, Y, and told

that X (the unknown number) and Y (the number you know) differ in precisely two positions. How many bits of information about X have you been given?

4. In Blackjack the dealer starts by dealing 2 cards each to himself and his opponent: one face down, one face up. After you look at your face-down card, you know a total of three cards. Assuming this was the first hand played from a new deck, how many bits of information do you have about the dealer's face down card after having seen three cards?

5. The following table shows the undergraduate and MEng enrollments for the School of Engineering:

Course (Department)	# of students	% of total
I (Civil & Env.)	121	7%
II (Mech. Eng.)	389	23%
III (Mat. Sci.)	127	7%
VI (EECS)	645	38%
X (Chem. Eng.)	237	13%
XVI (Aero & Astro)	198	12%
Total	1717	100%

   (a) When you learn a randomly chosen engineering student's department you get some number of bits of information. For which student department do you get the least amount of information?

   (b) **After studying Huffman codes in the next chapter,** design a Huffman code to encode the departments of randomly chosen groups of students. Show your Huffman tree and give the code for each course.

   (c) If your code is used to send messages containing only the encodings of the departments for each student in groups of 100 randomly chosen students, what is the average length of such messages?

6. You're playing an online card game that uses a deck of 100 cards containing 3 Aces, 7 Kings, 25 Queens, 31 Jacks and 34 Tens. In each round of the game the cards are shuffled, you make a bet about what type of card will be drawn, then a single card is drawn and the winners are paid off. The drawn card is reinserted into the deck before the next round begins.

   (a) How much information do you receive when told that a Queen has been drawn during the current round?

   (b) Give a numeric expression for the information content received when learning about the outcome of a round.

   (c) **After you learn about Huffman codes in the next chapter,** construct a variable-length Huffman encoding that minimizes the length of messages that report the outcome of a sequence of rounds. The outcome of a single round is encoded as A (ace), K (king), Q (queen), J (jack) or X (ten). Specify your encoding for each of A, K, Q, J and X.

(d) **Again, after studying Huffman codes,** use your code from part (c) to calculate the expected length of a message reporting the outcome of 1000 rounds (i.e., a message that contains 1000 symbols)?

(e) The Nevada Gaming Commission regularly receives messages in which the outcome for each round is encoded using the symbols A, K, Q, J, and X. They discover that a large number of messages describing the outcome of 1000 rounds (i.e., messages with 1000 symbols) can be compressed by the LZW algorithm into files each containing 43 bytes in total. They decide to issue an indictment for running a crooked game. Why did the Commission issue the indictment?

7. Consider messages made up entirely of vowels (A, E, I, O, U). Here's a table of probabilities for each of the vowels:

l	p_l	$\log_2(1/p_l)$	$p_l \log_2(1/p_l)$
A	0.22	2.18	0.48
E	0.34	1.55	0.53
I	0.17	2.57	0.43
O	0.19	2.40	0.46
U	0.08	3.64	0.29
Totals	1.00	12.34	2.19

(a) Give an expression for the number of bits of information you receive when learning that a particular vowel is either I or U.

(b) **After studying Huffman codes in the next chapter,** use Huffman's algorithm to construct a variable-length code assuming that each vowel is encoded individually. Draw a diagram of the Huffman tree and give the encoding for each of the vowels.

(c) Using your code from part (B) above, give an expression for the expected length in bits of an encoded message transmitting 100 vowels.

(d) Ben Bitdiddle spends all night working on a more complicated encoding algorithm and sends you email claiming that using his code the expected length in bits of an encoded message transmitting 100 vowels is 197 bits. Would you pay good money for his implementation?