

NOTES ON SHANNON'S INFORMATION THEORY

KARL PETERSEN

We will learn here (1) that *entropy* is a measure of the fundamental information content of messages, giving the minimal number of binary bits per symbol needed to encode the source; (2) that information sources can be *compressed* via block coding, so that the number of binary digits in the recoded message per symbol in the original message is very near the entropy; and (3) that even if messages are going to be altered by random noise, they can first be encoded in such a way that the original message can be recovered with an arbitrarily low probability of error. These startling statements were discovered in the late 1940's by Claude Shannon, and they enable many of the technological marvels we use today: computers, CD's and DVD's, telephones, and so on. This is pretty much a retelling of the account in [1].

1. INFORMATION SOURCES

The mathematical study of information processing begins with the concept of an *information source*. In our everyday experience one of our most familiar sources of information is the natural (human) language which we may be hearing or reading at any particular time. The sources of mathematical information theory do include idealized models of natural languages, but they are not the only starting point for analyzing such languages rigorously. The field of *formal languages* seeks to understand the basic principles governing human languages, computer languages, and even structures in chemistry, biology, and the physics of materials, by carefully analyzing the properties of abstract languages.

Let D be a finite alphabet. The elements of D will be called *symbols* or *letters*. Frequently $D = \{0, 1, \dots, d-1\}$ for some $d \in \mathbb{N}$. The symbol D^* denotes the set of all *finite strings* on the symbols in D , including the empty string, ϵ .

Definition 1.1. A *formal language* is any set of finite strings on a finite alphabet, that is, any $\mathcal{L} \subset D^*$ for any finite alphabet D .

This definition is very very broad, It's useful, because it is so inclusive. But it's not too useful, since it includes so many languages of so many different kinds. Analysis can begin when we consider particular examples of languages and languages of particular kinds.

Example 1.1. Let $D = \{0, 1\}$ and let \mathcal{L}_e denote the set of all strings on the alphabet D which contain an *even* number of 1's.

Example 1.2. Again let $D = \{0, 1\}$ and let \mathcal{L}_p denote the set of *palindromes* on D , that is, all strings on the alphabet D which read the same forwards as backwards.

Example 1.3. Let $\mathcal{L}_=$ denote the set of all strings on $D = \{0, 1\}$ which contain the same number of 0's as 1's.

Example 1.4. Let \mathcal{L}_{11} denote the set of all strings on $D = \{0, 1\}$ which do not contain two consecutive 1's.

Exercise 1.1. For each of the four languages in the preceding examples, list all the words in the language of length less than or equal to four.

In the *Notes on Elementary Probability* we defined an *information source* as a system $\mathcal{X} = (\Omega(D), \mathcal{B}, \mathcal{P}, \sigma)$, or $(\Omega^+(D), \mathcal{B}, \mathcal{P}, \sigma)$. Here D is a finite alphabet; $\Omega(D)$ and $\Omega^+(D)$ are the sets of one- or two-sided infinite strings on the alphabet D ; \mathcal{B} is a family of subsets of the set of strings on D which contains all the cylinder sets and is closed under complementation and countable unions and intersections; P is a probability measure defined for all sets in \mathcal{B} ; and σ is the shift transformation, which makes time go by or allows us to read one incoming symbol from the source at a time. The set of all finite strings found in all infinite sequences emitted by the source is *the language of the source*. Usually we deal with sources that are stationary and ergodic. By a major theorem of the twentieth century, every word in the language of a stationary ergodic source appears in almost every sequence emitted by the source with limiting frequency equal to the probability of the cylinder set to which it corresponds.

2. THE POSSIBILITY OF INFORMATION COMPRESSION

One if by land, two if by sea;
 And I on the opposite shore will be,
 Ready to ride and spread the alarm
 Through every Middlesex village and farm,
 For the country folk to be up and to arm.

Henry Wadsworth Longfellow

Suppose that someone has a choice of only two messages to send. Each message could consist of even a lengthy string of symbols, but the information boils down to just the choice between two alternatives: whether it is message A that is sent or message B. The messages could be numbered 0 and 1, or represented by one lantern or two in the steeple of Christ Church, Boston. (Interestingly, 0 lanterns or 1 lantern might not work: often one needs also to signal that a message is present.) This situation shows us in action the fundamental unit of information: the designation of one of two possibilities constitutes the *bit*. Information is measured by the minimum number of bits—elementary yes-no decisions—required to convey it.

In the Paul Revere story, we see that the message “The British are coming by land” has been *compressed* to the message “1”, and the message “The British are coming by sea” has been compressed to the message “2”. Any two symbols could have been used instead of 1 and 2, for example 0 and 1 would be a basic choice. But it takes a minimum of two symbols to distinguish the two distinct messages.

If we have 3 or 4 messages to choose among, we can distinguish them by assigning to each a different string of length 2 on the symbols 0 and 1. The original message strings could be quite long, but once the assignment of strings is made and agreed on, we have to send only 00, 01, 10, or 11. The messages have been compressed. If we have m messages to send, and we want to assign to each one a different binary string of length k , we must have

$$(1) \quad 2^k \geq m, \quad \text{that is,} \quad k \geq \log_2 m :$$

It takes binary strings of length $\log_2 m$ to distinguish m different messages.

Suppose now that someone has a repertoire of messages from which to choose, sending one a day, but with varying probabilities. Suppose further that the messages have widely varying lengths. We can also assign binary strings of varying lengths to these messages, and it would be smart to use shorter strings for the more probable messages. Then on average we will be sending fairly short messages, achieving information compression.

Example 2.1. Dagwood goes to the same diner for lunch every day. Usually he orders either “the special”, whatever it is, unless he doesn't like it, and then he orders chili, extra hot, with lettuce and tomato

on the side, ranch dressing, and two croissants. In the latter case, to save time, he just says “the usual”. Very rarely, he is tired of chili and the special is unattractive, so he orders “pea soup and meat loaf with French fries”. Note the economical use of letters on the average.

Example 2.2. During the summer in North Carolina, the weather forecast is often for partly cloudy weather, temperature in the high 80’s, chance of afternoon and evening thundershowers. Sometimes TV forecasters will even say, “Same as yesterday”. When the weather is unusual (which happens not so often), longer descriptions take place.

Here is a fairly efficient way to encode English text for compression that is suggested in the book by Pierce. Let us take for our *source alphabet* D the 26 English letters plus space, ignoring upper and lower case, punctuation marks, and other special characters. (If really needed, they can be spelled out.) In order to encode these $m = 27$ symbols by binary strings, that is, by a *code alphabet* $C = \{0, 1\}$, we would need strings of length at least $k = 5$, since 5 is the smallest integer k for which $2^k \geq m$ ($2^5 = 32$). But we propose to encode text not letter by letter, but actually word by word, at least for the *most probable* words.

Note: In what follows, we will also consider a *channel*, which typically accepts one binary digit per second and outputs one binary digit per second (maybe different strings come out than are put in). Then the code alphabet $C = \{0, 1\}$ will also be the *channel alphabet*. Any source to be sent across the channel will have to be first encoded into strings on the channel alphabet. (This happens all the time, for example when data put into a computer is converted to binary strings.)

Suppose we use binary strings of length 14 to encode our source, standard English text on an alphabet of 32 characters. There are $2^{14} = 16,384$ such strings. Let’s assign $16,384 - 32 = 16,352$ of these strings to the 16,352 most common English words and assign the 32 remaining strings to our 26 letters, space, and the most common punctuation symbols, so as to be able to spell out any words not in our list of the most probable 16,352.

If we go to encode an English text into binary strings of length 14 in this way (Plan A), most of the time we will be encountering the most frequent words, and occasionally we will have to spell out an unusual word that comes up. Is there any saving over character-by-character encoding? If we used binary strings of length 5 to encode each of our 32 characters (Plan B), an English string of length N would be encoded

to a binary string of length $5N$. What length binary string would Plan A yield?

The answer depends on the lengths of the words involved, since in Plan A we encode word by word. Pierce states that on average English words have length 4.5 characters, hence really 5.5 since a space precedes each word. So in Plan A, our input English text of length N contains approximately $N/5.5$ words, and most of them are probable. Each of these probable words is assigned a binary string of length 14, and the result is a binary string that is not much longer than $(N/5.5)14 \approx 2.55N$. Thus Plan A gives us an encoding that is about *half the length* of Plan B's character-by-character encoding.

In Plan B, we are using 5 binary digits per symbol. In Plan A we use only 2.55 binary digits per symbol—some excess, redundant, or predictable chaff has been winnowed out. Somehow the essential information in the message is less than 2.55 *bits per symbol*. Note that Pierce uses *bit* for the basic unit of information—a fundamental yes-no answer—and *binary digit* for the symbols 0 and 1.

The minimum number of binary digits per symbol needed to encode a message is taken to be the information content per symbol of the message. It is called the entropy and is measured in bits per symbol.

So we could have a source, such as English text, that is presented as 5 binary digits per symbol, say if we look at it character by character. But in actual fact the entropy of English text is less than 2.55 bits per symbol—on average, English text can be encoded by binary strings of length 2.55 per symbol. In fact, it turns out that English text has entropy about *one bit per symbol*:

A typical English text of length N (a string on 32 characters, say) can be encoded to a *binary string of approximately the same length!*

3. THE ENTROPY OF A SOURCE

The idea was announced at the end of the preceding section: the entropy of an information source is the minimum number of binary digits per symbol needed, on average, to encode it. The entropy is measured in bits per symbol. We will see how to try to achieve this compression by block coding. For now, we focus on three equivalent definitions of the entropy of a source, which do not require considering

all possible encodings. Each definition gives its own insight into the idea.

1. The entropy of a source is the *average information per symbol* that it conveys. How to quantify this?

One key idea is that *the amount of information gained equals the amount of uncertainty removed*.

Another is that when we are told that a highly probable event has occurred we gain less information than when we are told that a highly improbable event has occurred. So when we are told that an event A of probability $P(A)$ has occurred, we say that *the amount of information gained is $-\log_2 P(A)$ bits*. A couple of points to help explain this:

- If $P(A) = 1$, so that A is *almost sure* to occur, then the information conveyed in telling us that A occurred is 0.
- If $P(A)$ is very small, then when we are told that $P(A)$ occurred we gain a *lot* of information—we are really really surprised.
- We use \log_2 (logarithm base 2) because we want to measure the information in *bits*—the number of binary digits required to convey it. Recall that each simple yes-no distinction can be conveyed by giving one binary digit, 0 or 1. If the two possibilities are equally likely, each yields one bit of information. This gibes with the fact that if $P(A) = P(A^c) = 1/2$, then $-\log_2 P(A) = \log_2(1/2) = \log_2(2) = 1$. If we use *natural logarithms*, that is $\ln = \log_e$ ($e = 2.718281828459045\dots$), the unit of information is sometimes called the *nat*.
- If A_1 and A_2 are *independent events*, so that $P(A_1 \cap A_2) = P(A_1)P(A_2)$, then the information gained in learning that *both* events occurred is the *sum* of the information gains from learning that *each* of them has occurred. This is reasonable, since for independent events the occurrence of either one does not affect the probability of occurrence of the other. This point is one of the main justifications for the use of the logarithm in the definition of information.

Exercise 3.1. Calculate $\log_2 x$ in case $x = 32, 1/16, 20$, and -4 .

Suppose we have a source that is emitting symbols from a finite alphabet $D = \{0, 1, \dots, d-1\}$, symbol i occurring with probability p_i (but the symbols do not necessarily occur independently of one another). If we were to read just one symbol, the one that is arriving just

now, our *average information gain* would be

$$(2) \quad H(p) = - \sum_{i=0}^{d-1} p_i \log_2 p_i.$$

Note that here our probability space is $(\Omega(D), \mathcal{B}, P)$, where $\Omega(D)$ is the set of two-sided infinite sequences on the alphabet D . If we define $f(x) = -\log_2 p_{x_0}$, the logarithm of the probability of occurrence of the symbol being read at time 0, then the above average information gain is just the *expected or average value* of the random variable f :

$$(3) \quad H(p) = \mathbb{E}(f).$$

If the symbols were actually coming independently (that is, if we had a Bernoulli source—see §3 of the notes on probability) Equation 2 would actually give the entropy of the source. In general, though, occurrence of certain symbols can influence the probabilities of occurrence of other ones at various times, thereby affecting the amount of new information gained as each symbol is read. The three definitions of entropy of a source which we are here working up deal with this possible interdependence in three different ways. The first one suggests that we *sliding block code* the word into long blocks, say of length n , considering these as new symbols; apply Formula (2) to this new source to get an idea of its entropy (actually an overestimate); divide by n to get an overestimate of the entropy of the original source, in bits per original symbol; and hope that these numbers converge to a limit as $n \rightarrow \infty$ (in fact they will).

Let's describe this more precisely, with mathematical notation. Denote by D^n the collection of all words of length n on the elements of the alphabet D . The sliding block code mentioned above associates to each infinite sequence $x \in \Omega(D)$ with entries from D the sequence $s_n(x) \in \Omega(D^n)$ with entries n -blocks on the alphabet D defined by

$$(4) \quad (s_n(x))_j = x_j x_{j+1} \dots x_{j+n-1} \quad \text{for all } j \in \mathbb{Z}.$$

As we shift along the sequence x *one* step at a time, we see a sequence of symbols from D^n , which is the new sequence on the new alphabet of n -blocks.

Carrying out the strategy outlined in the preceding paragraph leads to the definition of *the entropy of the source* as

$$(5) \quad h(\mathcal{X}) = \lim_{n \rightarrow \infty} \left[-\frac{1}{n} \sum_{B \in D^n} P(B) \log_2 P(B) \right].$$

We repeat that this number is supposed to give the average information received per symbol (which equals the amount of uncertainty removed, or the degree of surprise) as we read the symbols emitted by the source one by one.

2. The second definition measures our average uncertainty about the next symbol to be read, given all the symbols that have been received so far. Naturally this involves the idea of conditional probability—see §2 of the *Notes on Elementary Probability*. Given a symbol $s \in D$ and a block $B \in D^n$, let us agree to denote by $P(s|B)$ the conditional probability that the next symbol received will be s , given that the string of symbols B has just been received. Since our source is stationary, the time when we are waiting to see whether s is sent does not matter, so we may as well take it to be the present, time 0. Thus

$$(6) \quad \begin{aligned} P(s|B) &= \frac{P\{x \in \Omega(D) : x_{-n} \dots x_{-1} = B, x_0 = s\}}{P\{x \in \Omega(D) : x_{-n} \dots x_{-1} = B\}} \\ &= \frac{P(Bs)}{P(B)}. \end{aligned}$$

Given that a block $B \in D^n$ has just been received, according to the preceding discussion (in connection with Definition 1 of entropy), our average information gain upon receiving the next symbol is

$$(7) \quad - \sum_{s \in D} P(s|B) \log_2 P(s|B).$$

We average this over all the possible n -blocks B , weighting each according to its probability, and then take the limit as $n \rightarrow \infty$, so that we are allowed to look farther and farther back into the past to decide how surprised we ought to be when a symbol s arrives at time 0. This gives us the second definition of the entropy of the source:

$$(8) \quad h(\mathcal{X}) = \lim_{n \rightarrow \infty} \left[- \sum_{B \in D^n} P(B) \sum_{s \in D} P(s|B) \log_2 P(s|B) \right].$$

We repeat that this measures our average information gain when a symbol is received at time 0, given all of the symbols that have been received in the past.

3. The third description that we present of the entropy of a (stationary, ergodic) source is actually a theorem, due to Claude Shannon, Brockway McMillan, and Leo Breiman. It leads to methods for finding the entropy of a source by examining the typical sequences that it

emits. With probability 1,

$$(9) \quad h(\mathcal{X}) = \lim_{n \rightarrow \infty} -\frac{1}{n} \log_2 P(x_0 x_1 \dots x_{n-1}).$$

This implies that for large n , there are approximately $2^{nh(\mathcal{X})}$ probable n -blocks B on the symbols of D , each with probability approximately $2^{-nh(\mathcal{X})}$. This is because for the initial n -block $B = x_0 x_1 \dots x_{n-1}$ of a typical sequence x emitted by the source, for large n we will have

$$(10) \quad h(\mathcal{X}) \approx -\frac{1}{n} \log_2 P(B),$$

so that

$$(11) \quad \log_2 P(B) \approx -nh(\mathcal{X}) \quad \text{and hence} \quad P(B) \approx 2^{-nh(\mathcal{X})}.$$

This third “definition” of the entropy $h(\mathcal{X})$ of a stationary ergodic source \mathcal{X} substantiates our initial effort to understand the entropy of a source as the minimal average number of binary digits per symbol needed to encode it. Recall Formula (1): In order to be able to assign a different binary string of length k to each of m messages, we need $k \geq \log_2 m$. Now choose n large enough so that there are approximately $m = 2^{nh(\mathcal{X})}$ probable n -blocks. Using the strategy outlined in the preceding section, with $k \geq \log_2 m = nh(\mathcal{X})$, we can assign a binary string of length k to each of these probable n -blocks (reserving a few binary strings to assign to basic characters, so as to be able to spell out improbable words). The result is that, on the average, strings of length n are coded one-to-one by binary strings of length $k \approx \log_2 m = nh(\mathcal{X})$, so that the source is encoded by

$$(12) \quad \frac{k}{n} \approx \frac{nh(\mathcal{X})}{n} = h(\mathcal{X}) \quad \text{binary digits per symbol.}$$

(It takes some further argument to see that it is not possible to improve on this rate.)

Example 3.1. The entropy of a *Bernoulli source* (see §3 of the *Notes on Elementary Probability*) which emits symbols from an alphabet $D = \{0, 1, \dots, d-1\}$ independently, symbol i appearing with probability p_i , is

$$(13) \quad H(p_0, \dots, p_{d-1}) = -\sum_{i=0}^{d-1} p_i \log_2 p_i.$$

This can be seen by easy calculation based on any of the three definitions of entropy given above.

Example 3.2. Consider a *Markov source* as in §4 of the *Notes on Elementary Probability* on the alphabet $D = \{0, 1, \dots, d-1\}$ determined by a probability vector $p = (p_0, \dots, p_{d-1})$, which gives the probabilities of the individual symbols, and a matrix $P = (P_{ij})$, which gives the probabilities of transitions from one symbol to the next, so that

$$(14) \quad P\{x \in \Omega(D) : x_n = j | x_{n-1} = i\} = P_{ij} \quad \text{for all } i, j, n.$$

A short calculation based on Definition 2 above shows that this source has entropy

$$(15) \quad H(p, P) = - \sum_{i=0}^{d-1} p_i \sum_{j=0}^{d-1} P_{ij} \log_2 P_{ij}.$$

Exercise 3.2. Suppose we have a source \mathcal{X} on the alphabet $D = \{0, 1\}$ which emits 0's and 1's with equal probabilities, but subject to the requirement that no 0 can be emitted immediately after another 0, and no 1 can be emitted immediately after another 1. Use each of the three preceding definitions of entropy to calculate the entropy of this source.

Exercise 3.3. Given a source $\mathcal{X} = (\Omega(D), \mathcal{B}, \mathcal{P}, \sigma)$ as above and $n = 1, 2, \dots$, we can associate to it the *n-blocks source*

$$(16) \quad \mathcal{X}^{(n)} = (\Omega(D^n), \mathcal{B}^{(n)}, \mathcal{P}^{(n)}, \sigma^n),$$

as in the beginning of §3 of the *Notes on Elementary Probability*. The symbols of the new source are *n*-blocks on the original alphabet D , the observable events and probability measure are defined in a natural way (note—not necessarily assuming that the *n*-blocks come independently), and we now shift *n* steps at a time instead of 1, in order always to shift to the next *n*-block. Use as many as possible of the definitions or explanations of entropy given above to verify that $h(\mathcal{X}^{(n)}) = nh(\mathcal{X})$.

4. THE ENTROPY OF A LANGUAGE

We have just seen that a stationary ergodic source emits, for large n , about $2^{nh(\mathcal{X})}$ probable blocks of length n . Given any language \mathcal{L} on a finite alphabet D , we can ask for a number $h(\mathcal{L})$ such that the *total number* of words of length n in the language is approximately $2^{nh(\mathcal{L})}$. If we define \mathcal{L}_n to be the set of words in \mathcal{L} that have length n , and $|\mathcal{L}_n|$ to be the size of this set, that is, the number of words in \mathcal{L} that have length n , we seek

$$(17) \quad h(\mathcal{L}) = \lim_{n \rightarrow \infty} \frac{1}{n} \log_2 |\mathcal{L}_n|.$$

Sometimes this limit exists, and then we call it the *entropy* or *topological entropy* of the language. If \mathcal{X} is a source that emits only words from a language \mathcal{L} , then it follows from above that

$$(18) \quad h(\mathcal{X}) \leq h(\mathcal{L}).$$

Exercise 4.1. Calculate or at least estimate the entropies of the languages in the examples in §1.

5. SOURCE CODING FOR COMPRESSION

So far we have implicitly been assuming that we are dealing with a *noiseless channel*: a source \mathcal{X} is emitting symbols from an alphabet D , and we receive exactly what is sent (or at least we can recover, with probability 1, what was sent from what was received). (Soon we will consider the possibility that there may be random errors, so that what is received may differ from what was sent.) We have also considered *recoding* the source, perhaps replacing strings on the alphabet D by binary strings, so as to improve (lower) the average number of new binary digits per old symbol, or to be able to input our messages to a channel which accepts and puts out only binary strings. For example, encoding English text character by character requires about 5 binary digits per character (allowing for 32 characters); but we saw above how to recode so that in a long typical text we use only about 2.55 binary digits per English character.

Shannon's Source Coding Theorem says that given a stationary ergodic source \mathcal{X} of entropy $h(\mathcal{X})$, it is possible to encode the strings output by the source in a one-to-one (uniquely decodable) way so that the average number of binary digits used per source symbol is as close as we like to the entropy of the source. Moreover, we cannot recode to get the average number of binary digits used per source symbol to be less than the entropy of the source.

In Section 3 we saw the possibility of doing this sort of “information compression”, but the method envisaged there depended on forming a list of the most probable blocks of a certain length that are emitted by the source that we are trying to code. There are more efficient and practical ways to accomplish this goal, devised by Shannon and Fano, Huffman, Lempel and Ziv (used for example in computer data compression programs like gzip), and others. Pierce presents the method of *Huffman coding*:

List the symbols to be encoded (possibly the set D^n of words of some fixed length length n emitted by the source) in order of decreasing probabilities. We operate on this list of probabilities, converting it into a new list. Choose two of the lowest possible probabilities in the list and draw lines from each to the right that meet. Label the upper line with 1 and the lower with 0. At the joining point, enter the sum of the two probabilities; this replaces the pair of probabilities that we started with, thus giving us a new list of probabilities whose length is 1 less than that of the original.

Continue in this way, constructing a tree with edges labeled by 1 and 0 and vertices by probabilities, until we arrive at a vertex labeled by probability 1. Now starting from this final vertex, reading right to left, write down the labels on the edges encountered as one traverses the graph toward an original symbol s . The resulting binary string, $b(s)$, is the codeword assigned to s , and the set of codewords is called the *Huffman code* assigned to the set of source words to be encoded.

Here are a couple of important points about this Huffman coding procedure:

- The Huffman code is a *variable-length* code: each of m source symbols is assigned a binary string of length less than or equal to m .
- The set of codewords is *prefix free*: no code word is an initial segment of any other codeword. This guarantees that the code is *invertible*: each string of codewords can be parsed into a unique string of source words.
- When a string of symbols from a *Bernoulli* source \mathcal{X} of entropy $h(\mathcal{X})$ is encoded by the Huffman procedure, the average number of binary digits used per symbol is less than $h(\mathcal{X}) + 1$.

This last observation requires some proof, which we do not attempt here. But we can see immediately how it could yield Shannon's Source Coding Theorem. Choose a large n and form the source $\mathcal{X}^{(n)}$ which reads n -blocks of the original source, always moving from one full n -block to the next (by applying the n 'th power of the shift transformation). By Exercise 3.3, this new source has entropy $h(\mathcal{X}^{(n)}) = nh(\mathcal{X})$. Moreover, it is approximately Bernoulli, in the sense that

$$(19) \quad - \sum_{B \in D^n} P(B) \log_2 P(B) \approx h(\mathcal{X}^{(n)})$$

(see Formula (5)). When the symbols of $\mathcal{X}^{(n)}$ are encoded to binary strings by the Huffman procedure, according to the above we use, approximately and on average, less than $h(\mathcal{X}^{(n)}) + 1 = nh(\mathcal{X}) + 1$ binary digits per symbol of $\mathcal{X}^{(n)}$, and therefore we use on average less than

$$(20) \quad \frac{h(\mathcal{X}^{(n)}) + 1}{n} = \frac{nh(\mathcal{X}) + 1}{n} = h(\mathcal{X}) + \frac{1}{n}$$

binary digits per symbol of the original source \mathcal{X} .

Finally, we attempt to explain the discussion in Pierce, pp. 97–98, about the relation of this theorem to a kind of channel capacity. Suppose that we have a noiseless channel which accepts binary strings as inputs and also puts out binary strings—maybe not exactly the same ones, but at least in an invertible manner, so that the input can be recovered from the output, at least with probability 1. Let us also introduce a time element for the channel, so that it can transmit τ binary digits per second. Now if n -blocks from a source \mathcal{X} are coded to binary strings of length k to be fed into the channel, presumably with k/n on average not much larger than $h(\mathcal{X})$, then τ binary digits per second will correspond to approximately $\tau n/k$ symbols of the original source traversing the channel per second (in binary digit encoded form).

Example 5.1. Suppose we have a source whose 2-blocks are coded by binary strings of average length 4, and we have a channel of time-capacity $\tau = 5$ binary digits per second. Then binary strings move across the channel at 5 digits per second. This corresponds to source strings being input at one end of the channel and output at the other end, without taking into account any encoding or decoding time, at $5(2/4)$ symbols per second. Each binary string of length 1 makes it across the channel in $1/5$ seconds, so each binary string of length 4 makes it across the channel in $4/5$ seconds and corresponds to a source string of length 2, so source strings are being conveyed at $(4/5)/2$ seconds per symbol. Then take the reciprocal.

Thus we have an average transmission rate of source symbols across this channel of

$$(21) \quad \tau \frac{n}{k} = \frac{\tau}{k/n} \quad \text{source symbols per second.}$$

Recalling that we can make k/n , which is always greater than or equal to $h(\mathcal{X})$, as close as we like to $h(\mathcal{X})$, but not any smaller, we see that the rate in (21), which is always smaller than $\tau/h(\mathcal{X})$, can be made as close as we like to $\tau/h(\mathcal{X})$, but not any greater. This is supposed to explain the set-off statements at the top of p. 98 in Pierce.

Exercise 5.1. A source emits once a second one of 6 different symbols, with probabilities $1/2, 1/4, 1/8, 1/16, 1/32, 1/32$, respectively, independently of what has been emitted before or will be emitted later.

(a) Calculate the entropy of the source.

(b) Devise a Huffman code for this source.

(c) If a typical long message of length N emitted by the source is recoded with the code from part (b), what will likely be the length of the resulting binary string?

(d) If the source is Huffman encoded as in Part (b) and then is put into a channel that conveys 0.57 binary digits per second, what will be the transmission rate in source symbols per second?

6. THE NOISY CHANNEL: EQUIVOCATION, TRANSMISSION RATE, CAPACITY

In the preceding section we saw how to recode a source \mathcal{X} of entropy $h(\mathcal{X})$ to *compress* its output, representing it by binary strings with a rate not much more than $h(\mathcal{X})$ binary digits per source character. Now we consider how to add some *check digits* to protect against errors in transmission. As before, we suppose that the output of our source is recoded into binary strings and fed into a channel which accepts and puts out binary strings (or more generally accepts strings on an alphabet A and puts out strings on an alphabet B), but we now admit the possibility of random alteration of the strings during transmission, so that the input may not be recoverable, with probability 1, from seeing the output. Can we protect against this? How well? At what cost?

One way to *detect* errors in transmission is just to repeat every digit twice: for example, to send a string 010010, put 001100001100 into the channel. This method will detect single isolated errors, but it entails the cost of doubling transmission time. To *correct* single isolated errors, one could triple each digit, and this would multiply the cost even more. Can we correct *all errors* (with probability 1)? If so, at what cost?

A *channel* is actually a family of probability measures, one for each infinite string that can be fed into the channel, specifying the probabilities of the observable events regarding the output string, namely that in the output we see a particular finite-length block at a particular time. Further technical conditions must be put on channels to ensure

that they make sense and that the theorems we want to prove are true. This work is far from finished. In this section we deal with a very simple kind of noisy channel, the *discrete memoryless channel*, for which each incoming symbol produces the various possible output symbols with fixed probabilities, independently. Thus we are given transition probabilities

$$(22) \quad \begin{aligned} p_x(y) &= \text{probability that } y \text{ comes out when } x \text{ is put in,} \\ q_y(x) &= \text{probability that when } y \text{ comes out } x \text{ was put in,} \end{aligned}$$

where x, y are symbols of the channel alphabet (for us usually 0 and 1). We will use the same notations for analogous probabilities when x and y are *strings* on the input and output alphabets.

Given such a channel, it turns out that we can associate to it an important number, C , called the *capacity* of the channel, which tells the complete story about the prospects for coding a source for error protection before trying to send it across that channel. We will also define a quantity E , called the *equivocation*, which is a measure of the amount of information lost on average when messages are sent across the channel. We also consider F , the *average frequency of error*, the proportion of digits in the limit (in long messages) which are changed in transmission across the channel. Once these quantities are understood, we can appreciate

Shannon's Channel Coding Theorem: Given a channel of capacity C , a source \mathcal{X} of entropy $h(\mathcal{X}) < C$, and any number $\epsilon > 0$, we can recode the source so that when it is sent over the channel and the result is decoded both the frequency of error and the equivocation are less than ϵ . If $h(\mathcal{X}) > C$, we must always have the equivocation $E \geq h - C$, but by recoding the source we can make it as close to $h - C$ as we like: given $\epsilon > 0$, we can have $h - C \leq E < h - C + \epsilon$.

In this statement, entropy, capacity, etc. are thought of in *bits per second*. If we eliminate time as a consideration, then this remarkable result says that any (stationary ergodic) source can be sent across any (binary symmetric) channel virtually error free: first compress the source by source coding, then assign n -blocks of the source to long enough k -blocks of the channel. The theorem guarantees that we can recode to reduce the frequency of errors, and the equivocation, below any desired level. Similar results apply to more general channels, but satisfyingly general and clean results have not yet been achieved.

It is also remarkable that this theorem is proved by a *random coding argument*. One selects k and n large and with n/k having the right ratio (close to $C/h(\mathcal{X})$ in case all alphabets involved are $\{0, 1\}$). Then a calculation is done to show that when each n -block of the source is assigned to a different randomly chosen k -block to be put into the channel, with high probability we get a recoding of the source that satisfies the theorem. Another way to say it refers to the approximately $2^{nh(\mathcal{X})}$ “good n -blocks” put out by the source (see the third definition given above of the entropy of the source). It turns out that there are also approximately 2^{kC} *distinguishable k -blocks* that can be put into the channel, blocks whose corresponding outputs will probably be very different from one another. (In fact when we choose this many k -blocks at random, if k is large enough we will with high probability get a set of distinguishable blocks.) So when we assign the good n -blocks of the source to the distinguishable k -blocks to be put into the channel, we code most of the output of the source in a way that the channel is unlikely to confuse.

To finish off our discussion of this amazing theorem, we have to define the key terms “capacity” and “equivocation”, and before that we have to define and examine some other concepts as well. We assume that we have a stationary ergodic source \mathcal{X} , with an alphabet A , that has already been encoded, if necessary, so that its alphabet matches the input of our channel. When this source is put into the channel, the output, the composition of input source and channel, constitutes *another source*, $\mathcal{Y} = \mathcal{S}(\mathcal{X})$, on another alphabet B (usually $\{0, 1\}$). We assume that the channel has the property that this source \mathcal{Y} will *also* always be stationary and ergodic. (It is difficult to describe exactly which channels have this property, but the binary symmetric one does.) We will now consider strings x of length n that are put into the channel and examine the strings y of length n that emerge (we assume zero time delay).

(1) The *joint input-output process* $\mathcal{X} \vee \mathcal{Y}$ is the source whose output consists of infinite strings of pairs of symbols (a, b) , with a a symbol emitted by the source \mathcal{X} at each time j and fed into the channel, and b the symbol output by the channel at the same time j . This corresponds to a know-it-all observer who can see both the input and the output of the channel at the same time. To avoid complicated notation, we use the same symbol P to describe the probabilities of events related to this process as for the processes \mathcal{X} and \mathcal{Y} . The entropy of the joint

input-output process is

$$(23) \quad h(\mathcal{X} \vee \mathcal{Y}) = - \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{\substack{x \in A^n \\ y \in B^n}} P(x, y) \log_2 P(x, y).$$

(2) We define the *uncertainty about the output, given the input*, to be

$$(24) \quad H_{\mathcal{X}}(\mathcal{Y}) = - \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{\substack{x \in A^n \\ y \in B^n}} p(x) p_x(y) \log_2 p_x(y).$$

The sum gives the weighted average, over all input strings x of a given length n , of the uncertainty about the output (which equals the amount of information gained or the degree of surprise when the output is received) when the input string x is known.

(3) We define the *uncertainty about the input, given the output*, to be

$$(25) \quad H_{\mathcal{Y}}(\mathcal{X}) = - \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{\substack{x \in A^n \\ y \in B^n}} q(y) q_y(x) \log_2 q_y(x).$$

The sum gives the weighted average, over all output strings y of a given length n , of the uncertainty about the input when the output string y is known. This quantity measures the loss of information when the source is sent across the channel, and it is called the *equivocation*.

(4) We have the key relation

$$(26) \quad h(\mathcal{X} \vee \mathcal{Y}) = h(\mathcal{X}) + H_{\mathcal{X}}(\mathcal{Y}) = h(\mathcal{Y}) + H_{\mathcal{Y}}(\mathcal{X}).$$

The first equality says that the uncertainty about what will come out of the joint source is the sum of the uncertainty about what is put into the channel plus the uncertainty about what will come out of the channel when what is put in is already known. Thus these equations seem to make some sense.

(5) When a source \mathcal{X} is connected to the channel as above, producing an output process \mathcal{Y} and a joint input-output process $\mathcal{X} \vee \mathcal{Y}$, we define the *information transmission rate* to be

$$(27) \quad \begin{aligned} R(\mathcal{X}) &= h(\mathcal{X}) - H_{\mathcal{Y}}(\mathcal{X}) \\ &= h(\mathcal{Y}) - H_{\mathcal{X}}(\mathcal{Y}) \\ &= h(\mathcal{X}) + h(\mathcal{Y}) - h(\mathcal{X} \vee \mathcal{Y}). \end{aligned}$$

This probably calls for some discussion. The first expression for R , namely $R(\mathcal{X}) = h(\mathcal{X}) - H_Y(\mathcal{X})$, says that the rate at which information comes across the channel equals the rate at which the source puts information into the channel, minus the amount lost in the channel—the uncertainty remaining about what the input was once we see the output. The second expression says the rate is the amount of information received (uncertainty about the output), minus the spurious extra uncertainty due to noise in the channel—the uncertainty about what will come out even when we know what was put in. The third expression says that information flow across the channel is given by the sum of the uncertainties about the input and output separately, minus the uncertainty about the process in which the input and output are connected.

(6) Finally we can define the *capacity* C of the channel to be the maximum (more precisely *supremum*, abbreviated sup—the supremum of the interval $[0, 1)$ is 1, while this set has no maximum) of all the possible information transmission rates over all the possible stationary ergodic sources that can be fed into the channel:

$$(28) \quad C = \sup\{R(\mathcal{X}) : \mathcal{X} \text{ is a source input to the channel}\}.$$

Example 6.1. Let's consider the binary symmetric channel in which when a symbol 0 or 1 emerges, there is probability $\alpha \in (0, 1)$ that it is the same as the symbol that was put in, independently of all other symbols. Thus

$$(29) \quad q_0(0) = q_1(1) = \alpha, \quad \text{while} \quad q_0(1) = q_1(0) = 1 - \alpha.$$

There are reasons to believe that the maximum information transmission rate for such a channel will be achieved by the Bernoulli source (see §6 of the *Notes on Elementary Probability*) which at each instant emits one of the the symbols 0 and 1, each with probability $1/2$, independently of what is emitted at any other instant. (At least for this source we have maximum uncertainty about what it will emit, and the symmetry of the channel makes us think that the optimal source should also be symmetric). Then the symbol output probabilities are $q(0) = q(1) = 1/2$, and because of independence in Formula (25) we can look just at 1-blocks and do not have to take a limit. We find that

$$(30) \quad H_Y(\mathcal{X}) = -\frac{1}{2}[\alpha \log_2 \alpha + (1-\alpha) \log_2(1-\alpha) + (1-\alpha) \log_2(1-\alpha) + \alpha \log_2 \alpha],$$

so that

$$(31) \quad C = h(\mathcal{X}) - H_Y(\mathcal{X}) = 1 + \alpha \log_2 \alpha + (1 - \alpha) \log_2(1 - \alpha).$$

Note that

$$(32) \quad H(\alpha) = -[\alpha \log_2 \alpha + (1 - \alpha) \log_2(1 - \alpha)] \geq 0,$$

so that $C \leq 1$. Moreover, if $\alpha = 1/2$, so that we have no idea whether each symbol coming out equals the one put in or not, the channel capacity is 0. On the other hand, the capacity is the maximum of 1 *both* when $\alpha = 0$ and when $\alpha = 1$: when the symbol out is always different from the symbol in, we can still recover the input message perfectly well.

7. CODING FOR ERROR PROTECTION

Shannon's Channel Coding Theorem establishes the existence of good error-correcting codes by a random coding argument and so does not tell us how to construct such good sets of codewords in practice. Finding methods to construct good codes, especially ones that are easy to implement, continues as an active area of research, some of it involving very fancy modern mathematics, with results which have immediate impact in modern society.

We look here at an early example of a clever error-correcting coding system, the *Hamming codes*. For each $n = 2, 3, \dots$ there is a Hamming code consisting of binary strings of length $2^n - 1$. The first $2^n - n - 1$ binary digits are *data bits*, and the final n are *check bits*. (We relax here the distinction between "binary digits" and "bits".) For example, consider the case $n = 3$, which gives us strings of length 7 with 4 data bits, thus called the *Hamming (7, 4) code*. Each data string of length 4 (and there are $2^4 = 16$ of them) $a = a_1a_2a_3a_4$ is assigned a string $c = c(a) = c_1c_2c_3$ of 3 check bits according to the following rule, in which all the additions are modulo 2:

$$(33) \quad \begin{aligned} c_1 &= a_1 + a_2 + a_3 \\ c_2 &= a_1 + a_3 + a_4 \\ c_3 &= a_2 + a_3 + a_4. \end{aligned}$$

Let's think for a moment about the strategy behind making a set of codewords. We have some data that we want to send across a noisy channel. We will assign each of our data symbols (or n -blocks, possibly the result of a preliminary source coding for data compression) to a different codeword to be put into the channel. We want our codewords to be somehow stable when they go across the channel, so that when we look at what comes out of the channel we will have a pretty good

idea of what went in. The channel may change some of the digits in each codeword, but we want the code to be such that it will be very unlikely that two codewords will change to the same string. A natural desideratum, then, is that every codeword should differ from every other codeword in a large number of places. There is an associated concept of distance, called the *Hamming distance*: if u and v are binary strings of length n , we define

$$(34) \quad d_H(u, v) = \sum_{i=1}^n |u_i - v_i| = \sum_{i=1}^n (u_i + v_i) \pmod{2}.$$

(Note the convenience of arithmetic modulo 2: adding detects when two entries disagree.) Thus we look for codes with many codewords, all at large Hamming distance from one another. And of course we want them also to be short if possible. Visualizing the codewords as points in a space that are well spread out may be helpful.

Note that the Hamming code given above is *linear*: the modulo 2 sum of two codewords is again a codeword. This is because the check bits are just modulo 2 sums of certain data bits, so that if a data word a has a check word $c(a)$ added on its end, then $c(a + a') \equiv c(a) + c(a') \pmod{2}$; thus if $ac(a)$ and $a'c(a')$ are codewords, so is their (entrywise modulo 2) sum, since it equals $(a + a')c(a + a')$.

Of course the sum of a codeword with itself is the word of all 0's, which we denote by $\bar{0}$. Then, with addition modulo 2,

$$(35) \quad d_H(u, v) = \sum_{i=1}^n (u_i + v_i) = d_H(u + v, \bar{0}).$$

Thus the *minimum distance* of the code (the minimum distance between any two codewords) is the same as the minimum distance from any nonzero codeword to $\bar{0}$, and this is the same as the *weight* t of the code, which is the minimum number of 1's in any nonzero codeword.

If we want to be able to *correct* E errors in the digits of any codeword due to noise in the channel, we will want to use a code whose minimum distance is $2E + 1$: then changing no more than E digits in codewords will still keep them distinguishable. When we look at a word put out by the channel, we will interpret it as the codeword that is closest to it in terms of the Hamming distance. For the Hamming $(7, 4)$ code given above, the weight, and thus the minimum distance, is 3, so we can correct 1 error in transmission, but not 2. Note that this includes errors in transmission of either data or check bits.

With the $(7, 4)$ code we can *detect* up to two errors in transmission of any codeword, since it requires changing at least 3 entries to convert any codeword to another codeword. In general, a code with minimum distance t will allow *detection* of up to $t - 1$ errors, and *correction* of up to $\lfloor (t - 1)/2 \rfloor$ errors. (Recall that $\lfloor x \rfloor$ means the greatest integer that is less than or equal to x .)

REFERENCES

- [1] J. R. Pierce, *Symbols, Signals and Noise*, Harper Torchbooks, New York, 1961.