Notes on Information Theory

Riccardo Lo Iacono

September 11, 2025

Contents.

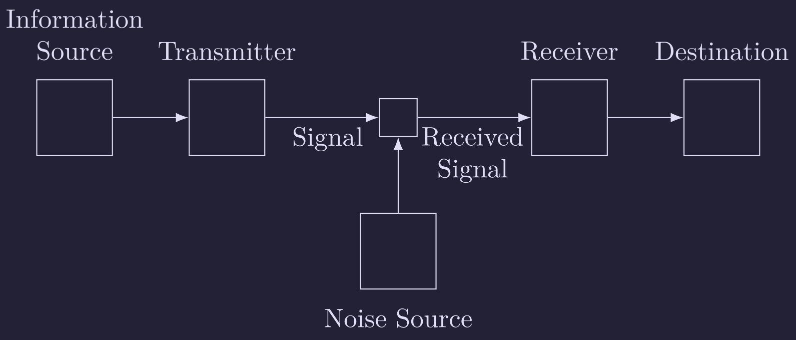
- 1 - Basics of information theory.

Information theory, alongside data compression, plays a key role in modern computer science. The former provides theoretical tools useful, among other things, for understanding the limits of computability; the latter enables the reduction of space requirements (in terms of bits) without loss of information.

- 1.1 - Shannon's Entropy.

When discussing information, we all have a general notion of what it represents, but how is it defined formally? Is there a way to quantify information?

The first to answer these questions was *Claude Shannon*, widely considered the father of information theory. In [1], Shannon analyzes various communication systems: from the discrete noiseless one to the continuous noisy one. A general structure of such systems is shown below.



Here:

- The *Source*, or more precisely, the *Information Source*, refers to some entity (a human, a computer, etc.) that produces messages.
- The *Transmitter* encodes the messages coming from the source and transmits them through the channel.
- The *Channel* is the physical medium through which the messages are transmitted.
- The *Receiver* has the opposite role of the transmitter.
- The *Destination* is the entity to which the messages are intended.

Remark. We note that, henceforth, we refer to the noiseless source. We also assume that the source is memoryless; that is, each symbol produced is independent of the previous one.

Let S be a source of information. Let Σ be the alphabet of symbols used by the source, and for each symbol let $p_i = \Pr(S = \sigma_i)$ at any given time. We seek a function H, if it exists, that quantifies the uncertainty associated with S.

One can prove (see [1, Appendix 2]) that the only form that H can take is the following:

$$H = -K \sum_{i=1}^{|\Sigma|} p_i \log_b p_i. \tag{1}$$

Here, K is a positive constant, and b is usually 2.

We define $H(S)$ to be the *entropy* of S. Formally, entropy measures the average amount of information contained in each symbol of the source output. Thus, the information associated with an event[1] can be defined as the reduction in uncertainty once the outcome is observed.

- 1.2 - Encoding of a source.

Let's now focus our attention onto the source itself. In general, the alphabets they use may not be suitable for transmission for various reasons. For this reason, an *encoder* is used to convert the source alphabet into a new one, which is more suitable for transmission. On the receiving side, a *decoder* is used to convert back to the original alphabet. This process is called *source encoding/decoding*.

Formally, given a source S defined on some alphabet, and X a new alphabet, called the *input alphabet*, we define a function $C : S \to X$ that maps sequences of symbols of S to sequences of symbols in X. A sequence of symbols in X is called a *codeword*.

Before we consider any particular *code*, let us consider the general case. Let C be a generic mapping from a source S to some new alphabet X. Since we have imposed no conditions on C, one may define it in such a way that two or more source symbols share the same codeword. It is easy to observe that, in doing so, the decoded message may not be correct or even unique.

Example

Consider the following alphabet: $\Sigma = \{s_1, s_2, s_3, s_4\}$. Assume the code $C = \{0, 11, 01, 11\}$ was defined for Σ. How should we decode the text 000111? There are two possible interpretations: either as $s_1 s_1 s_3 s_2$ or as $s_1 s_1 s_3 s_4$. As noted above, the decoded message may not be unique. Moreover, without additional context, there is no way to determine which interpretation is correct.

[1] In this context, by "event" we refer to the symbol produced by the source at a given time.

From the above example, we can conclude that a “good” code must encode each source symbol with a unique codeword. We call such codes *non-singular codes.*

- 1.2.1 - Uniquely decodable codes (UD).

One might think that non-singular codes are sufficient, but in most they don’t. In fact, it can happen that a codeword is a prefix of another, making decoding ambiguous or tedious.

Example

Consider the following code: $C = \{0, 1, 01, 11\}$. Assume the alphabet of the previous example. How should we decode the string 000111? Again, multiple decodings are possible: for example $s_1s_1s_3s_2s_2$ or $s_1s_1s_3s_4$. The correct decoding cannot be determined unless context is given.

We say that a code is UD if and only if each sequence of codewords corresponds to at most one sequence of source symbols. From this definition, two questions follow:

- How do we construct such codes?
- How can we check whether a given code is UD?

The next section focuses on the second question, while *Sections* **??**-**??** focus on the construction of UD codes.

- 1.2.2 - Sardinas-Patterson algorithm.

The *Sardinas-Patterson* algorithm provides a way to check whether a code is UD or not. Conceptually, the algorithm and its underlying theorem are based on the following remark: consider a string that is the concatenation of codewords. If we try to construct two distinct factorizations, each word in one of the factorizations is either part of a word in the other factorization, or it starts with a prefix that is a suffix of a word in the other factorization. Hence, a code is non-UD if a suffix is itself a codeword.

As stated before, the algorithm is based on a theorem, given below.

Theorem 1.1 *Given C a code on an alphabet Σ, consider the sets $S_0, S_1, \ldots$ such that:*

- $S_0 = C$
- $S_i = \{\omega \in \Sigma^* \mid \exists\alpha \in S_0, \exists\beta \in S_{i-1} : \alpha = \beta\omega \vee \beta = \alpha\omega\}$

Then a necessary and sufficient condition for C to be a UD code is that $\forall n > 0, S_0 \cap S_n = \varnothing$.

Example

Consider the following code: $C = \{a, c, ad, abb, bad, deb, bbcde\}$. Is it UD? Applying Sardinas-Patterson step by step, we get the following:

Iteration 1 Let $\beta = a$. If we let $\omega = d$, we have $\alpha = \beta\omega = ad \in S_0$; hence, $\omega = d \in S_1$. By the same logic we $bb \in S_1$.

Iteration 2 Let $\beta = d$. If we let $\omega = eb$, we get $\alpha = deb \in S_0$; thus, $eb \in S_2$. With a similar reasoning we get $cde \in S_2$.

Iteration 3 Let $\alpha = c$. If we let $\omega = de$, it derives that $\beta = cde \in S_2$; therefore, $de \in S_3$.

Iteration 4 Let $\beta = de$. If we let $\omega = b$, we get $\alpha = deb \in S_0$; hence, $b \in S_4$.

Iteration 5 Let $\beta = b$. If we let $\omega = bcde$, we have $\alpha = bbcde \in S_0$; thus, $bcde \in S_5$. By the same logic $ad \in S_5$

From the above steps (summarised in *Table 1*), it follows that the code is not UD since $S_0 \cap S_5 = \{ad\} \neq \varnothing$.

S_0	S_1	S_2	S_3	S_4	S_5
a					
c			de		
ad	d				
abb	bb				
bad					ab
deb		eb		b	
bbcde		cde			bcde

Table 1: Steps of Sardinas-Patterson for the code C. Note that each ω has been added to the row of the value use for α.

- 1.2.3 - Prefix codes.

Let's look back at the example of *Section 1.2.1*. Given the codewords $\{0, 1, 01, 11\}$ for the source symbols s_1, s_2, s_3, s_4 respectively, how should we decode the string 000111? As noted before, unless some context is given, we cannot be certain.

Then, what is the issue? Essentially, even though each source symbol is assigned a distinct codeword, these codewords are not prefix-free, meaning that some are prefixes of others (e.g., the codeword for s_1 is a prefix of the codeword for s_3). From this observation, a natural solution is to define codes that are prefix-free. One can also prove that prefix-free (or simply prefix) codes are also instantaneous; that is, each codeword can be immidiately decoded without ambiguity.

Theorem. *Let C be a prefix-free code; then C is UD.*

Proof It follows directly from *Theorem 1.1.* □

- 1.3 - Average code length and Kraft inequality.

In the previous section, we introduced prefix codes. The question now is: given two distinct prefix-free codes, which one is more efficient? A good choice is the code that, on average, has the shortest length.

Definition (Average code length (ACL)) Let C be a code with source alphabet $S = \{s_1, \dots, s_n\}$ and code alphabet $X = \{x_1, \dots, x_m\}$. Let $\{c_1, \dots, c_n\}$ be the codewords with lengths $l_1, \dots, l_n$, respectively. Let $\{p_1, \dots, p_n\}$ be the probabilities of the source symbols. Then, we define the quantity

$$L_S(C) = \sum_{i=1}^{n} p_i l_i$$

as the average code length of the code C.

From the above, it makes sense to look for the UD code with the lowest average code length. That is, among all UD codes for the same source and with the same code alphabet, the one with the lowest ACL. But how do we find such codes? A good starting point is the entropy of the source. Once again, thanks to Shannon, we have the following result.

Theorem (Shannon) *Let C be a UD code for a memoryless source S, whose probabilities are $\{p_1, \dots, p_n\}$, and code alphabet X of size d. Then*

$$L_S(C) \geq \frac{H(S)}{\log_b d}$$

- 1.3.1 - The Kraft-McMillan inequality.

We have discussed what prefix-free codes are, and what the average code length represents. We now provide a necessary and sufficient condition for the existence of a prefix code: the *Kraft-McMillan inequality.*

Theorem 1.2 (Kraft-McMillan) *Let $S = \{s_1, \dots, s_n\}$ be a source alphabet and $X = \{x_1, \dots, x_d\}$ a code alphabet. Let $l_1, \dots, l_n$ be a set of lengths. Then, a necessary and sufficient condition for the existence of a prefix-free code C over the alphabet X with codeword lengths $l_1, \dots, l_n$ is that*

$$\sum_{i=1}^{n} d^{-l_i} \leq 1,$$

where d is the size of the code alphabet.

In other words, if a set of lengths satisfies the inequality, then there exists at least one way to arrange the codewords into a prefix code.

- 1.3.2 - Optimal codes and the Shannon-Fano encoding.

The concept of average code length allows us to define an interesting class of codes: the *compact* (or optimal) codes. These are UD codes that have the lowest ACL.

A first attempt to achieve such codes was proposed by Shannon and *Robert Mario Fano*, who developed the so-called Shannon-Fano encoding.

Shannon-Fano encoding.

We will consider the binary case. The encoding itself is very simple: we order the symbols in decreasing order, divide the symbols into two sets such that the sum of probabilities in each set is almost equal, encode the symbols in the first set with 0 and the other with 1, and finally repeat the procedure recursively for each set.

Example

Consider the alphabet $\Sigma = \{a, b, c, d\}$ and let $p_a = 1/2, p_b = 1/8, p_c = 1/4$ and $p_d = 1/8$ Applying the steps above, we get the codes shown in the table below. Here, each color is used to show a different iteraction.

Symbol	**Codeword**
a	0
c	10
b	110
d	111

One can prove that Shannon-Fano encoding is not optimal. We present it just for hystorical reasons.

References.

[1] Claude E. Shannon. “A mathematical theory of communication”. In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423. DOI: `10.1002/J.1538-7305.1948.TB01338.X`.

Acronyms.

ACL	average code length. 5
UD	uniquely decodable codes. 3