Codes and Shannon's theorem

Diana-Maria Harambas

Spring 2024

This paper was written to support my presentation within the Undergraduate Seminar II course offered at Constructor University, and held by Dr. Keivan Mallahi-Karai.

ABSTRACT. The topics of codes and Shannon's theorem are part of greater mathematical areas called information theory and coding theory. In this paper there will be an introduction to codes, discussing the properties of unique decoding, block and instantaneous codes. These are tightly related to Kraft's inequality and McMillan's theorem - statements that will also be presented. After the introductory concepts, the focus will shift towards the ideas behind a theorem that Claude Shannon (the father of information theory) introduced and that targets the average string length of uniquely decodable codes. In the end, some examples of error detecting codes and Hamming codes (with a focus on calculating Hamming distances) are given.

Contents

1 Motivation and background

Computer systems, even if efficient in transmitting information, can suffer from data loss. This data loss is caused by disruptions in the communication channel or corrupted hard drives, for example. We need to think about ways to design optimal communication systems, that carry the maximum amount of correct information, and about the impact of noisy channels on the information. For that, we need to make use of the branches of mathematics called **information theory** and **coding theory**. In this section, we will introduce some necessary background on codes. [1] [2]

1.1 Introduction to codes. Unique decoding

Definition 1.1. *Let us take the finite set* $\boldsymbol{A} = \{a_1, a_2, ..., a_r\}$ *and call it a* ***code alphabet.*** $\boldsymbol{A}$ *has* r *elements and we call* r *the* ***radix*** *of the code. Denote by* $\boldsymbol{A^*}$ *the set of all words over* $\boldsymbol{A}$*. A subset* C *of* $\boldsymbol{A^*}$ *is called an* ***r-ary code*** *over* $\boldsymbol{A}$ *and the elements of* C *are* ***codewords.***

Definition 1.2 (Encoding function)**.** *An encoding function is a bijection* $f : S \longrightarrow C$*, where* $S = \{s_1, s_2, ..., s_q\}$ *is the source alphabet and* C *is a code.*

Let us take an example to understand these concepts better.

Example 1.3. *Let our source alphabet be* $S = \{a, b, c, \ldots, z\}$*, the Latin alphabet. We want to encode it using the code alphabet* $A = \{0, 1, 2, \ldots, 9\}$*. For a code* $C = \{00, 01, ..., 25\}$*, we can define the encoding function* $f : S \longrightarrow C$ *by:*

$$f(a) = 00,\ f(b) = 01,\ f(c) = 02,\ \ldots\ , f(x) = 23,\ f(y) = 24,\ f(z) = 25.$$

In the code above, notice that we took the words 00, 01, ..., 09 instead of just single-letter words 0, 1, ..., 9, which would naturally associate the source alphabet letters to "their number". To understand the reason, we shall discuss decoding.

Definition 1.4 (Unique decoding)**.** *Let* C *be a coding. We define the rule* C^* *to be the coding of source messages, which assigns to each word* $x_1, ..., x_m$ *in the source alphabet the word* $C^*(x_1...x_m) = C(x_1)C(x_2)...C(x_m)$ *obtained by concatenation.*

Definition 1.5. *The code* C *is* ***uniquely decodable/uniquely decipherable*** *if* C^* *is injective, i.e. no two different sequences of codewords represent the same string over the code alphabet.*

This means that if the following sequences of codewords

$$c_1c_2 \ldots c_n = d_1d_2 \ldots d_m$$

encode the same string, then $m = n$ and $c_1 = d_1, c_2 = d_2, \ldots c_n = d_n$.

Now, if we were to go back to our example with the Latin alphabet, we can see that, given a sequence of codewords, we can uniquely decode it by following a simple rule: grouping into codewords, i.e. grouping the letters of the code alphabet in pairs from the beginning (since all codewords of $C = \{00, 01, \ldots, 25\}$ have the same length 2).

We'll take the altered encoding version as a counterexample for unique decoding.

Example 1.6. *Let our source alphabet be* $S = \{a, b, c, \ldots, z\}$*, the Latin alphabet. We want to encode it using the code alphabet* $A = \{0, 1, 2, \ldots, 9\}$*. For a code* $C = \{0, 1, ..., 25\}$*, we can define the encoding function* $g : S \longrightarrow C$ *by:*

$$g(a) = 0,\ g(b) = 1,\ g(c) = 2,\ \ldots\ , g(x) = 23,\ g(y) = 24,\ g(z) = 25.$$

Now, we'll show that this is not uniquely decipherable by encoding the string "xyz" and then decoding it. "xyz" gets mapped to $g(x)g(y)g(z) = 232425$. *However,* $232425 = g(c)g(d)g(y)g(z)$, *for example. This means that it can be decoded to "cdyz", too. Thus, C is not uniquely decodable.*

1.2 Block codes. Instantaneous codes

The key difference between the two examples presented before comes from the length of the codewords. We saw that the equal length in the first example was crucial for the unique decoding, therefore we should introduce the term **block code.**

Definition 1.7 (Block code)**.** *A block code of length n is a code whose codewords all have the same length n.*

These types of codes don't need special symbols for separating characters and can be efficient when all characters have the same frequency. However, if some characters that need to be encoded are used more frequently than others, it is a disadvantage to keep the same length for the "usual" ones as for the "rare" characters. Hence, we want to introduce some new type of code that has variable length and can still be uniquely decipherable (after some imposed conditions): **instantaneous codes.**

Definition 1.8 (Instantaneous code)**.** *An instantaneous code is a coding such that no codeword is a prefix of another codeword, i.e. if a source symbol has a code* $c_1c_2 \dots c_n$, *then there is no other source symbol encoded as* $c_1c_2 \dots c_nc_{n+1} \dots c_m$.

Example 1.9. *Probably one of the most popular examples of instantaneous codes is the Morse code (seen in the figure below [2]). Here are some of its attributes:*

- *its code alphabet is* $A = \{\cdot, -,$ "*space*"$\}$
- *to decode it, one should look for separating characters: the first "space"*
- *in English, the frequency of E is higher than the frequency of F, hence it is efficient to encode it in a shorter sequence*

A	·—	N	—·
B	—···	O	———
C	—·—·	P	·——·
D	—··	Q	——·—
E	·	R	·—·
F	··—·	S	···
G	——·	T	—
H	····	U	··—
I	··	V	···—
J	·———	W	·——
K	—·—	X	—··—
L	·—··	Y	—·——
M	——	Z	——··

Back to the general case: let us discuss how to construct an $r-$ary instantaneous code of the source alphabet $\{s_1, s_2, ..., s_n\}$.

For such a construction, it is sufficient to specify the lengths $l_1, ..., l_n$ of the expected code words and assume $0 < l_1 \leq l_2 \leq ... \leq l_n$. Then, choose a random word $C(s_1)$ of length l_1. Next, choose a random $C(s_2)$ of length l_2 but avoid those which have $C(s_1)$ as the prefix. Now, we would have to check whether such a choice is possible or not. If it is, we continue inductively - always checking the prefix condition and making sure that a new choice of

codeword exists - until we reach our final encoding of s_n into $C(s_n)$.

With this construction algorithm in mind, we move on to the next section, where we will present some useful results on instantaneous codes and uniquely decipherable codes. We will continue using similar notations to [1] and follow the logic of proofs in [2].

2 Kraft's inequality. McMillan's theorem

2.1 Kraft's inequality

Theorem 2.1 (Kraft)**.** *There exists an instantaneous $r-ary$ code $C = \{c_1, c_2, ..., c_n\}$ with given codeword lengths of $l_1, l_2, ..., l_n$, whenever Kraft's inequality is satisfied:*

$$\sum_{i=0}^{n} \frac{1}{r^{l_i}} \leq 1.$$

Proof. In this proof, we will use the construction algorithm presented before. Let's start by assuming that the source alphabet $\{s_1, s_2, ..., s_n\}$, with the expected encoding $C(s_i) = c_i, \forall i \in \{1, 2, \ldots, n\}$, is presented in such order that the length of the codewords $\{c_1, c_2, \ldots, c_n\}$ is increasing:

$$0 < l_1 \leq l_2 \leq ... \leq l_n.$$

Choose c_1 of length l_1 arbitrarily. For our inductive construction, let's take the choice of c_2 as the induction base case. This is possible because $r^{l_2} > r^{l_2 - l_1}$ - the inequality is true as r^{l_2} represents the number of possible codewords of length l_2 and $r^{l_2 - l_1}$ is the number of codewords of length l_2 with a fixed prefix of length l_1.

Now, we want to show the following induction step: if $c_1, c_2, \ldots, c_k$ are chosen, then a choice for c_{k+1} of length l_{k+1} (such that the prefix condition holds) is possible.

Again, we will use a similar counting argument and note that, out of the $r^{l_{k+1}}$ possible r-ary codewords of length l_{k+1}, we need to eliminate those having any $c_i, i \leq k$ as a prefix. The number of codewords of length l_{k+1} with c_i as a prefix is $r^{l_{k+1} - l_i}$. Hence, we can choose c_{k+1} from

$$r^{l_{k+1}} - \sum_{i=1}^{k} r^{l_{k+1} - l_i}$$

possible codewords. This number is greater or equal to one, i.e., at least one such c_{k+1} exists, as Kraft's inequality implies.

$$\sum_{i=0}^{k+1} \frac{1}{r^{l_i}} \leq 1 \Longrightarrow \sum_{i=0}^{k+1} \frac{r^{l_{k+1}}}{r^{l_i}} \leq r^{l_{k+1}} \Longrightarrow \sum_{i=0}^{k+1} r^{l_{k+1} - l_i} \leq r^{l_{k+1}} \Longrightarrow$$

$$\Longrightarrow 1 + \sum_{i=0}^{k} r^{l_{k+1} - l_i} \leq r^{l_{k+1}} \Longrightarrow 1 \leq r^{l_{k+1}} - \sum_{i=1}^{k} r^{l_{k+1} - l_i}.$$

With this, the induction is complete and we were able to show that if the given lengths satisfy Kraft's inequality, then one can construct an instantaneous code with those lengths. QED

It is important to remark that the satisfaction of Kraft's inequality is necessary and sufficient for constructing instantaneous codes. This inequality can also be linked to unique decoding, via McMillan's theorem.

2.2 McMillan's theorem

Theorem 2.2 (McMillan)**.** *Every uniquely decipherable code satisfies Kraft's inequality.*

The theorem actually tells us that instantaneous codes are as efficient as uniquely decipherable codes and that for any uniquely decipherable one, we can find an instantaneous code with the same codeword lengths.

Proof. Let us use the same notations as in the previous proof:

- source alphabet $\{s_1, s_2, ..., s_n\}$;
- $r-$ary code $C = \{c_1, c_2, ..., c_n\}$ with given codeword lengths of $l_1, l_2, ..., l_n$;
- $C(s_i) = c_i, \forall i \in \{1, 2, \dots, n\}$;
- C uniquely decodable.

As before, we notice that we have a total number of r^m codewords of length m.

However, since C is uniquely decodable, we know that no two sequences can yield the same string of length m. Thus the number of sums $l_{i_1} + l_{i_2} + \cdots + l_{i_k} = m$, which is the same as the number sequences of codewords $c_{i_1} c_{i_2} \dots c_{i_k}$ with total string length $l_{i_1} + l_{i_2} + \cdots + l_{i_k} = m$, is at most r^m.

Our goal is to prove that

$$a = \sum_{i=1}^{n} \frac{1}{r^{l_i}} \leq 1,$$

and we will do so by showing that $\frac{a^m}{m}$ is bounded for any $m \in \mathbb{N}$.

We know that if a were to be greater than 1, then $\lim_{m\to\infty} \frac{a^m}{m} = \infty$, therefore obtaining a bounded sequence of positive numbers $(\frac{a^m}{m})$ is equivalent to stating that $a \leq 1$.

Let's compute the powers of a :

$$a^2 = (\sum_{i=1}^{n} \frac{1}{r^{l_i}}) \cdot (\sum_{j=0}^{n} \frac{1}{r^{l_j}}) = \sum_{i,j=0}^{n} \frac{1}{r^{l_i + l_j}},$$

$$\vdots$$

$$a^m = \sum_{i_1, i_2, \dots, i_m = 1}^{n} \frac{1}{r^{l_{i_1} + l_{i_2} + \cdots + l_{i_m}}}$$

We can reorder the elements of the general sum in a^m such that they are grouped by $\frac{1}{r^j}$ as follows:

$$a^m = \sum_{i_1, i_2, \dots, i_m = 1}^{n} \frac{1}{r^{l_{i_1} + l_{i_2} + \cdots + l_{i_m}}} = \sum_j \Big(\sum_{l_{i_1} + l_{i_2} + \cdots + l_{i_m} = j} \frac{1}{r^{l_{i_1} + l_{i_2} + \cdots + l_{i_m}}} \Big) = \sum_j \Big(\sum_{l_{i_1} + l_{i_2} + \cdots + l_{i_m} = j} \frac{1}{r^j} \Big),$$

where j is bounded from above by ml with $l = \max(l_1, l_2, \dots, l_n)$.

Since there are at most r^j sums of $l_{i_1} + l_{i_2} + \cdots + l_{i_m} = j$, the inequality below holds

$$a^m = \sum_j \Big(\sum_{l_{i_1} + l_{i_2} + \cdots + l_{i_m} = j} \frac{1}{r^j} \Big) \leq \sum_{j=1}^{ml} \frac{r^j}{r^j} = ml.$$

Thus $\frac{a^m}{m} \leq l$ for any natural number m, and therefore $a \leq 1$, i.e. Kraft's inequality is satisfied. QED

3 Shannon's theorem

The next part of this paper delves deeper into information theory. We will introduce the concept of **information entropy** [3] [4] and then present versions of **Shannon's theorem** [1] [4] [5].

3.1 Information entropy

In information theory, we are concerned with understanding the concept of **information** from a mathematical point of view. Therefore, we need to define a way to quantify it.

Let's say an event x has probability p_x of occurring. Then the quantity of information $I : (0, 1] \to [0, \infty)$ associated with x is defined as

$$\boldsymbol{I(p_x) = -\log_2(p_x)}.$$

While at first glance this quantity function seems oddly chosen, we can informally describe the intuition behind it. Let's consider I as a measure of "surprise" and, naturally, say that I is high if we gain more information than expected ("big surprise"). We want this quantity to be represented by a function that has the properties:

- I is **continuous** (a small change in the probability of an event yields a small change in the information quantity)
- I is **non-negative** (in case "we are not surprised at all", then $I = 0$, but otherwise, we still gain an amount of "surprising information")
- I is **monotonically decreasing** (if two events x, y have probabilities $p_x < p_y$, then $I(p_x) > I(p_y)$, since we are more "surprised" of the amount of information received from the event with a smaller probability of happening)
- $I(1) = 0$ and $\lim_{p\to 0} I(p) = \infty$ (clearly, if we are sure that an event is happening, we won't be "surprised" by the information we receive, whereas if an event has a probability close to 0, I will tend to infinity)
- I is **additive** (given two independent events, the amount of "surprising" information gained will, naturally, be the sum of the individual amounts)

All these properties are satisfied by only one function, $\boldsymbol{I(p_x) = -\log_2(p_x)}$. The choice of logarithm base does not matter - choosing something that is not 2 will just end up scaling everything by the same constant.

Now that I was introduced, we can finally define the useful notion of **information entropy.**

Definition 3.1 (Information entropy)**.** *In a probability space, given the random variable X with probability mass function P, the entropy of X is calculated as the expected value:*

$$H(X) = \mathbb{E}_{\mathbb{X}}[I(P(X))] = -\sum_{x\in\mathcal{X}} P(X = x)\log_2(P(X = x)).$$

As a remark, high entropy is equivalent to the random variable being closer to being uniform, i.e. having equally likely outcomes.

With this definition in mind, we can continue with Shannon's source coding theorem, which will showcase the meaning and usage of entropy.

3.2 Shannon's theorem

As mentioned in the abstract of this paper, Claude Shannon is considered to be the father of information theory, having discovered several results on coding. We will focus on understanding **Shannon's noiseless coding theorem** also known as **Shannon's source coding theorem.** First, let's state an informal version of this theorem, and then state and prove a second version.

Theorem 3.2 (Shannon - version 1, informal)**.** *If X is encoded as a collection of binary strings which can be uniquely decoded, then the average string length is at least $H(X)$.*

Theorem 3.3 (Shannon - version 2)**.** *Given $X = \{x_1, ..., x_n\}$ and its associated probabilities $\{p_1, ..., p_n\}$, it is always possible to design a source encoder to find a uniquely decodable binary code C such that*

$$H(X) \leq E[|C(X)|] \leq H(X) + 1.$$

Proof. For any symbol x_k, pick a code with length l_k such that l_k is the smallest integer for which

$$I(X = x_k) \leq l_k < I(X = x_k) + 1.$$

Such natural numbers l_k can be found and they will satisfy Kraft's inequality (and therefore McMillan's theorem on unique decoding) because

$$I(X = x_k) \leq l_k < I(X = x_k) + 1 \iff -\log_2 p_k \leq l_k < -\log_2 p_k + 1 \iff$$

$$\iff \frac{1}{p_k} \leq 2^{l_k} \iff \frac{1}{2^{l_k}} \leq p_k \iff$$

$$\iff \sum_{k=1}^{n} \frac{1}{2^{l_k}} \leq \sum_{k=1}^{n} p_k = 1,$$

which is Kraft's inequality for binary codes. So, we can find a uniquely decipherable binary code C with such lengths. It remains to verify the inequality given in the theorem statement.

Back to our $I(X = x_k) \leq l_k < I(X = x_k)+1$, by multiplying everything with the associated probability p_k, we obtain:

$$I(X = x_k)p_k \leq l_k p_k < I(X = x_k)p_k + p_k,$$

which, by summing everything up, leads to

$$H(X) \leq E[|C(X)|] < H(X) + \sum_{i=1}^{n} p_i = H(X) + 1,$$

and with this, the theorem is proven. QED

This theorem can be generalized to r-ary uniquely decipherable codes, but in that case, one has to use the r-ary entropy

$$H_r(X) = -\sum_{x \in \mathcal{X}} P(X = x) \log_r(P(X = x)) = \frac{H(X)}{\log_2 r}.$$

4 Error correcting codes

The final part of this paper targets error correcting codes and introduces Hamming codes, which are considered to be probably the most famous error correcting codes and which were discovered around 1950.

4.1 Methods for protection against data loss

As computer and information systems are prone to data loss, it is important to find ways to protect them against it, by detecting and correcting errors. [6] Note that in this section, we will use the term **bit** (0 or 1 digit) for referring to the unit of data stored by the computer.

We can start with the simplest method of detecting corrupted bits, **redundancy,** which is **repeating each bit** instead of storing it just once. Even if this is good enough for small errors, it is inefficient in terms of memory and, in case of huge errors (for example, wiping out all copies of certain bits), it ends up being unfortunately useless.

A slightly improved method is using so-called **checksums**. Let's take two examples.

Example 4.1. *Our goal is to transmit three bits b_1, b_2, b_3 correctly, so we will use an additional bit for our transmission $b_4 = b_1 \oplus b_2 \oplus b_3$, where $\oplus$ is the XOR logical operator (for bits x, y, $x \oplus y = 0$ if $x = y$ and 1 otherwise). This b_4 checksum is called the parity of b_1, b_2, b_3 because $b_1 \oplus b_2 \oplus b_3 = (b_1 + b_2 + b_3)(mod\ 2)$.*

Assume that one of the bits b_1, b_2, b_3 is corrupted (it gets flipped). Then, b_4 will be incorrect since the parity would need to change. Thus, an error can be detected.

Now assume that two of the bits b_1, b_2, b_3 get flipped. This time, the new $b_1 \oplus b_2 \oplus b_3$ will have the same parity as b_4 and therefore won't be able to detect if two bits are erroneous.

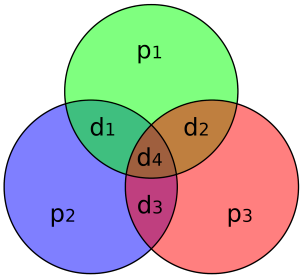
Example 4.2. *We saw that the previous example could not detect errors in the case of two corrupt bits. Let's enhance this method by taking better checksums. This time, we will use 7 bits:*

- *b_1, b_2, b_3 for the desired bits*
- *$b_1 \oplus b_2$, $b_1 \oplus b_3$, $b_2 \oplus b_3$*
- *$b_1 \oplus b_2 \oplus b_3$*

Now, a corrupted bit will change three of the parity bits, detecting an error. Two flipped bits will change two of the parity bits, so the existence of an error can be signalled.

These were some particular examples, but we can actually extend these methods to work with **generalized checksums**. If we encode k bits using 2^k checksums by taking the parity of all possible subsets, we will be able to actually protect the data even if almost half of the bits are corrupted.

Because we are getting more and more familiar with parity bits, we can now make the transition to **Hamming codes**. In the diagram to the right, four data bits are represented (d_1, d_2, d_3, d_4) alongside three parity bits: $p_1 = d_1 \oplus d_2 \oplus d_4$, $p_2 = d_1 \oplus d_3 \oplus d_4$, and $p_3 = d_2 \oplus d_3 \oplus d_4$. Called the **Hamming(7,4)**, it has the goal of not only detecting single-bit errors but also correcting them. Because of the overlapping parity bits, this is possible.



4.2 Hamming codes

Hamming codes are linear error correcting codes, and informally speaking, they can be seen as "upgraded" parity codes that can also correct erroneous bits. As discussed briefly in the last example from **Section 4.1**, Hamming codes are designed to detect and correct single-bit errors.

Now, let's introduce some properties that these codes have, starting with the definition of **Hamming distance**. [7]

Definition 4.3 (Hamming distance)**.** *The Hamming distance is a metric on a code C given by*

$$d(u, v) = |\{i \in \mathbb{N} \mid u_i \neq v_i\}|,$$

for $u = (u_1, ..., u_s)$, and $v = (v_1, ..., v_s)$.

Take $\boldsymbol{d = \min d(u, v)}$ for all distinct $u, v \in C$. The following propositions hold for Hamming codes:

Proposition 4.4. *Any code can detect up to $d - 1$ errors.*

Proposition 4.5. *Any code can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors.*

These definitions and properties are enough to be able to solve simple problems, like the following example.

Example 4.6. *Consider the binary code consisting of the four codes:*

$$c_1 = 00000, c_2 = 01011, c_3 = 10101, c_4 = 11110.$$

We wish to calculate the maximum number of erroneous bits that can be corrected.

First, we observe that the Hamming distance between two such codes u, v is just the number of positive bits in $u \oplus v$. For calculating all Hamming distances, we perform all possible $c_i \oplus c_j$ (example: $c_2 \oplus c_3 = 01011 \oplus 10101 = 11110$, hence $d(c_2, c_3) = 4$) and obtain:

$$\begin{array}{lll} d(c_1, c_2) = 3, & d(c_1, c_3) = 3, & d(c_1, c_4) = 4, \\ d(c_2, c_3) = 4, & d(c_2, c_4) = 3, & d(c_3, c_4) = 3. \end{array}$$

Therefore, the minimum distance is $d = 3$ and the code can correct at most $\lfloor \frac{d-1}{2} \rfloor = 1$ erroneous bit.

5 Conclusion

To conclude, in this paper we were introduced to the basics of coding theory and information theory, by starting with definitions of types of codes and Kraft's and McMillan's theorem, then advancing to Shannon's source coding theorem. We then explored ideas of error detecting methods and ended with examples and properties of Hamming codes - one of the most studied error correcting codes. In the future, one can use the concepts in this paper to further study other results from information theory, such as Huffman encoding, or delve deeper into other methods for error detection and correction, like Reed-Muller codes, and Golay codes.

References

[1] Steven Roman. *Introduction to coding and information theory.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1997.

[2] Jiří Adámek. *Foundations of coding.* A Wiley-Interscience Publication. John Wiley & Sons, Ltd., Chichester, 1991. Theory and applications of error-correcting codes with an introduction to cryptography and information theory.

[3] Matthew N. Bernstein. Foundations of information theory.

[4] Yuval Wigderson. Claude Shannon, Master of Uncertainty, 2019.

[5] Chen Zhe. Shannon's theory and some applications. 2018.

[6] Sanjeev Arora. Protecting against Information Loss: Coding Theory. 2016.

[7] Andrew Krieger. Ulam's Game and Error Correcting Codes, 2019.

Websites that were used to understand the topic better and add examples:
Wikipedia. Hamming code
Hamming Code - Solved Problems - Example 4.6
Wikipedia. Hamming(7,4) - image source in Section 4.1