Basic coding and information theory

Emmanuela Orsini

CoCoNut
February 15, 2016

Notes on the course

- Recommended references
  - Chapters 1-3: Standard references for coding theory are : [20], [9], [15], [13], [10], [16]. Other useful references are: [6], and [1].
  - Chapter 4: For this part of the course we mainly follow: [3], [7], [9], [18], [16].
  - Chapter 5: This chapter is mainly from [4], [5] and [2].
  - Chapters 6-7: For this part of the course we mainly follow: [13], [9], [10], [16], [18].
  - Chapter 8: [11], [18].

- Prerequisites: Linear Algebra, Basic Probability and some familiarity with the basic concepts of groups, rings and fields

- What is this course about?: Suppose Alice secretly thinks of a number between 1 and a million. You can ask questions to which Alice can only answer YES or NO. What is the minimal number of questions you need to ask in order to find out this number? What if Alice is allowed to tell one lie? What is a winning strategy in this case?

  Information and coding theory provide the answers to this questions respectively. The topics covered in this course underlie the modern digital world, ranging from applications that you encounter in your daily lives such as interactions on social networks, video-streaming on the internet or listening to mp3 music on your portable devices, to sending commands to a space probe.

- Learning outcomes: At the end of the course you will be able to: compute the entropy of simple channels; understand the effectiveness of codes based on bounds and compute parameters like minimum distance and cardinality of a code; apply basic knowledge on algebra to construct codes; compute the complexity of simple decoding algorithms and apply these algorithms to small examples.

Contents

1 Error correcting codes: Introduction

In this chapter

- Motivation
- Block codes: notation
- Hamming distance
- Maximum Likelihood Decoding

1.1 Introduction

Coding theory studies the problem of reliably transmitting information through a communication channel. A channel can be thought as a physical medium linking a transmitter and a receiver which are spatially separated (e.g. a telephone line, a wireless link) or perhaps separated in time (e.g. a storage device). Typically, in a channel the transmitted message gets distorted by noise and the receiver needs to perform some operations in order to obtain an estimation of the transmitted message. Aim of coding theory is to find ways to eliminate or sharply reduce the quantity of errors during transmission.

Historically, Coding Theory (and more broadly Information Theory) was initiated by two seminal papers:

- In 1948, Shannon wrote “A Mathematical Theory of Communication", which marked the beginning of both Information and Coding Theory ([17]);
- In 1950, Hamming wrote “Error Detecting and Error Correcting Codes", which was the first paper explicitly introducing error-correcting codes ([8]).

Following Shannon's paper, we can consider the problem of two parties, typically Alice and Bob, that wish to communicate over a certain *channel*. Shannon considered both “noiseless” and “noisy” channel. In this course we first consider noisy channels. Examples of noisy channels are telephone lines, storage devices, fiber-optic cables, radio link, etc; and the noise can be caused by atmospheric conditions, human errors, electrical malfunction, etc. Whenever data is transmitted over a noisy channel, there is some probability that the received message will not be identical to the transmitted message. One fundamental problem in coding theory is to determine what message was sent on the basis of what is received.

Two commonly considered noisy (discrete) channels are:

- **Binary Symmetric Channels (BSC)**: Let $\mathcal{X} = \{0, 1\}$ be the input alphabet and $\mathcal{Y} = \{0, 1\}$ the output alphabet, this channel is parametrized by the probability p, $0 \leq p < 1/2$, that an input bit is flipped. p depends on the noise level and is called *crossover probability*.
- **Binary Erasure Channels (BEC)**: Let $\mathcal{X} = \{0, 1\}$ be the input alphabet and $\mathcal{Y} = \{0, 1, e\}$ the output alphabet. In this type of channel bits do not get flipped, but erased with probability p (*erasure probability*) depending on the noise level. (Figure 1.1)

BSC, and BEC, are examples of *discrete memoryless channels* (DMC), channels in which inputs and outputs are discrete and the probability of error,resp. erasure, in a bit is independent of previous bit.

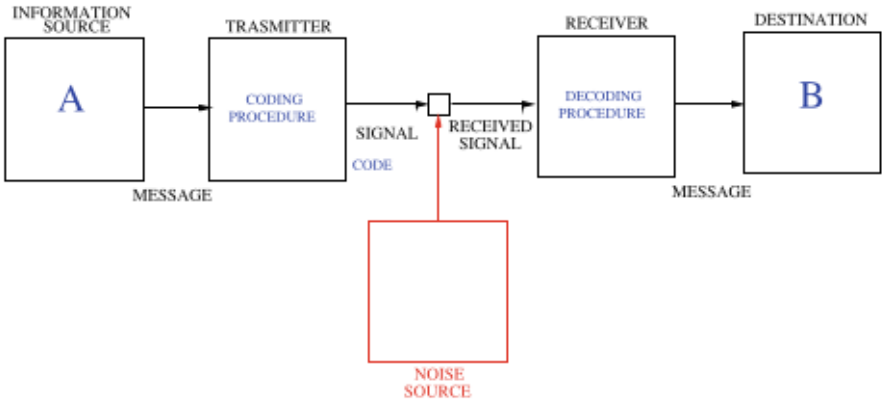
To deal with this situation information theory and coding theory offer a solution: adding some kind of redundancy to the message x that Alice (A) wants to send to Bob (B).

Following Figure 1.2, A hands the message x to a device called a *transmitter* that uses a *coding procedure* to obtain a longer message c that contains redundancy. The transmitter sends c through the channel to another device called a *receiver*. Because of the noise in the channel, it may be that the message r, obtained after the transmission, is different from c. If the occurred errors are not too many (in a sense that will be clear later), the receiver is able to recover the original message x, using a *decoding procedure*.

Information theory mainly deals with the theoretical limitations of such a solution; while coding theory deals with the study and construction of practical encoding and decoding schemes.

Figure 1.1: Transition diagrams representing binary symmetric channel and binary erasure channel

Figure 1.2: A communication schema



We will begin by introducing error-correcting codes, essentially following Hamming's approach.

1.2 Error-correcting codes

What are codes? Generally speaking a code is simply a set of strings over a certain alphabet.

One of the most commonly used code is the International Standardized Book Number (ISBN) Code. Every book is assigned an ISBN, and that ISBN is typically displayed on the back cover of the book. An ISBN is a 10-digit number, for example, the ISBN for "The Theory of Error-Correcting Codes" by MacWilliams and Sloane ([15]) is $0-444-85193-3$. The first nine digits $0-444-85193$ contain information about the book, for example the first digit indicates the language of the book, which in this case is 0 for English. The last digit, "3", is a check digit which is chosen on the basis of the first nine, and it is designed to detect errors. In general, the check digit a_{10} for the ISBN $a_1a_2a_3a_4a_5a_6a_7a_8a_9$ is chosen by computing $a_{10} = (a_1 + 2a_2 + \cdots + 9a_9)$. If $a_{10} \equiv i \pmod{11}$ for some i with $0 \leq i \leq 9$, we set $a_{10} = i$. If $a_{10} \equiv 10 \pmod{11}$, we set a_{10} to be the symbol "X". The point is that every book is assigned an ISBN using the same system for choosing a check digit, and so, for example, if one makes a mistake when typing in this number, the computer can be programmed to catch this error. The ISBN Code is a very simple code. It is not hard to see that it detects all single-digit errors (a mistake is made in one position) and all transposition errors (the numbers in two positions are flipped). It cannot correct any single-digit or transposition errors, but this is not a huge liability, since one can easily just type in the correct ISBN (re-send the message) if a mistake of this type is made.

Further, the ISBN code is efficient, since only one non-information symbol needs to be used for every nine-symbol piece of data.

More generally, we can consider digital data as a series of ones and zeros, $1011001101\ldots$. A very simple way to encode this data consists of bundling the original message into a group of bits, for example we can use groups of $4, 8, 32, 64$ bits at time, and then adding a **parity bit** in order to detect errors.

Example 1.1. *Suppose we want to send strings of* 4 *bits, i.e.* $m_1m_2m_3m_4 \in \{0,1\}^4$. *Then we add an extra bit,* m_5, *so that the total number of* 1 *in each resulting string is even, as follows:*

$$\begin{aligned} 0110 &\longrightarrow 0110\mathbf{0} \\ 1111 &\longrightarrow 1111\mathbf{0} \\ 1000 &\longrightarrow 1000\mathbf{1} \\ 1011 &\longrightarrow 1011\mathbf{1} \end{aligned}$$

The resulting string of 5 bits is called a **codeword**.
It is easy to see that now, when we receive a string of bits, we can detect whether a *single* error occurred. For example, using the previous example, we have:

- 10101 Errors (why?)
- 01110 Errors
- 11101 No errors
- 11111 Errors
- 00000 No errors
- 00001 Errors

The problem with this type of encoding is that we do not know where the errors are; more formally we are able to **detect** if one error occurs, but we are not able to find **error locations**, i.e. we can not **correct** errors. At this point, when an error is detected, we could simply ask for a retransmission, but often this solution is really expansive or even impossible.

Also, notice that when an even number of bits are flipped, the errors are not detected. For example, if 11101 is the transmitted codeword and $\mathbf{r} = \mathbf{0}1\mathbf{1}1\mathbf{1}$ is received then, as the parity of $\mathbf{r}$ is even, errors can not be detected.

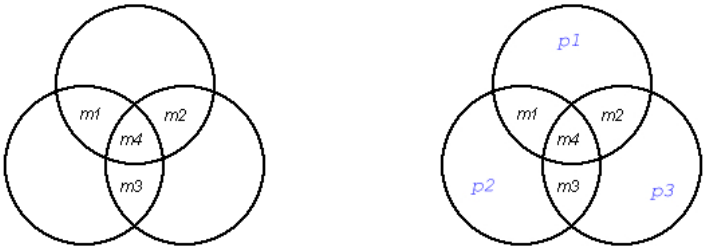
Example 1.2 (Hamming code). *An improvement of the previous example is given by Hamming. The high level idea of Hamming encoding is to use multiple parity-check bits (and not just one), in such a way that each parity-check bit is a parity bit of a group of bits in the resulting codeword.*

Suppose we have a message $\mathbf{m} = m_1m_2m_3m_4 \in \{0,1\}^4$ *of* 4 *bits as before. Put them in the middle of a* Venn-diagram *of three intersecting circles as given in Figure 1.3. Complete the three empty areas of the circles according to the rule that the number of ones in every circle is even. (See Sage 1.13).*

In this way we get 3 *redundant bits* p_1, p_2, p_3. *The first redundant bit* p_1 *provides an even parity for the three information bits* m_1, m_2, m_4, p_2 *is the parity of* m_1, m_3, m_4, *and* p_3 *is the parity bit of information bits* m_2, m_3, m_4. *Then we obtain* $\mathbf{c} = c_1c_2c_3c_4c_5c_6c_7 \in \{0,1\}^7$, *where*

- $c_1 = m_1$

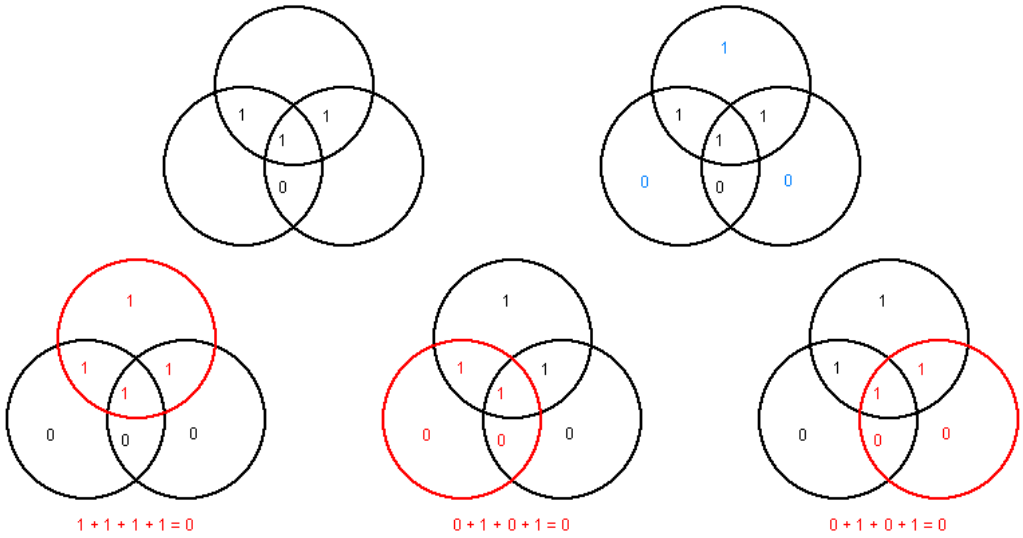
Figure 1.3: Construction of the binary Hamming code of length 7



- $c_2 = m_2$
- $c_3 = m_3$
- $c_4 = m_4$
- $c_5 = m_1 \oplus m_2 \oplus m_4 = p_1$
- $c_6 = m_1 \oplus m_3 \oplus m_4 = p_2$
- $c_7 = m_2 \oplus m_3 \oplus m_4 = p_3$.

In each block of 7 *bits the receiver can detect 2 errors and correct one error, since the parity in every circle should be even. So, once a word is received, if the parity in each circle is even we declare the circle correct; if the parity is odd we declare the circle incorrect, and conclude that the error is in the incorrect circles and in the complement of the correct circles. We see that every pattern of at most one error can be corrected in this way.*
For instance, let $\mathbf{m} = 1101$ *be the message,* $\mathbf{p} = 100$ *the redundant information added, and* $\mathbf{c} = 1101100$ *the codeword sent.*

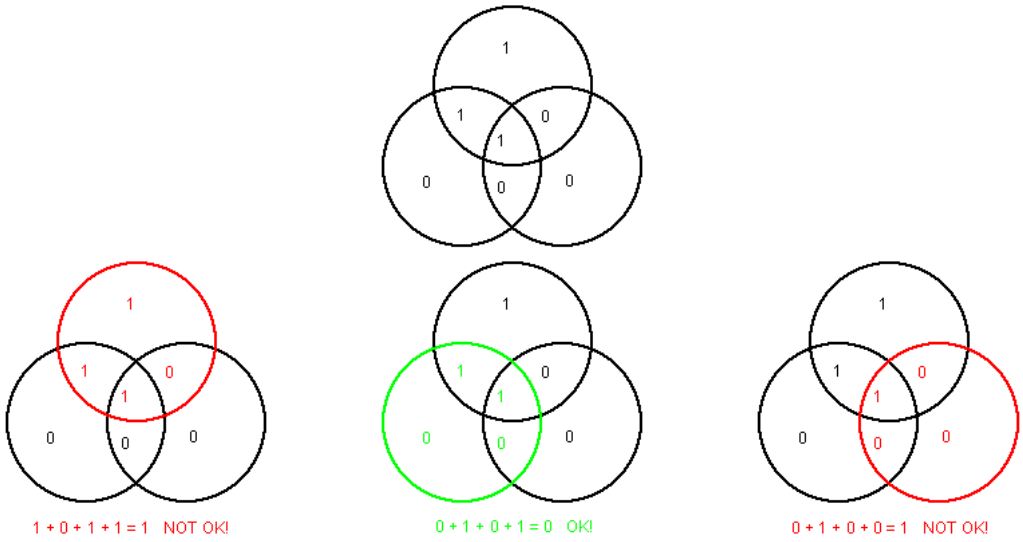
Figure 1.4: A communication without errors



Suppose that after transmission one symbol is flipped and $\mathbf{r} = 1001100$ *is received, as shown in Figure 1.5. Then we conclude that the error is in the left and upper circle, but not in the right one. And we correctly conclude that the error is at* m_2.

We can now suppose that 2 *errors have occurred, and for instance the word* $\mathbf{r}_0 = 1001000$ *is received. Then the receiver would assume that the error occurred in the left circle and not in the upper circle, and would therefore conclude that the transmitted codeword was* 1001001. *Hence the decoding scheme creates an extra error.*

Figure 1.5: One-error detection and correction



We can represent Hamming encoding as a map $\mathsf{Enc} : \{0,1\}^4 \to \{0,1\}^7$, *that maps each of the* 2^4 *strings of* 4 *bits into a codeword. The resulting codewords are listed in Table 1.1.*

Table 1.1: List of all codewords Example 1.2

Information bits	Codeword	Information bits	Codeword
0000	0000000	1000	1000110
0001	0001111	1001	1001001
0100	0010101	1010	1010101
0011	0011100	1011	1011010
0010	0010011	1100	1100011
0101	0101010	1101	1101100
0110	0110110	1110	1110000
0111	0111001	1111	1111111

1.2.1 Notation

Here we formalise some of the concepts we have seen in the previous section.

Codes can be divided into two general types: *fixed length codes* and *variable length codes*.

In this lecture we restrict to fixed length or **block codes**, i.e. the message that is sent have a fixed length of k symbols and the encoded words have a fixed length of n symbols, both from the same alphabet $\mathcal{A}$. As previously mentioned, we add redundant symbols to the message in order to detect and/or correct errors.

More precisely:

Definition 1.2.1. We consider codes over an alphabet $\mathcal{A}$ of cardinality q (i.e. $|\mathcal{A}| = q$). If $q = 2$ then the codes are called *binary codes*. Let $\mathcal{A}^n$ be the set of all n-tuples $\mathbf{c} = (c_0, \ldots, c_{n-1})$ with entries $c_i \in \mathcal{A}$. A (*block*) *code* C of length n ($k \leq n$) over $\mathcal{A}$ is a non-empty subset of $\mathcal{A}^n$.

The elements of C are called *codewords*. If C contains M codewords, then M is the *size* of the code, and we denote such a code by (n, M)-code.

Example 1.3. *The code in Example 1.2 is the* $(7, 16)$ *binary Hamming code.*

For an (n, M)-code the value $n - \log_q(M)$ is the *redundancy* (and the value $k = \log_q(M)$ is the *message length* or *information length*) and $R = \log_q(M)/n$ is the *information rate*.

Associated with a code there is also an **encoding map** Enc, i.e. an injective map from the message set $\mathcal{A}^k$ into the set $\mathcal{A}^n$, such that the code is the image of this map.

Example 1.4. *The information rate for the binary* $(7, 16)$ *Hamming code is* $4/7$*, and* $\mathsf{Enc} : \{0,1\}^4 \to \{0,1\}^7$.

1.2.2 Hamming distance and minimum distance of a code

Here we introduce an important invariant of a code.

Definition 1.2.2. Given two strings $\mathbf{x}$ and $\mathbf{y} \in \mathcal{A}^n$, the **Hamming distance** between $\mathbf{x}$ and $\mathbf{y}$, denoted $\mathrm{d}(\mathbf{x}, \mathbf{y})$, is the number of coordinates where $\mathbf{x}$ and $\mathbf{y}$ differ. The (Hamming) **minimum distance of a code** C is given by

$$d(C) = min\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}\},$$

i.e. the minimum Hamming distance between two distinct codewords of C.

Example 1.5. *Let* $\mathbf{v} = 0011$ *and* $\mathbf{w} = 1011$*, then* $d(\mathbf{v}, \mathbf{w}) = 1$.

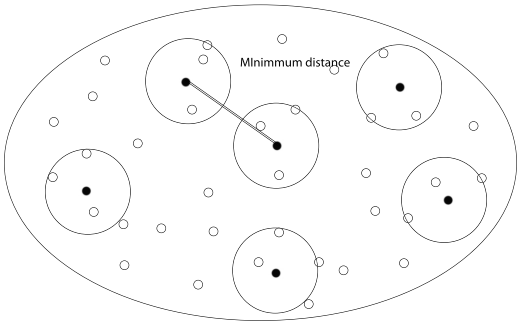
Exercise: Prove that the Hamming distance is a metric. In other words, show that $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{A}^n$, we have:

a. $\mathrm{d}(\mathbf{x}, \mathbf{y}) \geq 0$, with $\mathrm{d}(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$

b. $\mathrm{d}(\mathbf{x}, \mathbf{y}) = \mathrm{d}(\mathbf{y}, \mathbf{x})$ (symmetry)

c. $\mathrm{d}(\mathbf{x}, \mathbf{z}) \leq \mathrm{d}(\mathbf{x}, \mathbf{y}) + \mathrm{d}(\mathbf{y}, \mathbf{z})$ (triangle inequality)

Figure 1.6: Minimum distance



Definition 1.2.3. The **Hamming weight** of a string $\mathbf{x}$ over an alphabet $\mathcal{A}$ is defined as the number of non-zero symbols in the string, and denoted by $\mathrm{wt}(\mathbf{x})$.

Exercise: Prove that in the binary case $\mathrm{d}(\mathbf{v}, \mathbf{w}) = \mathrm{wt}(\mathbf{v} + \mathbf{w})$.

Example 1.6. *If* $\mathbf{v} = 21201 \in \mathbb{Z}_3^5$ *and* $\mathbf{w} = 21002 \in \mathbb{Z}_3^5$, *we have*

$$\mathrm{wt}(\mathbf{v}) = 4, \quad \mathrm{wt}(\mathbf{w}) = 3, \quad d(\mathbf{v}, \mathbf{w}) = 2.$$

Exercise: Compute the weight of the following words in $\mathbb{Z}_2^7$, and the distance between each pair of them:

1. $\mathbf{v}_1 = 1001010$
2. $\mathbf{v}_2 = 0110101$
3. $\mathbf{v}_3 = 0011100$
4. $\mathbf{v}_2 + \mathbf{v}_3$

Example 1.7.

- *The minimun distance of the binary* $(7, 16)$ *Hamming code is* 3 *(see Table 1.1) (why?).*
- *The distance of* $C = \{00000, 11110, 01111, 10001\}$ *is* 2 *(why?).*

Exercise: Find the minimum distance of the code

$$C = \{00000, 01110, 10011, 11111\}.$$

Why are the size and the distance of a code important? Suppose C is an (n, M) code with information length k. It is clear that we want k large with respect to n, so that we are not transmitting a lot of redundancy symbols. This makes the code efficient.

On the other hand, the minimum distance $d(C)$ of a code measures the error-resilience of C. Roughly speaking, suppose that a codeword $\mathbf{c} \in C$ is transmitted after encoding: if the decoder

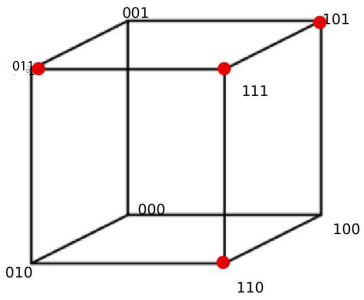
receives a codeword, then it decides that not disturbance has occurred and no correction is needed; but if it receives a string $\mathbf{y} \in \mathcal{A}^n$, such that $\mathbf{y} \notin C$, then the decoder has to find the codeword $\mathbf{c}$ which has been sent by the encoder. There are different ways by which the decoder can make this choice, a possibility is that of finding, among all codewords, the one which has the *highest probability* to have been sent. More precisely, the decoder will find the codeword $\mathbf{c}$ that maximizes the probability that $\mathbf{r}$ is received given that $\mathbf{c}$ has been sent. Such a decoder is called a *maximum likelihood decoder*. We will see that, for some types of channels, this is equivalent to find the codeword $\mathbf{c}'$ *closest* to $\mathbf{y}$ in the Hamming distance (*nearest neighbor decoding*).

Let $\mathbf{x} \in \mathcal{A}^n$ and $t \in \mathbb{N}$, define $\mathcal{B}_t(\mathbf{x})$ to be the ball of radius t centered at $\mathbf{x}$. In other words, $\mathcal{B}_t(\mathbf{x})$ is the set of all vectors in $\mathcal{A}^n$ which are Hamming distance at most t away from $\mathbf{x}$:

$$\mathcal{B}_t(\mathbf{x}) = \{\mathbf{y} \in \mathcal{A}^n \mid d(\mathbf{x}, \mathbf{y}) \leq t\}.$$

Example 1.8. *In Figure 1.7 we show the set $\mathbb{Z}_2^3$. The elements in the ball $\mathcal{B}_1(111)$ in $\mathbb{Z}_2^3$ are shown as red solid dots. (See Sage1.11).*

Figure 1.7:



Since C has minimum distance d, two balls of radius $\lfloor \frac{d-1}{2} \rfloor$ centered at distinct codewords can not intersect. Thus, if at most $\lfloor \frac{d-1}{2} \rfloor$ errors are made in transmission, the received word will lie in a unique ball of radius $\lfloor \frac{d-1}{2} \rfloor$, and that ball will be centered at the correct codeword. In other words, a code of minimum distance d can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors. In particular this means that we want d large with respect to n.

More formally, we have the following definitions:

- The *error detection capability* of a code C is the number of errors that the code can detect, i.e. C is *e-error detecting* if, under the hypothesis than no more than e errors occur during the transmission, it is always possible to detect whether errors have been occurred or not. e is the largest integer with this property. A *e-error detecting* code has minimum distance $d = e + 1$.
- The *error correction capability* of a code C is the number of errors that the code can correct, i.e. C is *t-error correcting* if, under the hypothesis than no more than t errors occur during the transmission, it is always possible to determine which components of the received string are in errors, and correct them. t is the largest integer with this property. A *t-error detecting* code has minimum distance d such that $d = \lfloor \frac{t-1}{2} \rfloor$.

Summing up, as long as the received word is not a codeword, nearest neighbour decoding will decode it to some codewords, but the receiver has no way of knowing whether that codeword is

the one that was actually sent. The receiver only knows that, if C is a t-error correcting code, and no more than t errors were introduced, then the decoding will uniquely recover the correct codeword.

Example 1.9. *a) Let $C = \{00, 01, 10, 11\}$. It is trivial to see that every received word is a codeword, so C cannot detect any errors.*

b) We can modify C in a) by adding a third bit to each codeword so that the number of 1's in each codeword is even, as in Example 1.1. The resulting code is

$$C_1 = \{000, 011, 101, 110\}.$$

Suppose that 010 is received, then since this is not a codeword, the code C_1 can detect that an error has occurred. Each of the codewords $110, 011, 000$ has distance 1 from 010, so the code cannot decode univocally. In general, the parity code, of any length, is error detecting, but not error correcting.

Example 1.10. *The error correction capability of the binary Hamming code $(7, 16)$ is 1 and its error detection capability is 2, as $d = 3$. Indeed we have seen in Example 1.2 that C can detect any error of weight ≤ 2 and correct any error of weight 1.*

Exercise: Let C be the code $\{00000000, 11111000, 01010111, 10101111\}$. How many errors can C detect and how many can it correct?

Exercise: Let $C = \{0000000000, 1111100000, 1111111111\}$. Suppose that 0000000000 was sent and that $\mathbf{y} = 1110000000$ was received. The closest codeword to $\mathbf{y}$ is 1111100000, so the decoding outputs the wrong codeword. Why?

The distance d of a code C can be also used to determine the *erasure correction capability* of C. We note that erasures (recall "erasure" means that a transmitted symbol is unreadable and at its place an extra symbol ϵ is introduced) are different from errors, in that locations of errors are unknown, whereas the locations of erasures are known. Suppose $\mathbf{c} \in C$ is a transmitted codeword and $\mathbf{r}$ is the received word, containing τ erasures. Intuitively it is clear that the decoder can not correct $\mathbf{r}$ with the correct codeword if $\tau > d$, because there may be a codeword other than $\mathbf{c}$ closer to $\mathbf{r}$. In general, it can be proved that:

- a code C with distance d can correct s erasures if and only if $d - 1 \geq s$;
- the condition for simultaneous correction of t errors and s erasures is

$$d \geq 2t + s + 1.$$

Exercise: **[Repetition Code]** The simplest way to encode information in order to recover it in the presence of noise is to repeat each message symbol a fixed number of times.
Formally, we can define the q-ary *repetition code* of length n (over an alphabet $\mathcal{A} = a_1, \ldots, a_q$ such that $|\mathcal{A}| = q$) as

$$C_q^{rep}(n) = \{\underbrace{a_1 \ldots a_1}_{n}, \underbrace{a_2 \ldots a_2}_{n}, \ldots, \underbrace{a_q \ldots a_q}_{n}\}$$

For example, in the binary case, we can replace every single bit by a threefold repetition, obtaining $C_2^{rep}(3) = \{000, 111\}$.

1. What is the distance of the $C_2^{rep}(3)$ code?
2. Consider the $C_2^{rep}(3)$ code, and suppose that after a transmission over a BSC, the word $\mathbf{v} = 010$ is received. How many errors occurred (if any)? What is the sent codeword? Why?
3. What is the size of the binary $C_2^{rep}(n)$ code?
4. What are the distance and the error correction capability of the $C_q^{rep}(n)$?

Exercise: Consider the binary Hamming code $(7, 16)$, and assume that the transmission takes place over a Binary Erasure Channel (BEC). Decode, if possible, the received word $\mathbf{r} = 01\epsilon10\epsilon0$.

1.3 Sage code and examples

Example 1.11. *The ball $\mathcal{B}_1(111)$ in $\mathbb{Z}_2^3$. (See Example 1.8).*

```
B3 = graphs.CubeGraph(3)
B3.vertices()

vc = {}
vc['white'] = B3.vertices()
vc['red'] = ['011','111','101','110']
B3.plot3d(vertex_size=0.1,edge_size=0.01,vertex_labels=True,
          vertex_colors=vc,frame=false,graph_border=false)
```

Example 1.12. *We can visualize the binary repetion code of length 3.*

```
B3 = graphs.CubeGraph(3) B3.vertices()
vc = {}
vc['white'] = B3.vertices()
vc['red'] = ['000','111']
B3.plot3d(vertex_size=0.06,edge_size=0.01,vertex_colors=vc)
```

Example 1.13 (Hamming Code). `t = var('t')`

```
 circle1 = parametric_plot([10*cos(t)-5,10*sin(t)-5], (t,0,2*pi))
 circle2 = parametric_plot([10*cos(t)+5,10*sin(t)-5], (t,0,2*pi))
 circle3 = parametric_plot([10*cos(t),10*sin(t)+5], (t,0,2*pi))
 text1 = text("$m4$", (0,0))
 text2 = text("$m1$", (-6,2))
 text3 = text("$m3$", (0,-7))
 text4 = text("$m2$", (6,2))
 text5 = text("$p2$", (-9,-9))
 text6 = text("$p3$", (9,-9))
 text7 = text("$p1$", (0,9))
 textA = text("$A$", (-13,13))
 textB = text("$B$", (13,13))
 textC = text("$C$", (0,-17))
 text_all = text1+text2+text3+text4+text5+text6+text7+textA+textB+textC
 show(circle1+circle2+circle3+text_all,axes=false)

#Hamming 'decoding'
t = var('t')
m1=1
m2=1
m3=1
m4=1
p1=1
p2=0
p3=0
a1=m1+m4+m2+p1
a1
a2=m1+m4+m3+p2
a2
```

```
a3=m4+m3+m2+p3
a3

if a1 % 2  == 0:
 circle3 = parametric_plot([10*cos(t),10*sin(t)+5], (t,0,2*pi), color='green')
else:
 circle3 = parametric_plot([10*cos(t),10*sin(t)+5], (t,0,2*pi), color='red')


if a2 % 2 == 0:
 circle1 = parametric_plot([10*cos(t)-5,10*sin(t)-5], (t,0,2*pi), color='green')
else:
 circle1 = parametric_plot([10*cos(t)-5,10*sin(t)-5], (t,0,2*pi), color='red')


if a3 % 2 == 0 :
 circle2 = parametric_plot([10*cos(t)+5,10*sin(t)-5], (t,0,2*pi),color='green')
else:
 circle2 = parametric_plot([10*cos(t)+5,10*sin(t)-5], (t,0,2*pi),color='red')


text1 = text(m4, (0,0))
text2 = text(m1, (-6,2))
text3 = text(m3, (0,-7))
text4 = text(m2, (6,2))
text5 = text(p2  , (-9,-9))
text6 = text(p3, (9,-9))
text7 = text(p1, (0,9))
textA = text("$A$", (-13,13))
textB = text("$B$", (13,13))
textC = text("$C$", (0,-17))
text_all = text1+text2+text3+text4+text5+text6+text7+textA+textB+textC
show(circle1+circle2+circle3+text_all,axes=false)

 M=matrix(GF(2), [[0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1],
                  [0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1],
                  [0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1],
                  [0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1],
])
G = matrix(GF(2), [[1,0,0,0,1,1,0],
                   [0,1,0,0,1,0,1],
                   [0,0,1,0,0,1,1],
                   [0,0,0,1,1,1,1]])
sage: C = LinearCode(GG)
sage: C
Linear code of length 7, dimension 4 over Finite Field of size 2
sage: H = C.check_mat()
sage: H
[1 0 1 0 1 1 0]
```

```
[0 1 1 0 0 1 1]
[0 0 0 1 1 1 1]

CC = codes.HammingCode(3,GF(2))
L = CC.list()
L
```

1.4 Problems

Exercise 1. Determine the parameters (n, k, d) of the binary code

$$C = \{00001100, 00001111, 01010101, 11011101\}$$

Exercise 2. Let C be a binary block code that can detect any single error. If C has length 4, what is the maximum number of codewords in C?

Exercise 3. Prove that a code C can correct any pattern of t errors and s erasures provided that

$$2t + s + 1 \leq d.$$

(Hint: Consider the code obtained by deleting the erasure positions.)

Exercise 4. Suppose that four messages are encoded into 000000, 001111, 110011 and 111100. These four messages are transmitted with equal probability over a BSC with error probability p. If one receives a sequence different from the four sequences above, one knows that errors have been made during the transmission. What is the probability that errors have been made during the transmission and that the receiver will not find out.

2 Linear codes

In this chapter

- Linear codes
- Singleton Bound and MDS codes
- Reed-Solomon codes

Vector spaces over finite fields

In this section we recall some basic definitions and results about linear algebra. We assume the reader familiar with the concepts of basis and dimension of a vector space.

We use $\mathbb{F}_q$ to denote finite fields of q elements. Here we consider subspaces of vector spaces $\mathbb{F}_q^n$.

For vector spaces over finite fields $\mathbb{F}_q$ almost all the results valid for real vector spaces are still true, except that the scalars are now in $\mathbb{F}_q$. For example in the binary case, that is, for any vector space V over $\mathbb{F}_2$, this means that twice a vector is the zero vector: $\mathbf{v} + \mathbf{v} = 2\mathbf{v} = 0$, for every vector $\mathbf{v} \in V$. However, most of the notions we have seen for vector spaces, for example that of subspaces, linear independence, bases, dimension, linear maps, kernel and image, the Rank Nullity Theorem, matrices and determinants, all hold unvaried in this more general setting, as well as all the methods based on Gaussian reduction.

Definition 2.0.1. A subset $S \subseteq \mathbb{F}_q^n$ is a linear subspace of $\mathbb{F}_q^n$ if for every $\mathbf{v} = (v_0, \ldots, v_{n-1}), \mathbf{w} = (w_0, \ldots, w_{n-1}) \in S$, and every $a \in \mathbb{F}_q$ it holds

- $\mathbf{v} + \mathbf{w} = (v_0 + w_0, \ldots, v_{n-1} + w_{n-1}) \in S$
- $a \cdot \mathbf{v} = (av_0, \ldots, av_{n-1}) \in S$

Definition 2.0.2. The **span** of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ is the set

$$span(\mathbf{v}_1, \ldots, \mathbf{v}_k) = \left\{ \sum_{i=1}^{k} a_i \cdot \mathbf{v}_i \mid a_1, \ldots, a_k \in \mathbb{F}_q \right\},$$

i.e. it is the set of all linear combination of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$.

Exercise: Let $S = \{s_1, \ldots, s_k\}$ be a subset of $\mathbb{F}_q^n$, then span(S) denotes the set of all linear combinations of the vectors in S. Prove that, for any subset S of $\mathbb{F}_q^n$, span(S) is a subspace of $\mathbb{F}_q^n$.

Exercise: Let $S = \{(1,0,1,0), (0,1,0,1), (1,1,1,1)\} \subset \mathbb{F}_2^4$. List of the elements of span(S).

Definition 2.0.3. Let $\mathbf{v} = (v_0, \ldots, v_{n-1})$ and $\mathbf{w} = (w_0, \ldots, w_{n-1})$ be vectors in $\mathbb{F}_q^n$, we define the **scalar product** of $\mathbf{v}$ and $\mathbf{w}$ as

$$\mathbf{v} \cdot \mathbf{w} = v_0 w_0 + \cdots + v_{n-1} w_{n-1} \in \mathbb{F}_q.$$

Remark With finite fields the orthogonality relations between words is not intuitive. For example there are nonzero vectors that are orthogonal to themselves. Consider $0101 \in \mathbb{F}_2^4$, then

$$0101 \cdot 0101 = 0 + 1 + 0 + 1 = 0.$$

Definition 2.0.4. Let $S \subseteq \mathbb{F}_q^n$ be a linear subspace, we denote by $S^\perp$ the **null space** of S, i.e. set of all vectors orthogonal to S:

$$S^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v} \cdot \mathbf{w} = 0, \forall \mathbf{w} \in S\}.$$

Fortunately, the general laws for orthogonality among vector spaces are still valid. We have that

- Let S be a vector subspace of $\mathbb{F}_q^n$, then $(S^\perp)^\perp = S$
- Let S be a vector subspace of $\mathbb{F}_q^n$, then $\dim S + \dim S^\perp = n$.

Proposition 2.0.5. The null space of a linear subspace $S \subseteq \mathbb{F}_q^n$ is also a linear subspace of $\mathbb{F}_q^n$ and its dimension is $n - \dim(S)$.

Definition 2.0.6. Let V and W be two vector spaces over the same field $\mathbb{F}_q$. A map $L : V \to W$ is **linear** if $\forall \mathbf{v}, \mathbf{u} \in V$ and $a_1, a_2 \in \mathbb{F}_q$,

$$L(a_1\mathbf{v} + a_2\mathbf{u}) = a_1 L(\mathbf{v}) + a_2 L(\mathbf{w}).$$

Theorem 2.0.7 (Rank-nullity theorem). Let V and W be vector spaces over some finite field and let $L : V \to W$ be a linear map. Then the rank of L is the dimension of the image of L and the nullity of L is the dimension of the kernel of L, so we have

$$\dim(\mathrm{Im}(L)) + \dim(\ker(L)) = \dim(V).$$

or, equivalently,

$$\mathrm{rk}(L) + \mathrm{nul}(L) = \dim(V).$$

2.1 Linear codes

Linear codes are widely studied because of their algebraic structure, which makes them easier to describe than nonlinear codes.

Example 2.1. *Let C be the $(7, 16)_2$ Hamming code (see Lecture I). It is easy to see that C is a subspace of $\mathbb{F}_2^7$, so it is generated by a matrix whose rows form a basis for C, for example*

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

In this way the encoding map $\mathsf{Enc} : \{0,1\}^4 \to \{0,1\}^7$ *can be easily represented as*

$$\mathbf{m} \mapsto \mathbf{m}G.$$

So if $\mathbf{m} = 1010$*, then its associated codeword is*

$$(1\,0\,1\,0) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} = (1\,0\,1\,0\,1\,0\,1).$$

Note that the message and the codewords are represented as vectors.

Definition 2.1.1. Let $\mathbb{F}_q^n$ denote the vector space of all n-tuples over the finite field $\mathbb{F}_q$. Let $k \in \mathbb{N}$ be such that $1 \leq k \leq n$. An $[n,k]_q$ **linear code** C is a k- dimensional vector subspace of $(\mathbb{F}_q)^n$. In this case, we say that C is a linear code over $\mathbb{F}_q$ with length n and dimension k.

From now on we shorten "linear code on $\mathbb{F}_q$ with length n and dimension k" to "$[n,k]_q$ code".

Example 2.2. *The* $(7,16)_2$ *Hamming code is an* $[7,4]_2$ *binary linear code.*

Notice that the dimension k and the size M of an $[n,k]_q$ linear code are related, i.e. $M = q^k$ (why?).

Example 2.3. *We can consider the following subsets of* $\mathbb{F}_2^4$:

$$C_1 = \{1000, 0010\}, \quad C_2 = \{0010, 0000\}.$$

The set C_1 *is not a vector space, because the sum of its elements does not belong to* C_1 *(i.e.* C_1 *is not closed under the sum). The set* C_2 *is obviously a vector subspace of* $\mathbb{F}_2^4$ *generated by* 0010*. This fact can be represented in our setting saying that* C_2 *is a linear code over* $\mathbb{F}_2$ *of length 4 and dimension 1.*

Exercise: Is the $C_q^{rep}(n)$ (see Lecture 1) linear?

Notice that if C is a linear code, the minimum distance $d(C)$ of C is the same as the minimum weight of nonzero words:

$$d(C) = \min\{\mathrm{wt}(\mathbf{c}) \mid \mathbf{c} \in C, \mathbf{c} \neq 0\}.$$

The problem of finding (or at least to bound) the distance of a code is certainly one of the most difficult and important ones: actually some widely used codes still have their distance unknown. Generally speaking, the problem of distance determination for a linear code is hard and no sub-exponential algorithm is known (See part about Complexity of this course).

Exercise: Prove that for a linear code C:

a) $d(\mathbf{c}, \mathbf{c}') = \mathrm{wt}(\mathbf{c} - \mathbf{c}')$;

b) $d(C) = \mathrm{wt}(C)$, where $\mathrm{wt}(C)$ denotes the minimum weight of all nonzero codewords in C.

If we know the distance $d = d(C)$ of an $[n,k]_q$ code, then we can refer to this code as an $[n,k,d]_q$ code.

Example 2.4. *Consider the finite field $\mathbb{F}_4 = \{0, 1, \alpha, \alpha+1\}$, where $\alpha^2 = \alpha + 1$. We recall that we can represent elements of $\mathbb{F}_4$ as follows:*

Table 2.1: $\mathbb{F}_4$

0	0	00
1	1	10
α	α	01
α^2	$\alpha+1$	11

Let $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ be the following elements of $\mathbb{F}_4^3$:

$$\mathbf{x}_1 = (0, \alpha, \alpha+1) \quad \mathbf{x}_2 = (0, 1, \alpha+1) \quad \mathbf{x}_3 = (0, 0, 0).$$

Then $d(\mathbf{x}_1, \mathbf{x}_2) = 1$, $d(\mathbf{x}_1, \mathbf{x}_3) = 2$ and $d(\mathbf{x}_2, \mathbf{x}_3) = 2$.

2.1.1 Generator and parity-check matrix

Thanks to their algebraic structure, linear codes can be represented in a very efficient way.

If C is a linear code of length n and dimension k, we can find k basis elements for C, each of which will be a vector of length n.

Definition 2.1.2 (Generator matrix and encoding). Let $C \subseteq \mathbb{F}_q^n$ be a linear code of dimension k. A matrix $G \in \mathbb{F}_q^{k \times n}$ (i.e. a matrix of dimension $k \times n$ over $\mathbb{F}_q$) is said to be a **generator matrix** for C if its k rows span C.

The generator matrix provides a way to encode a message $\mathbf{m} \in \mathbb{F}_q^k$ as the codeword $\mathbf{c} = \mathbf{m}G \in \mathbb{F}_q^n$. Thus a linear code has an encoding map $\mathsf{Enc} : F_q^k \to \mathbb{F}_q^n$ which is an injective linear map defined as $\mathbf{m} \mapsto \mathbf{m}G$.

Notice that a linear code admits many different generator matrices, corresponding to the different choices of basis for the code as a vector space.

Example 2.5. *Consider the $[5,2,4]_4$ code with generator matrix*

$$G = \begin{pmatrix} 1 & 1 & \alpha^2 & 0 & \alpha^2 \\ 0 & 1 & 0 & 1 & \alpha \end{pmatrix}$$

The codeword corresponding to the message $\mathbf{m} = (1 \quad \alpha)$ is obtained by

$$(1 \ \alpha) \cdot \begin{pmatrix} 1 & 1 & \alpha^2 & 0 & \alpha^2 \\ 0 & 1 & 0 & 1 & \alpha \end{pmatrix} = (1 \quad 1+\alpha \quad \alpha^2 \quad \alpha \quad 0).$$

Exercise: Find a generator matrix of the repetition code $C_q^{rep}(n)$.

Fact 2.1.3. Let C be an $[n,k]_q$ linear code. After permuting coordinates if necessary, C has a generator matrix of the form $[I_k \mid \hat{G}]$ where I_k is the $k \times k$ identity matrix and $\hat{G}$ is some $k \times (n-k)$ matrix. A generator matrix in the form $[I_k \mid \hat{G}]$ is said to be in **standard form**.

In order to obtain a generator matrix in standard form we can perform the following operations:

1. Permutation of rows.
2. Multiplication of a row by a non-zero scalar.
3. Adding a scalar multiple of one row to another row.
4. Permutation of columns.
5. Multiplication of any fixed column by a non-zero scalar.

Example 2.6. *Let us consider the code $[5,2,4]_4$ of the previous example, then permuting the second and the fourth columns we obtain the matrix:*

$$G' = \begin{pmatrix} 1 & 0 & \alpha^2 & 1 & \alpha^2 \\ 0 & 1 & 0 & 1 & \alpha \end{pmatrix},$$

that is in standard form.

Exercise: Consider $[6,3]$ over $\mathbb{F}_3$ code with generator matrix

$$G = \begin{pmatrix} 2 & 0 & 2 & 1 & 1 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Reduce G in standard form.

When a generator matrix in standard form is used for encoding, the encoding is called *systematic*: this means that the first k symbols of the codeword are just the message symbols, and the remaining $n-k$ symbols correspond to redundant check symbols. Thus, after permuting coordinates, if needed, every linear code admits a systematic encoding.
Notice that using a generator matrix in standard form is also useful to decode, if there are no errors, then the decoding procedures reduces to extract the first k symbols.

Example 2.7. *Consider the code $[5,2,4]_4$ of Example 2.5 and the message $\mathbf{m} = (1 \quad \alpha)$. By encoding $\mathbf{m}$ using the matrix in standard form G' of Example 2.6, we obtain the codeword $\mathbf{c} = (1 \quad \alpha \quad \alpha^2 \quad 1+\alpha \quad 0)$ which contains in the first 2 positions the $k = 2$ information symbols (i.e. the message).*

As a linear code is a subspace of a vector space, it is the kernel of some linear map. In particular, there is an $(n-k) \times n$ matrix H over $\mathbb{F}_q$, called a **parity-check matrix** for the $[n,k]$ code C, defined by

$$C = \{\mathbf{c} \in \mathbb{F}_q^n \mid H\mathbf{c}^T = 0\}.$$

In other words, C is the null space of H.

Example 2.8. *Consider the* $[7,4]_2$ *Hamming code. The matrix*

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

is a parity-check matrix of C*. In fact for each codeword* $\mathbf{c} \in C$ *it holds* $H\mathbf{c}^T = 0$*.*

Remark Recall that, given a matrix A, we denote by A^T the matrix which is obtained by A but with rows and columns swapped.

Example 2.9.

$$A = \begin{pmatrix} 3 & 4 \end{pmatrix} \qquad A^T = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

$$B = \begin{pmatrix} 3 & 4 & 1 \\ 6 & 7 & 8 \end{pmatrix} \qquad B^T = \begin{pmatrix} 3 & 6 \\ 4 & 7 \\ 1 & 8 \end{pmatrix}$$

Exercise: Prove that the rows of the $(n-k) \times n$ parity-check matrix H are independent. (Hint: $\forall \mathbf{x} \in \mathbb{F}_q^n$, the map $\mathbf{x} \mapsto H\mathbf{x}^T$ is a linear transformation from $\mathbb{F}_q^n \mapsto \mathbb{F}_q^{n-k}$. What is the kernel of this transformation? What is the rank of H?)

A generator matrix G of an $[n,k]_q$ linear code C is simply a matrix whose rows are independent and span the code. In the same way, since the $n-k$ rows of a parity-check matrix H are independent, H is a generator matrix too.
In particular, denoting by "$\cdot$" the usual scalar product, we can consider the dual space $S^\perp$ of a subspace S of $\mathbb{F}_q^n$.

Definition 2.1.4. The *dual code* of C is the $[n, n-k]_q$ linear code $C^\perp$ composed by all the vectors orthogonal to all words of C:

$$C^\perp = \{\tilde{\mathbf{c}} \mid \tilde{\mathbf{c}} \cdot \mathbf{c} = 0, \forall \mathbf{c} \in C\}.$$

Fact 2.1.5. An $(n-k) \times n$ generator matrix H of $C^\perp$ is a parity-check matrix for C.

Exercise: Prove that if $[I_k \mid \hat{G}]$ is a generator matrix for the $[n,k]_q$ linear code C in standard form, then $[-\hat{G}^T \mid I_{n-k}]$ is a parity-check matrix for C. (Hint: consider the product HG^T).

The following theorem shows the relation between the weight of a codeword and a parity-check matrix for a linear code.

Theorem 2.1.6. Let C be a linear code with parity check matrix H. If $\mathbf{c} \in C$, the columns of H corresponding to the nonzero coordinates of $\mathbf{c}$ are linearly dependent. Conversely, if a linear dependence relation with nonzero coefficients exists among w columns of H, then there is a codeword in C of weight w whose nonzero coordinates correspond to these columns.

Proof Consider a codeword $\mathbf{c}$ such that $\mathrm{wt}(\mathbf{c}) = w$. We know that

$$0 = H\mathbf{c}^T = \begin{bmatrix} \mathbf{h}_0 & \mathbf{h}_2 & \dots & \mathbf{h}_{n-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix},$$

where $\mathbf{h}_i$ are the columns of H. Let us suppose $\mathrm{supp}(\mathbf{c}) = \{c_{i_1}, \dots, c_{i_w}\}$, then this means

$$\mathbf{h}_{i_1} c_{i_1} + \cdots + \mathbf{h}_{i_w} c_{i_w} = 0,$$

i.e. $\mathbf{h}_{i_1}, \dots, \mathbf{h}_{i_1}$ are linearly dependent. Conversely, suppose that $\mathbf{h}_{i_1}, \dots, \mathbf{h}_{i_w}$ are linearly dependent, this means that there exist $a_1, \dots, a_w$, $a_i \in \mathbb{F}_q$, such that $\mathbf{h}_{i_1} a_1 + \cdots + \mathbf{h}_{i_w} a_w = 0$.. If we consider the vector $\mathbf{a} \in \mathbb{F}_q^n$, such that $\mathrm{supp}(\mathbf{a}) = \{a_1, \dots, a_w\}$, then it holds $H\mathbf{a}^T = 0$, hence $\mathbf{a} \in C$. ■

Example 2.10. *Let C be the $[7,3]_3$ linear code with generator matrix*

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

We can easily find a parity-check matrix for C, for example:

$$H = \begin{pmatrix} 1 & 0 & 0 & 2 & 1 & 0 & 2 \\ 0 & 1 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 1 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

The fourth and the fifth column are linearly dependent, and indeed there are two codewords in C, namely $\mathbf{c}_1 = 0001100$ and $\mathbf{c}_2 = 0002200$, of weight 2 whose nonzero components correspond exactly to these columns. Since there is no zero column in H, then we can conclude that C has not codeword of weight 1.

Suppose that any two columns of H are linearly independent (and hence none of them is zero). Then there cannot exist a codeword $\mathbf{c} \in C$ such that $\mathrm{wt}(\mathbf{c}) = 2$ (or $\mathrm{wt}(\mathbf{c}) = 1$), because otherwise there should exist a linear dependence relation between two columns (or one columns of zeros) of H, due to the previous theorem. So in this case we can say that $d(C) \geq 3$. In general the parity-check matrix can be used to find the minimum distance.

Proposition 2.1.7. A linear code has minimum weight d if and only every subset of $d-1$ columns in H are linearly independent and there exists a subset of d columns in H that are dependent.

Example 2.11. *Consider, as usual, the Hamming code* $[7,4]_2$*. Let* H *be its parity-check matrix (see Example 2.8). Then it is clear that we cannot find 2 or less linearly dependent columns of* H*. Then* $d \geq 3$*. Moreover, to prove that* $d = 3$*: it is enough to find a codeword* $\mathbf{c}$ *such that* $\mathrm{wt}(\mathbf{c}) = 3$*.*

2.2 Singleton bound and MDS codes

In this section we show a very simple bound on the distance of an $(n,k)_q$ code.

Theorem 2.2.1 (Singleton Bound). If C is an $(n,k,d)_q$ code, then $d \leq n - k + 1$

Proof [For linear codes] Since any $n - k + 1$ columns of a parity-check matrix are linearly dependent (why?), then $d \leq n - k + 1$. ■

Codes that meet this bound, i.e. satisfy $d = n - k + 1$ are called **Maximum Distance Separable** (MDS) codes.

The following proposition shows some interesting properties of MDS codes.

Proposition 2.2.2. Let C be an $[n,k,d]_q$ linear code. Let G and H be a generator and a parity-check matrix for C, respectively. Then the following are equivalent:

1) C is MDS;

2) Every subset of $n - k$ columns in H is linearly independent;

3) Every subset of k columns in G is linearly independent;

4) $C^\perp$ is an MDS code.

Proof 1) $\Leftrightarrow$ 2) : We know that every subset of $n-k$ columns of H is linearly independent if and only if $d(C) \geq n-k+1$. By the Singleton bound, it must be equality and so items (1) and (2) are equivalent.
3) $\Rightarrow$ 4) are equivalent for the same reason, since G is a parity check matrix for $C^\perp$.
1) $\Rightarrow$ 4): since H is a parity-check matrix of C, it is a generator of $C^\perp$ which is a $[n, n-k]_q$-code. We will show that $d(C^\perp) = k + 1$. If $d(C^\perp) \leq k$ the there exists a word $\mathbf{c} \in C^\perp$ such that $\mathrm{wt}(\mathbf{c}) \leq k$, and so at least $n-k$ of its coordinates are 0. Assume without loss of generality that these are the last coordinates. Write $H = (A|H')$, where $A \in \mathbb{F}_q^{n-k} \times k$, $H' \in \mathbb{F}_q^{(n-k)\times(n-k)}$. Since all the columns of H' are linearly independent (by the proof of equivalence between (1) and (2), H' is invertible. This implies that its rows are also linearly independent (by the facts that a square matrix M is invertible if and only if M^T is invertible, and M is invertible if and only if its columns are linearly independent). This implies that the codeword $\mathbf{c}$ with $\mathrm{wt}(\mathbf{c}) \leq k$ must be the word $\mathbf{c} = 0$. We conclude that all non-zero words have weight greater than k and so $d(C^\perp) = \mathrm{wt}(C^\perp) \geq k + 1$.
4) $\Rightarrow$ 1): this follows immediately from the fact that $(C^\perp)^\perp = C$ and the fact that 1) $\Leftarrow$ 4). ■

2.3 Reed-Solomon codes

Here we describe a well-known family of linear codes.

Definition 2.3.1 (Reed-Solomon codes). Take n distinct points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$, with n such that $q \geq n$, and let k be an integer such that $1 \leq k \leq n$. We define the Reed-Solomon code as

$$RS_q(n,k) = \{(f(\alpha_1), \ldots, f(\alpha_n)) \in \mathbb{F}_q^n \mid f \in \mathbb{F}_q[x] \text{ is a polynomial of degree } \leq k-1\} \cup \{0\}$$

Remark Usually the set of points $S = \{\alpha_1, \ldots, \alpha_n\}$ is $\mathbb{F}_q^*$.

Exercise: Let $\mathcal{P}_{k-1}$ be the vector space of all polynomials of degree $k-1$ over $\mathbb{F}_q$. Prove that $\mathcal{P}_{k-1}$ has dimension k and that $\{1, x, \ldots, x^{k-1}\}$ is a basis for it.

Using the previous exercise we can see an $RS_q(n,k)$ code C as the image of

$$\mathsf{Enc} : \mathcal{P}_{k-1} \longrightarrow \mathbb{F}_q^n$$

defined by

$$f \longmapsto (f(\alpha_1), \ldots, f(\alpha_n)).$$

Let $\{l_1, \ldots, l_k\}$ be a basis of $\mathcal{P}_{k-1}$ in $\mathbb{F}_q[x]$, a generator matrix for C is given by:

$$G = \begin{pmatrix} l_1(\alpha_1) & l_1(\alpha_2) & \ldots & l_1(\alpha_n) \\ l_2(\alpha_1) & l_2(\alpha_2) & \ldots & l_2(\alpha_n) \\ \vdots & \vdots & \ddots & \vdots \\ l_k(\alpha_1) & l_k(\alpha_2) & \ldots & l_k(\alpha_n) \end{pmatrix}.$$

In particular, since $\{1, x, \ldots, x^{k-1}\}$ is a basis of $\mathcal{P}_{k-1}$, the Vandermonde Matrix

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \ldots & \alpha_n^{k-1} \end{pmatrix}$$

is a generator matrix of C.

Let $\mathbf{m} = (m_0, \ldots, m_{k-1}) \in \mathbb{F}_q^k$ be a message, we can write $\mathbf{m}$ as a polynomial $m(x) \in \mathbb{F}_q[x]$ of degree $\leq k-1$, i.e.

$$m(x) = m_0 + m_1 x + \cdots + m_{k-1} x^{k-1}.$$

In this way the encoding map corresponds to evaluating $m(x)$ at $\alpha_1, \ldots, \alpha_n$.

Exercise: Prove that the Reed Solomon $\mathrm{RS}_q(n,k)$ code is an $[n,k]_q$ linear codes.

Proposition 2.3.2. The $RS_q(n,k)$ is MDS, i.e. it is an $[n, k, d = n-k+1]_q$ code.

Proof Since $RS_q(n,k)$ is an $[n,k]$ linear code, to prove the proposition it suffices to show that any non-zero codeword has Hamming weight at least $n-k+1$. Let $(m_0, m_1, \ldots, m_{k-1})$ be a nonzero vector in $\mathbb{F}_q^k$. The polynomial $m(x) = m_0 + m_1 x + \ + m_{k-1}x^{k-1}$ is a non-zero polynomial of degree at most $k-1$. Then $m(x)$ has at most $k-1$ roots (why?) and hence $(m(\alpha_1), \ldots, m(\alpha_n))$ has at most $k-1$ zeros. By the Singleton bound, we know that the distance d is at most $n-k+1$, Therefore it has to be $d = n-k+1$. ■

Example 2.12. *Consider the RS codes over $\mathbb{F}_9$ with $k = 3$. Let $\{1, x, x^2\}$ a basis for $\mathcal{P}_2$. Then let S be the set of points $\mathbb{F}_9^* = \{1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7\}$, we obtain the generator matrix*

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^1 & \alpha^2 & \alpha^4 & \alpha^6 \end{pmatrix}.$$

Notice that, for all $k < q$, the first $k = 3$ columns of the generator matrix, corresponding to the (monomial) basis of $\mathcal{P}_{k-1}$, give a Vandermonde matrix with nonzero determinant. It follows that the evaluation mapping is one-to-one, and the corresponding Reed- Solomon code is a linear code with block length $n = q-1 = 8$, and dimension $k = \dim \mathcal{P}_{k-1} = 3$. Hence G is a generator matrix for the $[8, 3, 6]$ Reed Solomon code over $\mathbb{F}_9$.

2.4 Hamming codes

We have seen that the distance of a linear code C is the minimum integer d such that d columns of H are linearly dependent. This means that $d(C) = 1$ if and only if H has a zero column; also, $d(C) = 2$ if and only if H has two columns $\mathbf{h}_{i_1}$ and $\mathbf{h}_{i_2}$ that are linearly dependent. In the binary case this implies $\mathbf{h}_{i_1} = \mathbf{h}_{i_2}$. So $d(C) \geq 3$ if and only if H has no zero columns and all columns are mutually independent. This is the case of the Hamming code.

Definition 2.4.1. Let $n = \frac{q^r-1}{q-1}$ and let $H_r(q) \in \mathbb{F}_q^{r \times n}$ be a matrix with nonzero columns and such that each pair of columns are linearly independent. Then $H_r(q)$ is a parity-check matrix of the **q-ary Hamming code** (with parameter r) ($\mathcal{H}_r(q)$).

Theorem 2.4.2. Let $r \geq 2$. Then the q-ary Hamming code $\mathcal{H}_r(q)$ has parameters $n = \frac{q^r-1}{q-1}, k = \frac{q^r-1}{q-1} - r, d = 3$.

Proof The rank of $H_r(q)$ is r, since $\mathbf{e}_1, \ldots, \mathbf{e}_r \in \mathrm{Span}(\mathbf{h}_1, \ldots, \mathbf{h}_n)$. Hence $H_q(r)$ is a parity-check matrix of a code with redundancy r. Any 2 columns of $H_r(q)$ are independent by construction and a column of $\mathrm{wt}(2)$ exists as a linear combination of 2 columns of weight 1. From this we obtain that $d(C) = 3$. ■

Example 2.13. *Let C be the $[7,4,3]_2$ code with parity-check matrix:*

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Then C is an $[7,4,3]$ Hamming code. Note that the columns of H are exactly the non-zero vectors of $\mathbb{F}_2^3$.

NOTE: The following sections contain more advanced material, and can be skipped

2.5 ◇ Equivalent codes

To simplify our study of codes, we introduce a notion of equivalence of codes. If C and D are codes of the same length over the same alphabet, we say that C is equivalent to D if we can get from C to D by a combination of the following operations.

Permutation of the positions in the codewords: Choose a permutation σ of $\{1,\ldots,n\}$ and for a codeword $c = c_1 \ldots_n$ in C, define

$$c_\sigma = c_{\sigma(1)} \cdots c_{\sigma(n)},$$

and define

$$C_\sigma = \{c_\sigma | c \in C\}.$$

Applying a permutation of $\mathcal{A}$ in a fixed position in the codewords: Choose a permutation σ of $\mathcal{A}$ and an integer $i \in \{1,\ldots,n\}$, and for $c = c_1 \ldots_n$ in C, define

$$c_{\sigma,i} = c_1 \ldots c_{i-1}\sigma(c_i)c_{i+1} \ldots c_n,$$

and then define

$$C_{\sigma,i} = \{c_{\sigma,i} | c \in C\}.$$

Example 2.14. *Consider the following ternary codes of length 2:*

$$C_1 = \{10, 21, 02\} \qquad C_2 = \{01, 12, 20\} \qquad C_3 = \{00, 11, 22\}.$$

We can obtain C_1 from C_2 (and viceversa) by Operation 1 , and we can obtain C_3 from C_2 by Operation 2, permuting the symbol appearing in the secon position via $\sigma = (021)$. Hence C_1, C_2, C_3 are equivalent codes.

The point of equivalence is that equivalent codes have the same size and the same minimum distance; we can often simplify both decoding procedures and some of our proofs by replacing codes with equivalent codes.

Lemma 2.5.1. Suppose C is a non-empty code over $\mathcal{A}$, and $a \in \mathcal{A}$. Then C is equivalent to a code containing the codeword $aa \ldots a$.

2.6 ◇ The main problem of coding theory

We have seen that the distance d is an important parameter for a code. A fundamental problem in coding theory is, given the length and the number of codewords (dimension if the code is linear), that of determining a code with largest distance, or equivalently, to find the largest code of a given length and distance. The following definition is useful to state some bounds on codes more clearly.

Definition 2.6.1. Let n, d be positive integers with $d \leq n$. Then the number $A_q(n, d)$ denotes the maximum number of codewords in a code of length n over an alphabet $\mathcal{A}$ of size q, and with distance d, i.e.

$$A_q(n, d) = \max\{M \mid \text{ there exists an } (n, M, d)\text{-code over } \mathcal{A}\}.$$

An $(n, M, d)_q$ code such that $M = A_q(n, d)$ is called **optimal**.
This maximum, when restricted to linear codes over $\mathbb{F}_q$, is denoted by $B_q(n, d)$, i.e.

$$B_q(n, d) = \max\{q^k \mid \text{ there exists a linear code } [n, k, d]\text{-code over } \mathbb{F}_q^n\}.$$

A linear code for which $q^k = B_q(n, d)$ is called **optimal code**.

Unfortunately, very little is known about $A_q(n, d)$, in what follows we show some well known facts about these numbers.

Clearly it can be $B_q(n, d) < A_q(n, d)$. Then, given n and d, if we look at the largest possible code, we have sometimes to use nonlinear codes.

Theorem 2.6.2. Let $q \geq 2$ be a prime power. Then, $\forall n$, we have

1. $\forall 1 \leq d \leq n$ it holds that $B_q(n, d) \leq A_q(n, d) \leq q^n$
2. $B_q(n, 1) = A_q(n, 1) = q^n$
3. $B_q(n, n) = A_q(n, n) = q$

Proof Here it is worth noting that in order to prove that $A_q(n, d) = A$, for some value A, it is enough to prove that:

- $A_q(n, d) \geq A$
- Exhibit a specific $(n, A)_q$ code C such that $d(C) \geq d$.

These facts clearly imply $A_q(n, d) \geq A_q(n, d(C)) \geq A$.

Now we prove the theorem:

1. This follows from the definition and from the fact that $C \subseteq \mathcal{A}^n$.
2. $C = \mathbb{F}_q^n$ is an $[n, n, 1]$ code and $\mathcal{A}^n$ is an $(n, q^n, 1)$ code, and hence $A_q(n, 1) \geq B_q(n, 1) \geq q^n$. So from 1. follows the thesis.

3. All codewords have length n and the distance is n. Hence each pair of codewords must differ in every coordinate. The alphabet has size q and then $B_q(n,n) \leq A_q(n,n) \leq q$. The repetition code is a linear code that achieves this exact bound, so $B_q(n,n) = A_q(n,n) = q$.

■

We recall some classical bounds that restrict the existence of codes with given parameters. For any $\mathbf{x} \in (\mathbb{F}_q)^n$ and any positive number r, let $\mathcal{B}_r(\mathbf{x})$ be the sphere of radius r centered in x, with respect to the Hamming distance. Note that the size of $\mathcal{B}_r(\mathbf{x})$ is independent of $\mathbf{x}$ and depends only on r, q and n. Let $V_q(n,r)$ denote the number of elements in $\mathcal{B}_r(\mathbf{x})$ for any $\mathbf{x} \in (\mathbb{F}_q)^n$. For any $\mathbf{y} \in \mathcal{B}_r(\mathbf{x})$, there are $(q-1)$ possible values for each of the r positions in which $\mathbf{x}$ and $\mathbf{y}$ differ. So we see that

$$V_q(n,r) = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i$$

From the fact that the spheres of radius $t = \lfloor \frac{d-1)}{2} \rfloor$ about codewords are pairwise disjoint, the *sphere packing (upper) bound (or Hamming bound)* immediately follows.

Theorem 2.6.3. For every $q > 1$ and $n, d \in \mathbb{N}$, such that $1 \leq d \leq n$ it holds

$$A_q(n,d) \leq \frac{q^n}{V_q(n,t)}.$$

Proof Let $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_M\}$ be the optimal code over $\mathcal{A}$ of size q and $t = \lfloor (d-1)/2 \rfloor$. Since d is the distance of the code, the balls $\mathcal{B}_t(\mathbf{c}_i)$ are all disjoint:

$$\sqcup_{i=1}^{M} \mathcal{B}_t(\mathbf{c}_i) \subseteq \mathcal{A}^n,$$

where $\sqcup$ indicates the disjoint union. Then we have

$$M \cdot V_q(n,t) \leq q^n.$$

Using the fact that $M = A_q(n,d)$, we obtain the thesis. ■

There exist codes that achieve the Hamming bound.

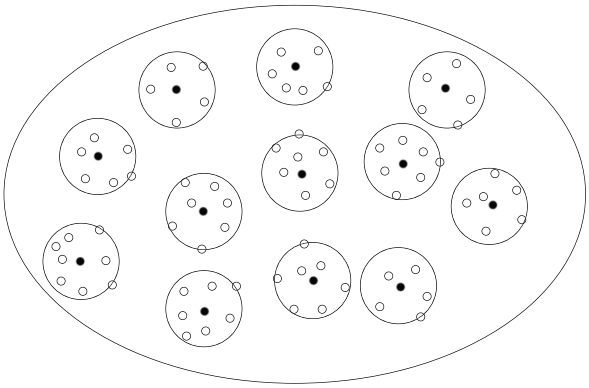
Definition 2.6.4. A code C over an alphabet of size q with parameters (n, M, d) is called a *perfect code* if

$$M = q^n / V_q(n,t).$$

Remark MDS and perfect codes are incomparable, in that exist perfect codes that are not MDS and MDS that are not perfect.

Proposition 2.6.5. The Hamming codes are perfect codes.

Figure 2.1: A perfect code



Proof Let C be a $\mathcal{H}_r(q)$, then $M = q^{\frac{q^r-1}{q-1}-r}$ and $t = 1$. We have that

$$V_q^n(1) = 1 + \frac{q^r - 1}{q - 1}(q - 1),$$

and

$$\frac{q^n}{V_q^n(1)} = \frac{q^{(\frac{q^r-1}{q-1} - r)}}{q^r} = M.$$

This concludes the proof.
■

Note that every perfect code is an optimal code, but not necessarily the other way around.

We state (without a proof) a lower bound for $A_q(n, d)$. The high level idea here is the following. Let C be a code and draw a sphere of radius $d - 1$ around every codeword. In an optimal code, it must be the case that the spheres include all of $\mathcal{A}^n$; otherwise, there exists a word that can be added to the code that would be at distance d from all existing codewords. This yields a larger code, in contradiction to the assumed optimality. Thus, the number of codewords in an optimal code is at least the size of the number of spheres that it takes to cover the entire space A^n.

Theorem 2.6.6 (Gilbert-Varshamov bound). For every $q \geq 1$, and for every $n, d \in \mathbb{N}$, such that $1 \leq d \leq n$ it holds

$$A_q(n, d) \geq \frac{q^n}{V_q(n, d - 1)}.$$

2.7 Sage, examples, exercises

Example 2.15. *Let* $S = \{11101, 10110, 01011, 11010\}$. *Find a basis for the linear code* $C = Span(S)$.

```
sage: MS = MatrixSpace(GF(2),4,5);
sage: A = MS([1,1,1,0,1,1,1,0,1,1,0,0,1,0,1,1,1,1,0,1,0]);
sage: A
[1 1 1 0 1]
[1 0 1 1 0]
[0 1 0 1 1]
[1 1 0 1 0]
sage: G=A.echelon_form()
sage: G
[1 0 0 0 1]
[0 1 0 1 1]
[0 0 1 1 1]
[0 0 0 0 0]
sage: C = LinearCode(G)
sage: C
Linear code of length 5, dimension 3 over Finite Field of size 2
```

Then $\{10001, 01011, 00111\}$ *is a basis for* C. *If we want the parity-check matrix of* C*:*

```
sage: MG = MatrixSpace(GF(2),3,5);
sage: G = MG([[1,0,0,0,1],[0,1,0,1,1],[0,0,1,1,1]])
G.right_kernel()
Vector space of degree 5 and dimension 2 over Finite Field of size 2
Basis matrix:
[1 0 0 1 1]
[0 1 1 1 0]
/*equivalently*/
H=C.check_mat()
[1 0 0 1 1]
[0 1 1 1 0]
```

Note that in the binary case, we can obtain a parity-check matrix H *from* G*, taking the* $k \times (n-k)$ $(3 \times 2$*, in this case*$)$ *matrix obtained from* G *by deleting the leading columns of* G*, i.e.*

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

In the remaining $n - k$ *rows we place the* $(n - k) \times (n - k)$ *identity matrix.*

```
sage: MH = MatrixSpace(GF(2),2,5);
sage: HH = MH([[0,1,1,1,0],[1,1,1,0,1]])
sage: MH = MatrixSpace(GF(2),2,5);
sage: HH = Matrix([[0,1,1,1,0],[1,1,1,0,1]])
sage: GG * transpose(H)
[0 0]
```

```
[0 0]
[0 0]
sage: transpose(HH)
[0 1]
[1 1]
[1 1]
[1 0]
[0 1]
```

Why this works? The idea is that the columns of H obtained in this way are linearly independent, and $C^{\perp} = n - dim(C) = n - k$. It is easy to prove that $G \cdot H = X + X = 0$ (since $G = [I_k|X]$ and $H = [X|I_{n-k}]$.

Example 2.16. *Given a random linear code $[30, 9]_3$ find the codeword corresponding to $m =$ (002120220).*

```
sage:  C = RandomLinearCode(30,9,GF(3));
sage: G=C.gen_mat()
sage: G
[1 0 0 0 0 0 0 0 0 0 1 2 1 0 1 2 0 0 0 0 1 2 2 1 1 1 0 2 2 0]
[0 1 0 0 0 0 0 0 2 0 0 1 2 2 1 1 2 2 2 1 2 1 0 0 1 2 0 2 1 2]
[0 0 1 0 0 0 0 0 2 0 2 1 2 2 2 2 2 1 1 0 1 2 0 0 1 2 1 0 1 1]
[0 0 0 1 0 0 0 0 2 0 2 2 1 1 0 2 2 0 2 2 2 0 2 2 0 1 0 2 1 1]
[0 0 0 0 1 0 0 0 2 0 0 2 2 2 1 2 2 2 0 1 0 0 0 0 1 1 2 2 2 1]
[0 0 0 0 0 1 0 0 2 0 2 2 1 1 1 0 2 2 2 2 1 0 2 1 2 1 0 2 0 1]
[0 0 0 0 0 0 1 0 1 0 1 2 2 1 2 1 0 2 0 2 0 1 2 0 1 0 0 0 2 1]
[0 0 0 0 0 0 0 1 1 0 2 1 1 0 1 0 0 2 1 1 2 1 0 0 2 2 2 2 1 0]
[0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 2 0 2 2 2 2 0 2 0 0 0 1 2 2 1 2]
sage: m=vector([0,0,2,1,2,0,2,2,0])
sage: m *G
(0, 0, 2, 1, 2, 0, 2, 2, 2, 0, 0, 2, 0, 2, 0, 0, 1, 2, 0, 1, 2, 2, 0, 2, 1, 2, 1, 1, 1, 1)
```

Example 2.17. *Find all the codewords of the linear code C with generator matrix*

$$G = \begin{pmatrix} 2 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 0 \\ 2 & 1 & 2 & 0 & 0 & 0 & 0 & 2 & 0 \end{pmatrix}$$

```
sage: G=matrix([[2, 0, 0, 0, 1, 2, 0, 1, 0],[2, 1, 2, 0, 0, 0, 0, 2, 0]])
sage: G
[2 0 0 0 1 2 0 1 0]
[2 1 2 0 0 0 0 2 0]
sage: C = LinearCode(G)
sage: C
Linear code of length 9, dimension 2 over Integer Ring
sage: M=matrix([[0, 1, 0, 2, 0, 1, 2, 1, 2],[0, 0, 1, 0, 2, 1, 2, 2, 1]])
sage: M
[0 1 0 2 0 1 2 1 2]
[0 0 1 0 2 1 2 2 1]
sage: transpose(M)*G
[0 0 0 0 0 0 0 0 0] (= c1)
[2 0 0 0 1 2 0 1 0] (= c2)
```

```
[2 1 2 0 0 0 0 2 0] (= c3)
[4 0 0 0 2 4 0 2 0] (= c4)
[4 2 4 0 0 0 0 4 0] (= c5)
[4 1 2 0 1 2 0 3 0] (= c6)
[8 2 4 0 2 4 0 6 0] (= c7)
[6 2 4 0 1 2 0 5 0] (= c8)
[6 1 2 0 2 4 0 4 0] (= c9)
```

Example 2.18. *Consider the finite field* $\mathbb{F}_8 = \frac{\mathbb{F}_2[x]}{x^3+x+1}$. *Find a linear code* C $[6,3]$ *over* $\mathbb{F}_8$. *Find the minimun distance of* C.

```
sage: R = GF(8,name='x'); R
sage: for i,x in enumerate(R): print i,x
0 0
1 a
2 a^2
3 a + 1
4 a^2 + a
5 a^2 + a + 1
6 a^2 + 1
7 1
sage: C=RandomLinearCode(6,3 ,R);
sage: C
Linear code of length 6, dimension 3 over Finite Field in a of size 2^3
sage: G=C.gen_mat()
sage: G
[          1           0           0           a           a       a + 1]
[          0           1           0         a^2           1 a^2 + a + 1]
[          0           0           1           1       a + 1           a]
sage: C.minimum_distance()
3
```

2.8 Problems

Exercise 5. Show that if C is a binary linear code, then the code obtained by adding an overall parity check to C is also linear.

Exercise 6. Give an example of a code which is not linear.

Exercise 7. Prove that in a binary linear code, either all the codewords have even weight or exactly half have even weight and half have odd weight.

Exercise 8. Determine $A_q(n, d)$ and write or describe the corresponding code that achieves the upper bound.

1. $A_2(8, d)$ for $d = 1$.
2. $A_2(n, 4)$ for $n = 4$ and $n = 5$.
3. $A_q(4, 3)$ for $q = 2$ and $q = 3$.

3 Decoding problem and syndrome decoding

In this chapter

- Decoding with standard array
- Syndrome decoding
- Decoding binary Hamming code

3.1 Decoding problem

In Lecture I we have seen that informally we can state the decoding problem as the problem of recover, from the received word, the codeword $\mathbf{c}$ associated to the message $\mathbf{m}$ that has been sent. *Maximum likelihood decoding* refers to any method for decoding in which a received word is decoded as one of the codewords which is most likely to have been sent.

Example 3.1. *Suppose that the space of messages is* $(\mathbb{F}_2)^2$:

$$00 = \mathbf{m}_1, \qquad 01 = \mathbf{m}_2, \qquad 10 = \mathbf{m}_3, \qquad 11 = \mathbf{m}_4.$$

Let C *be the* $[6, 2]_2$ *code generated by*

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Then:

$$C = \{000000, 111111, 000111, 111000\}.$$

To send $\mathbf{m}_3 = 10$ *we transmit the word* $\mathbf{m}_3 G = 000111$*; typically, during the transmission the message gets distorted by noise and the receiver has to perform some operations to obtain the transmitted word. Let* $\mathbf{w}$ *be the received vector. Several different situations could come up:*

1. $\mathbf{w} = 000111$*, then* $\mathbf{w} \in C$*, so the receiver deduces correctly that no errors have occurred and no correction is needed. It concludes that the message was* $\mathbf{m}_3$*.*

2. $\mathbf{w} = 000101 \notin C$*, then the receiver concludes that some errors have occurred. In this case it may "correct" and "detect" the error as follows. It may suppose that the word transmitted was* 000111*, since that is the word that differs in the least number of positions from the received word* w*.*

3. $\mathbf{w} = 000100 \notin C$*. The receiver correctly reaches the conclusion that there were some errors during the transmission, but if it tries to correct as in the previous case, it concludes that the word "nearest" to* $\mathbf{w}$ *is* 000000*. In this case it corrects in a wrong way.*

4. $\mathbf{w} = 000000 \in C$*. The receiver deduces incorrectly that no errors have occurred and that the message is* $\mathbf{m}_1$*.*

When the decoder gets a received word which is not a codeword, it has to find the codeword in C which has been sent by the encoder, *i.e.*, among all codewords, it has to find the one which has the "highest probability" of being sent. To do this it needs to know the type of channel used for transmission, more precisely it needs to know how the noise can modify the transmitted word.

We usually assume the following channel.

Definition 3.1.1. A **q-ary symmetric channel** (**SC** for short) is a channel with the following properties:

a) the components of a transmitted word (an element of $\mathbb{F}_q$ that here we name generally "symbol") can be changed by the noise only to another element of $\mathbb{F}_q$;
b) the probability that a symbol becomes another one is the same for all pairs of symbols;
c) the probability that a symbol changes during the transmission does not depend on its position;
d) if the i-th component is changed, then this fact does not affect the probability of change for the j-th components, even if j is close to i.

To these channel properties it is usually added a *source property*:

- *all words are equally likely to be transmitted.*

The q-ary SC is a model that rarely can describe real channels. For example the assumption d) is not reasonable in practice: if a symbol is corrupted during the transmission there is an high probability that some errors happened in the neighbourhood. Despite this fact, the classical approach accepts the assumptions of the SC, since it permits a simpler construction of the theory. The ways of getting around the troubles generated by this "false" assumption in practice are different by case and these are not investigated here. From now we will assume that the channel is a SC, and that the probability that a symbol changes to another is less than the probability that it is uncorrupted by noise.

In case of q-ary SC, any codeword whose distance to the received word $\mathbf{r}$ is minimum, i.e. it is a *nearest neighbour* of $\mathbf{r}$ in the Hamming metric, is also the codeword which is most likely to have been sent (see Lecture IV). Hence, if the communication channel is the q-ary SC, MLD is equivalent to *nearest neighbour decoding*. Formally,

Proposition 3.1.2. If the transmission uses a q-ary SC and the probability that a symbol changes into another one is less then the probability that a symbol is uncorrupted by noise, the word sent with the highest probability is the word "nearest" (in the sense of Hamming distance) to the received vector. If no more then t (the error correction capability, see Lecture I) errors have occurred, this word is unique.

Proof See Lecture IV. ■

In the next section we formally describe the decoding procedure in the linear case.

3.2 Decoding linear codes

Let $\mathbf{c}, \mathbf{e}, \mathbf{r} \in (\mathbb{F}_q)^n$ be the transmitted codeword, the error, and the received word, respectively. Then:

$$\mathbf{c} + \mathbf{e} = \mathbf{r}.$$

Given $\mathbf{r}$, our goal is to determine an $\mathbf{e}$ of minimal weight such that $\mathbf{r} - \mathbf{e}$ is in C (why?). Of course, this error vector might not be unique, since there may be more than one word nearest to $\mathbf{r}$, but if the weight of $\mathbf{e}$ is less then t (the error correction capability of the code), then it is unique.

Thus the decoding problem has become that of finding an efficient algorithm that will correct up to t errors. The most obvious decoding technique is to use the "brute force" method and examine all codewords until we find a codeword with distance t or less from the received word. Obviously this approach is impractical when the dimension k is large. Another possible strategy is to make a table consisting of a nearest codeword for each of the q^n elements in $\mathbb{F}_q$ and then look up a received vector in the table in order to decode it. This is impractical if q^n is very large.

For an $[n, k, d]_q$ linear code C, we can describe an algorithm using a table with q^{n-k}, rather than q^n entries , where one can find the nearest codeword by looking up one of these q^{n-k} entries. This general decoding algorithm for linear codes is called **syndrome decoding**.

3.2.1 Cosets

In order to describe a general decoding algorithm for linear codes, we recall some basic facts and definition.

Definition 3.2.1. Let C be an $[n,k,d]_q$ code and $\mathbf{a} \in (\mathbb{F}_q)^n$. We define the **coset** of C determined by $\mathbf{a}$ as

$$\mathbf{a}+C = \{\mathbf{a}+\mathbf{c} \quad | \quad \mathbf{c} \in C\}.$$

Example 3.2. *Let* $C = \{000, 111\}$ *and* $\mathbf{a} = 101$. *Then* $C + 101 = \{000 + 101, 111 + 101\} = \{101, 010\}$.

If C is a linear code, then one may think that there are as many cosets of C as the different elements in $\mathbb{F}_q^n$. This is not true, as stated by the following proposition.

Proposition 3.2.2. Let C be an $[n,k,d]_q$ linear code, and $\mathbf{v}, \mathbf{u} \in \mathbb{F}_q^n$. Then:

1. If $\mathbf{u} \in \mathbf{v}+C$, then $\mathbf{v}+C = \mathbf{u}+C$, i.e. each word in the coset determines that coset;
2. $\mathbf{u} \in \mathbf{u}+C$
3. If $\mathbf{u}-\mathbf{v} \in C$, the $\mathbf{v}$ and $\mathbf{u}$ are in the same coset;
4. Every element of $\mathbb{F}_q^n$ is contained in one and only one coset, i.e. either $\mathbf{v}+C = \mathbf{u}+C$ or these cosets are disjoint,
5. $|\mathbf{v}+C| = |C| = q^k$,
6. There are q^{n-k} different cosets
7. C is a coset.

Definition 3.2.3. Let C be an $[n,k,d]_q$ code. For any coset $\mathbf{a}+C$, $\mathbf{a} \in \mathbb{F}_q^n$, and any vector $\mathbf{v} \in \mathbf{a}+C$, we say that $\mathbf{v}$ is a **coset leader** if it is an element of minimum weight in the coset.

Example 3.3. *We list all the cosets of the code* $C = \{0000, 1011, 0101, 1110\}$*:*
First, note that the code has dimension $k = 2$ *(why?). From previous proposition, we know that* C *is a coset and every codeword determine the coset* C*, so*

$$C_1 = C = \{0000, 1011, 0101, 1110\}.$$

Then we can proceed taking an element in $\mathbb{F}_2^4$ *not in* C*. For example* 0100 *(for decoding purpose we pick an element of smallest weight possible). Then*

$$C_2 = 0100 + C = \{0100, 1111, 0001, 1010\}.$$

Now consider another element in $\mathbb{F}_2^4$ *that is not* C_1 *or* C_2*, say* 1000.

$$C_3 = \{1000, 0011, 1101, 0110\}.$$

Proceeding us before, we conclude with

$$C_4 = 0010 + C = \{0010, 1001, 0111, 1100\}.$$

The previous proposition suggests us the following decoding procedure. Notice that if $\mathbf{c}$ is the transmitted codeword and $\mathbf{r}$ is received, then the error is $\mathbf{e} = \mathbf{r} - \mathbf{c}$. This means that $\mathbf{r} + \mathbf{e} = \mathbf{c} \in C$. Once we receive $\mathbf{r}$, we choose a word $\mathbf{e}$ of minimum weight in the coset $\mathbf{r} + C$, and then conclude that $\mathbf{c} = \mathbf{e} + \mathbf{r}$ is the codeword that has been sent.

Example 3.4. *Consider the code C_1 in Example 3.3. The cosets of C are given in Table 3.1. Now suppose that $\mathbf{r} = 1101$. The word of least weight in the coset $C + \mathbf{r}$ is 1000, that we set to $\mathbf{e}$. Then we obtain $\mathbf{c} = 1101 + 1000 = 0101$.*

Table 3.1: Cosets of C (Example 3.4)

0000	1000	0100	0010
1011	0011	1111	1001
0101	1101	0001	0111
1110	0110	1010	1100
C	1000+C	0100+C	0010+C

Notice that in the coset $\{0100, 1111, 0001, 1010\}$ there are two words of smallest weight, i.e. the coset leader is not unique. This is correct as the distance of the code is 2, so C is 1-error detecting, but 0-error correcting.

The table having the cosets of C as columns is called **standard array**.

3.2.2 Syndrome decoding

The previous decoding procedure is quite expensive, because it requires to store an array of all the cosets. A nice property of parity-check matrix is that it can be used for an efficient implementation of nearest neighbour decoding. Let $\mathbf{r} = \mathbf{c} + \mathbf{e}$ be the received word. If we apply the parity check matrix H to $\mathbf{r}$, we get:

$$H\mathbf{r} = H(\mathbf{c} + \mathbf{e}) = H\mathbf{c} + H\mathbf{e} = (\text{why?})H\mathbf{e} = \mathbf{s}.$$

The vector $\mathbf{s} \in \mathbb{F}_q^{n-k}$ is called the **syndrome** associated to $\mathbf{r}$ (or to $\mathbf{e}$).

Definition 3.2.4. If $\mathbf{s}$ is a syndrome corresponding to an error of weight $\text{wt}(\mathbf{e}) \leq t$, then we say that $\mathbf{s}$ is a **correctable syndrome**.

If we transmit another codeword $\mathbf{c}'$ and the same error $\mathbf{e}$ occurs, we get

$$\mathbf{s}' = H(\mathbf{c}' + \mathbf{e}) = H\mathbf{c}' + H\mathbf{e} = H\mathbf{e} = \mathbf{s}.$$

So the syndrome does not depend on the specific codeword that has been sent through the channel, but only on the error occurred during the transmission. Actually the syndrome have a more geometric meaning, as we are going to see.

Proposition 3.2.5. Let C be an $[n,k,d]_q$ code and $\mathbf{x},\mathbf{y} \in \mathbb{F}_q^n$. Then $\mathbf{x}$ and $\mathbf{y}$ are in the same coset if and only if they have the same syndrome. Moreover $\mathbf{c} \in C$ if and only if the syndrome associated to $\mathbf{c}$ is the zero vector.

Proof Exercise. ■

Given a table containing all the pairs $(\mathbf{e}, \mathbf{s_e})$, where $\mathbf{e}$ is the coset leader of its coset and $\mathbf{s_e}$ is the associated syndrome, the decoding with the standard array proceeds as follows:

- given $\mathbf{r}$, compute the syndrome $\mathbf{s} = H\mathbf{r}$
- find the corresponding coset leaders $\mathbf{e}$ in the standard array table
- output $\mathbf{r} - \mathbf{e}$.

The computational cost of this procedure is much smaller than the computational cost of the brute force method, as in the standard array we need to store just the syndromes and the coset leaders. Summing up, it requires the computation of the standard array (in its reduced form), to be done once and for all, and of the syndrome for each received word. The main disadvantage of using the standard array is that the coset leader for each coset has to be memorized. Notice that we store in the table only coset leader of weight at most t (why?).

Example 3.5. *Let C be the binary linear code considered in the previous example. A parity-check matrix for C is*

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix},$$

and a standard array is given in Table 3.2. Suppose to send the codeword $\mathbf{c} = 1011$ which

Table 3.2: Standard array with corresponding syndromes

coset leader	0000	1000	0100	0010
	1010	0011	1111	1001
	1101	1101	0001	0111
	0111	0110	1010	1100
syndrome	00	11	01	10

corresponds to the message $\mathbf{m} = 10$. Let $\mathbf{r} = 1111$ be the received word, its corresponding syndrome is 01. The coset leader associated to this syndrome is 0100, hence $\mathbf{c} = 1111 - 0100 = 1011$.

We have seen in Proposition 3.2.2 that the number of different cosets for a linear $[n,k]_q$ code is q^{n-k}. Each syndrome associated to C is an element in $\mathbb{F}_q^{n-k}$, and since $|\mathbb{F}_q^{n-k}| = q^{n-k}$, so the set of syndromes is the entire space $\mathbb{F}_q^{n-k}$.

Example 3.6. *Consider the usual* $[7,4]_2$ *Hamming code with parity-check matrix* H*, and suppose* $\mathbf{r} = 1011000$ *is the received word. Then the associated syndrome is*

$$\mathbf{s} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.$$

Interestingly, we note that the syndrome corresponds to the 6th column of H*. More generally for binary Hamming code, if a single error occurred, say in the* i*th component, then the error vector is* $\mathbf{e}_i$ *(the* i*th vector of the standard basis), and its associated syndrome,* $H\mathbf{e}_i$*, is exactly the* i*th column of* H*. This means that if* $\mathbf{r}$ *is received, the sent codeword is* $\mathbf{r} - \mathbf{e}_6 = 1011010$*.*

Summing up, the decoding procedure for binary Hamming code is as follows: suppose a single error occurred in the ith component, then $\mathbf{s} = H\mathbf{e}_i = \mathbf{d}_i$, where $\mathbf{d}_i$ is the ith column of H.

- Compute the syndrome $\mathbf{s} = H\mathbf{r}$;
- Find the d_i column of H that matches the syndrome;
- Complement the ith bit of the received word.

3.3 Problems

Exercise 9. Construct standard arrays for binary codes having each of the following generator matrices:

$$G_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad G_2 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad G_3 = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Exercise 10. Using the third array of the previous Problem:

(a) Decode the received vectors 11111 and 01011;

(b) Give examples of

i. Two errors occurring in a codeword and being corrected,

ii. two errors occurring in a codeword and not being corrected.

4 Basic probability theory

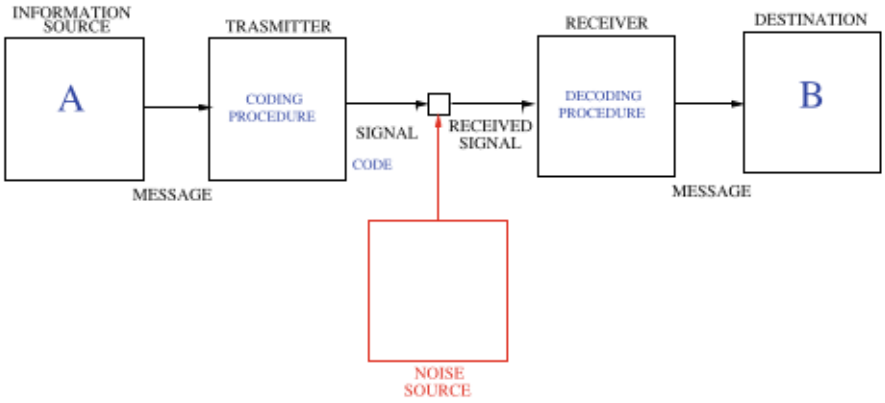
In this chapter

- Discrete probability theory

4.1 Introduction

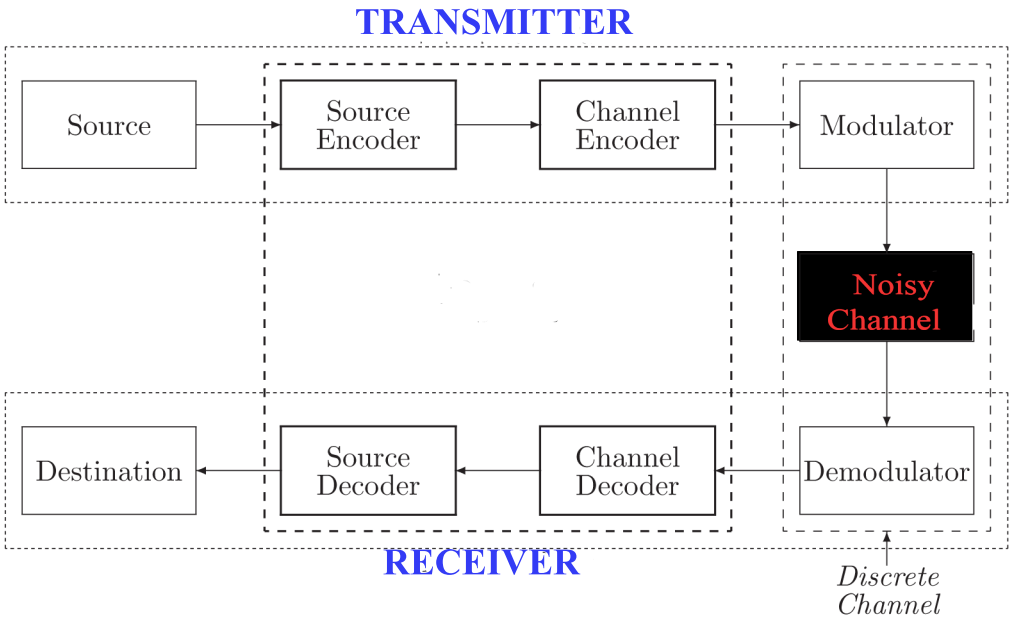
In Lecture I we have seen a communication schema (see Figure 4.1) in which a source sends a

Figure 4.1: A communication schema



message, say m, to a *transmitter*, that uses what we generically called a *coding procedure* to transform the original message m in a longer message, say c, by adding some redundancy. Then the transmitter sends c through a noisy channel to another device called *receiver*. Because of the noise, it could be that the received message, say r, is different from c. Goal of the receiver is to apply some *decoding procedure* in order to recover the original message m.
Here we would like to analyse the transmitter and receiver in more details (see Figure 4.2). A

Figure 4.2: A more detailed communication schema



typical transmitter is composed of two components:

- *Source encoder*: it represents the source message in a compact fashion by removing unnecessary content (*data compression*);

- *Channel encoder*: it outputs a "channel codeword", introducing a systematic redundancy to tolerate errors that might be introduced by the channel (*error correcting encoding*).

The channel codeword is then transformed by the *Modulator encoder* into waveforms suitable for transmission over the physical channel. The receiver part consists of a *demodulator*, a *channel decoder* and a *source decoder* that perform the reverse operations.

The main role of information theory is that of providing a mathematical framework for the theory of communication by establishing the fundamental limits on the performance of various communication systems. More precisely, it permits to answer two fundamental questions:

- Given a source, how much can we compress? Are there any limits for compression of information?
- Given a noisy channel, how much redundancy is necessary to minimize error probability in decoding?

In the next lectures we try to explain how to answer these questions.

First we need to introduce some basic definitions and mathematical concepts required for subsequent development of theory.

4.2 Discrete probability

In this section we briefly recall some fundamental definitions and results of discrete probability.

Suppose to toss a coin once, the result can be either H (head) or T (tail), i.e. one element of the set $\{H, T\}$. If a 6-sided die is rolled once, the result is clearly one element of the set $\{1, 2, 3, 4, 5, 6\}$. More generally, we have the following definition.

Definition 4.2.1. The **sample space**, usually denoted by Ω, is the set of all possible outcomes of a process/experiment. Any subset A of Ω is called an **event**.

In the first example above, the sample space is $\Omega_1 = \{H, T\}$, and in the second it is $\Omega_2 = \{1, 2, 3, 4, 5, 6\}$.

Example 4.1. *Suppose to toss a coin two times, then*

$$\Omega = \{HH, HT, TH, TT\},$$

where, for example, HT means the first toss is a head and the second is a tail. If E is the event "the second toss is a tail" then

$$E = \{HT, TT\} \subset \Omega.$$

Definition 4.2.2. Let $\Omega = \{s_1, \ldots, s_n\}$ be a sample space and x the outcome of a process which takes one of the possible values s_i, $i = 1, \ldots, n$. For each s_i it is associated a real number p_i, such that

$$\Pr(s_i) = \Pr(x = s_i) = p_i$$

$$0 \leq p_i \leq 1 \quad \text{and} \quad \sum_{i=1}^{n} p_i = 1.$$

The sequence $p_1, \ldots, p_n$ is called a **probability distribution** for Ω and $\Pr : \Omega \to \mathbb{R}$ is a **probability law** for Ω.

Remark If E is a subset of Ω, then $\Pr(E) = \Pr(x \in E) = \sum_{s_i \in E} \Pr(x = s_i)$.

Definition 4.2.3. If the sample space Ω consists of n possible outcomes which are equally likely, i.e. $p_i = p_j, \forall i, j \in \{1, \ldots, n\}$, then the probability of any event E is given by

$$\Pr(E) = \frac{|E|}{|\Omega|} = \frac{|E|}{n}.$$

Proposition 4.2.4. Let Ω be a finite sample space and $E \subseteq \Omega$ an event. Then it holds:

1. $\Pr(E^c) = 1 - \Pr(E)$, where E^c denotes the complementary set of E.
2. If E and F are two **mutually exclusive** events in Ω, i.e. $E \cap F = \emptyset$, then

$$\Pr(E \cup F) = \Pr(E) + \Pr(F).$$

This property can be easily extended to more than two events.

Example 4.2. *A fair 6-sided die is rolled once. Consider the events*
$E = \{$*the result is* $1\}$
$E^c = \{$*the result is* $\underline{not}\ 1\} = \{$*the result is* $2, 3, 4, 5, 6\}$.
Then $\Pr(E) = 1/6$ *and* $\Pr(E^c) = 1 - \Pr(E) = 5/6$.

Example 4.3. *Consider an experiment involving a single coin toss. There are two possible outcomes, H and T. The sample space is* $\Omega = \{H, T\}$. *If heads and tails are "equally likely" we should assign equal probabilities to the two possible outcomes:* $\Pr(H) = \Pr(T) = 0.5$. *The additivity property implies that* $\Pr(\{H, T\}) = \Pr(H) + \Pr(T) = 1$, *and the probability law is given by*

$$\Pr(\{H, T\}) = 1 \quad \Pr(H) = 0.5 \quad \Pr(T) = 0.5 \quad \Pr(\emptyset) = 0.$$

Example 4.4. *Consider another experiment involving three coin tosses. The outcome will now be a 3-long string of heads or tails. The sample space is*

$$\Omega = \{HHH, HHT, HTH, HTT, TTT, TTH, THT, THH\}.$$

We can assume that each possible outcome has the same probability of $1/8$ *(why?). Consider now the event*

$$E = \{\textit{exactly 2 heads occur}\} = \{HHT, THH, HTH\}.$$

Then

$$\Pr(E) = \Pr(HHT) + \Pr(THH) + \Pr(HTH) = 3/8.$$

Exercise: Consider an experiment with 3 possible outcomes: a, b, c. In which of the following cases we have a plausible probability distribution? Why?

1. $\Pr(a) = \Pr(b) = \Pr(c) = 1/3$
2. $\Pr(a) = 0.66, \Pr(b) = 0.38, \Pr(c) = -0.04$
3. $\Pr(a) = 0.35, \Pr(b) = 0.52, \Pr(c) = 0.26$.

In the following proposition we list a number of properties of a probability law over a sample space Ω.

Proposition 4.2.5. Let E, F, G be events:

(a) If $E \subset F$ then $\Pr(E) \leq \Pr(F)$

(b) $\Pr(E \cup F) = \Pr(E) + \Pr(F) - \Pr(E \cap F)$

(c) $\Pr(E \cup F \cup G) = \Pr(E) + \Pr(E^c \cap F) + \Pr(E^c \cap F^c \cap G)$

Conditional probability

Sometimes we need to consider the probability of an event E occurs knowing that an event F has already occurred. This probability is denoted by $\Pr(E|F)$, meaning probability of E given F.

Example 4.5. *A fair 6-sided die is rolled once. We consider the following two events*
$E = \{$*the result is a number less than 4*$\} = \{1, 2, 3\}$
$F = \{$*the result is an odd number*$\} = \{1, 3, 5\}$.
We want to compute $\Pr(E|F)$*, i.e. the probability to get a number less than* 4*, knowing that it is an odd number. We know that*

$$\Pr(E) = 1/2 \qquad \Pr(F) = 1/2.$$

Roughly speaking, if we know that F *has occurred, then we have only 3 possible results, that is the sample space is "reduced" to* $\{1, 3, 5\}$ *and the probability of obtaining a number less than* 4 *is now* $1/3 + 1/3 = 2/3 = \Pr(E|F)$.
More formally we note that:

$$E \cap F = \{1, 3\}, \qquad \Pr(E \cap F) = 2/6 = 1/3,$$

and that

$$\Pr(E|F) = \frac{\Pr(E \cap F)}{\Pr(F)} = 2/3.$$

Generalizing the argument of previous example we introduce the following definition.

Definition 4.2.6. The **conditional probability** of an event E given an event F with $\Pr(F) > 0$, is defined by

$$\Pr(E|F) = \frac{\Pr(E \cap F)}{\Pr(F)}.$$

From this definition we easily obtain

Proposition 4.2.7 (Multiplication rule). Let E and F be two events, then

$$\Pr(E \cap F) = \Pr(F) \cdot \Pr(E|F) \quad \text{if} \quad \Pr(F) > 0$$

and

$$\Pr(E \cap F) = \Pr(E) \cdot \Pr(F|E) \quad \text{if} \quad \Pr(E) > 0.$$

Example 4.6. *Consider a box containing 15 red balls and 5 green balls. Let $E = \{$a red ball is extracted$\}$ and $F = \{$a green ball is extracted$\}$ be two events. We extract two balls consecutively and we want to compute the probability that first a red ball and then a green ball are extracted. Easily we can see that:*

$$\Pr(E) = 15/20 \quad \textit{and} \quad \Pr(F|E) = 5/19.$$

According to the multiplication rule we have:

$$\Pr(E \cap F) = \Pr(E) \cdot \Pr(F|E) = 15/20 \cdot 5/19 = 15/76.$$

Notice that $\Pr(E \cap F)$ here denotes the probability of obtaining in two extractions a red ball and a green one. In the same way we can compute the probability that first a green ball and then a red ball are extracted:

$$\Pr(F) = 5/20 \quad \textit{and} \quad \Pr(F|E) = 15/19.$$

Then

$$\Pr(E \cap F) = \Pr(F) \cdot \Pr(E|F) = 5/20 \cdot 15/19 = 15/76.$$

An important special case arise when the occurrence of event F does not change the probability that E occurs. More formally,

Definition 4.2.8. Two events E and F are **independent** if

$$\Pr(E|F) = \Pr(E).$$

In this case we also have that

$$\Pr(F|E) = \Pr(F).$$

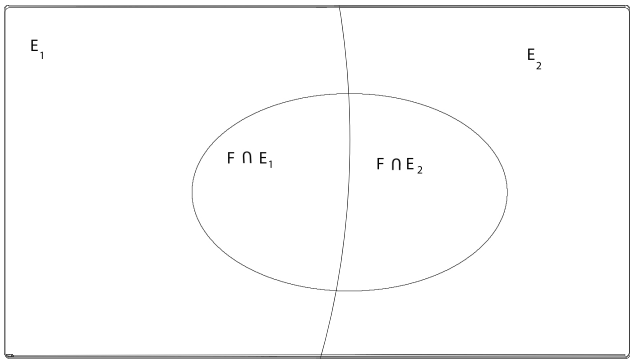
We can rewrite Proposition 4.2.7 in case of independent events:

Proposition 4.2.9. Given two independent events E and F, then

$$\Pr(E \cap F) = \Pr(E) \cdot \Pr(F).$$

Let us consider the Venn diagram in Figure 4.3.

Figure 4.3:



The events $F \cap E_1$ and $F \cap E_2$ are mutually exclusive, then

$$\Pr(F) = \Pr(F \cap E_1) \cup \Pr(F \cap E_2) = \Pr(F \cap E_1) + \Pr(F \cap E_2)$$

and, by applying the multiplication rule (Proposition 4.2.7), we get

$$\Pr(F) = \Pr(E_1) \cdot \Pr(F|E_1) + \Pr(E_2) \cdot \Pr(F|E_2).$$

This formula expresses the total probability rule in case of two events E_1 and E_2. We can generalize it as follows.

Theorem 4.2.10 (Total Probability Theorem). Let $E_1, \ldots, E_n$ be disjoint events that form a partition of the sample space (this means that each possible outcome is included in one and only one of the events $E_1, \ldots, E_n$) and assume $\Pr(E_i) > 0$, for all $i = 1, \ldots, n$. Then for any event F, it holds

$$\Pr(F) = \Pr(F \cap E_1) + \cdots + \Pr(F \cap E_n) = \Pr(E_1) \cdot \Pr(F|E_1) + \cdots + \Pr(E_n) \cdot \Pr(F|E_n)$$

Example 4.7. *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be two boxes, such that $\mathcal{B}_1$ contains 2 red and 1 black balls, and $\mathcal{B}_2$ contains 3 red and 2 black balls. We choose one box at random and extract from it a ball. We want to compute the probability that the extracted ball is black.*

We can formalize this problem as follows: let E_1, E_2 be the events

$E_1 = \{\mathcal{B}_1$ *is the chosen box* $\}$

$E_2 = \{\mathcal{B}_2$ *is the chosen box* $\}$.

Then it holds

$$E_1 \cap E_2 = \emptyset \qquad E_1 \cup E_2 = \Omega.$$

Now let F be the event $\{$a black ball is extracted$\}$. By applying the total probability theorem we get:

$$\Pr(F) = \Pr(E_1) \cdot \Pr(F|E_1) + \Pr(E_2) \cdot \Pr(F|E_2).$$

We have that

$$\Pr(E_1) = \Pr(E_2) = 1/2 \qquad \Pr(F|E_1) = 1/3 \qquad \Pr(F|E_2) = 2/5,$$

hence we obtain

$$\Pr(F) = 1/3 \cdot 1/2 + 2/5 \cdot 1/2 = 11/30 \approx 0.367.$$

Example 4.8. *Consider the same situation of previous example. We now want to compute the probability that $\mathcal{B}_1$ is the chosen box, knowing that the extracted ball is black.*

Clearly we have to compute $\Pr(E_1|F)$. *Then from Proposition 4.2.7, we obtain*

$$\Pr(E_1|F) \cdot \Pr(F) = \Pr(F|E_1) \cdot \Pr(E_1)$$

and from this

$$\Pr(E_1|F) = \frac{\Pr(F|E_1) \cdot \Pr(E_1)}{\Pr(F)} = 5/11.$$

From the previous example we can derive the following important result. It is often used to relate conditional probabilities of the form $\Pr(E|F)$ with conditional probabilities of the form $\Pr(F|E)$.

Theorem 4.2.11 (Bayes' rule). Let $E_1, E_2, \ldots, E_n$ be disjoint events that form a partition of the sample space Ω, and assume that $\Pr(E_i) > 0$, for all i. Then, for any event F such that $\Pr(F) > 0$, we have

$$\Pr(E_i|F) = \frac{\Pr(F|E_i) \cdot \Pr(E_i)}{\Pr(F)} = \frac{\Pr(F|E_i) \cdot \Pr(E_i)}{\Pr(E_1) \cdot \Pr(F|E_1) + \cdots + \Pr(E_n) \cdot \Pr(F|E_n)}.$$

4.3 Decoding and likelihood principle

Using usual notation, let $\mathbf{c} \in C$ be the sent codeword and $\mathbf{r}$ the received word. Then $\Pr(\mathbf{c})$ denotes the probability that $\mathbf{c}$ is sent and $\Pr(\mathbf{r})$ the probability that $\mathbf{r}$ is received. As pointed out in Lecture I, there are different ways by which a decoder can recover the sent codeword. For example:

- The decoder could choose $\bar{\mathbf{c}} = \mathbf{c}$ for the codeword $\mathbf{c} \in C$ such that $\Pr(\mathbf{c}|\mathbf{r})$ is maximum. Such a decoder is called **maximum a posteriori probability** decoder.
- Alternatively, the decoder could choose $\bar{\mathbf{c}} = \mathbf{c}$ for the codeword $\mathbf{c}$ such that the probability $\Pr(\mathbf{r}|\mathbf{c})$ is maximum. Such a decoder is called a **maximum likelihood** (ML) decoder.

Let us consider a Binary Symmetric Channel (BSC). Let $\mathbf{r} = r_1 \ldots r_n$ and $\mathbf{c} = c_1 \ldots c_n$, then, as in this type of channel the error probabilities in different components are independent (See Definition of q-ary symmetric channel in Lecture III), then

$$\Pr(\mathbf{r}|\mathbf{c}) = \prod_{i=1}^{n} \Pr(r_i|c_i).$$

Moreover from Figure 1 Lecture I, we know that:

$$\begin{cases} \Pr(r_i|c_i) = p & \text{if } r_i \neq c_i \\ \Pr(r_i|c_i) = 1 - p & \text{if } r_i = c_i \end{cases}$$

Then we obtain:

$$\Pr(\mathbf{r}|\mathbf{c}) = p^{d(\mathbf{r},\mathbf{c})}(1-p)^{n-d(\mathbf{r},\mathbf{c})} = (1-p)^n \left(\frac{p}{1-p}\right)^{d(\mathbf{r},\mathbf{c})},$$

where $d(\mathbf{r},\mathbf{c})$ denotes the Hamming distance between $\mathbf{r}$ and $\mathbf{c}$. Since $0 < p < 1/2$ and $0 < p/(1-p) < 1$, we can deduce that maximizing $\Pr(\mathbf{r}|\mathbf{c})$ is equivalent to minimizing $d(\mathbf{r},\mathbf{c})$, that is finding the codeword $\mathbf{c}$ closest to the received vector $\mathbf{r}$ in Hamming distance. This means that on a BSC, maximum likelihood decoding and nearest decoding are the same. This result can be generalized to the q-ary symmetric channel.

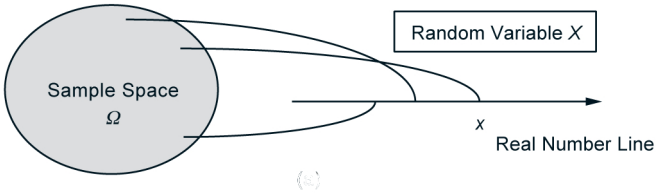
4.4 Random variables

In many probabilistic models the outcomes of some experiments are of a numerical nature; in other experiments the outcomes are not numerical, but they may be associated with some numerical values of interest: for example, the outcome when a coin is tossed can be 'heads' or 'tails'. However we often want to represent outcomes as numbers. A **random variable** is a function that associates a unique numerical value with every outcome of an experiment. The value of the random variable will vary from trial to trial as the experiment is repeated.

Given an experiment and the corresponding set of possible outcomes (the sample space), a random variable associates a particular number with each outcome. We refer to this number as the numerical value of the random variable. Mathematically, a random variable is a real-valued function of the experimental outcome:

$$X : \Omega \to \mathbb{R}.$$

Figure 4.4: Visualization of a random variable: it is a function that assigns a numerical value to each possible outcome of the experiment



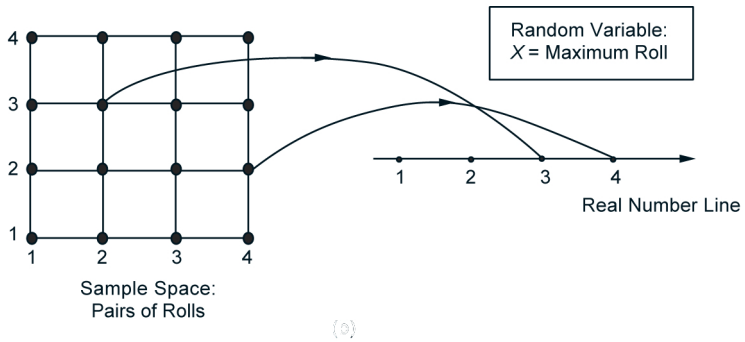
Example 4.9.

1. *Consider 3 tosses of coins. The number of heads in the sequence is a random variable, i.e. $X(H) = m, m \in \mathbb{R}$ defines a random variable. But the 3-long sequence of heads and tails is not consider a random variable because it does not have an explicit numerical value.*

2. *Consider two rolls of a 4-sided die. Let X be the maximum of the two rolls. If the outcome is $(4,2)$ then the value of X is 4, if the outcome is $(2,3)$, the value of X is 3.*

3. *Consider the transmission of a message over a noisy channel, the time needed to transmit the message, the number of symbols received in error, and the delay with which the message is received are all random variables.*

Figure 4.5: Example of a random variable



A random variable is called **discrete** if its range (the set of values that it can take) is finite or at most countably infinite. For example, all the random variables mentioned in examples above can take at most a finite number of numerical values, and therefore they are discrete.

The most important way to characterize a random variable is through the probabilities of the values that it can take. For a discrete random variable X, these are captured by the *probability mass function* (probability for short) of X, denoted by p_X . In particular, if x is any possible value of X, the probability mass of x, denoted $p_X(x)$, is the probability of the event $\{X = x\}$ consisting of all outcomes that give rise to a value of X equal to x:

$$p_X(x) = \Pr(X = x).$$

Example 4.10. *Let X be the random variable defined as the number of heads H in two coin tosses. Then*

$$p_X(2) = \Pr(X = 2) = 1/4, \qquad \Pr(X = 1) = 1/2,$$
$$\Pr(1 < X < 2) = 0, \qquad \Pr(0 \leq X \leq 2) = 1 \qquad \Pr(1 < X \leq 2) = 1/4.$$

In some probabilistic models several random variables might be involved. This motivates us to consider probabilities involving simultaneously these random variables and to investigate their mutual coupling.

Definition 4.4.1. Let X and Y be two discrete random variables associated with the same experiment. The **joint probability** of X and Y is defined as

$$p_{X,Y}(x, y) = \Pr(X = x, Y = y),$$

for all pairs of numerical values (x, y) that X and Y can take.

The joint probability determines the probability of any event that can be specified in terms of X and Y. If E is the set of all values (x, y) which satisfy certain property, then

$$\Pr((X, Y) \in E) = \sum_{(x,y)\in E} p_{X,Y}(x, y).$$

It must be the case that $p_{X,Y}(x,y) \geq 0$ and $\sum_{(x,y)\in E} p(x,y) = 1$.

We can compute the probability of X and Y using the following formulas:

$$p_X(x) = \sum_y p_{X,Y}(x,y), \qquad p_Y(y) = \sum_x p_{X,Y}(x,y).$$

We can verify these relations as follows. For example, let us consider $p_X(x)$:

$$\begin{aligned} p_X(x) &= \Pr(X = x) \\ &= \sum_y \Pr(X = x, Y = y) \\ &= \sum_y p_{X,Y}(x,y), \end{aligned}$$

where the second equality follows by noting that the event $\{X = x\}$ is the union of the disjoint events $\{X = x, Y = y\}$ as y ranges over all the different values of Y . The formula for $p_Y(y)$ is verified similarly. We sometimes refer to p_X and p_Y as the **marginal probability**, to distinguish them from the joint probability.

We can visualize the marginal probabilities from the joint probability using the tabular method. Let X and Y be two discrete random variables and E the set of all values that that these random variables can take. Let us assume that $E = \{(x_i, y_j) \mid i = 1, \ldots, h, j = 1, \ldots, l\}$. Then we can write :

Table 4.1: Joint probability of X and Y

X \ Y	y_1	y_2	$\cdots$	y_l	$\Pr_X(x)$
x_1	$p_{X,Y}(x_1,y_1)$	$p_{X,Y}(x_1,y_2)$	$\cdots$	$p_{X,Y}(x_1,y_l)$	$p_X(x_1)$
x_2	$p_{X,Y}(x_2,y_1)$	$p_{X,Y}(x_2,y_2)$	$\cdots$	$p_{X,Y}(x_2,y_l)$	$p_X(x_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_h	$p_{X,Y}(x_h,y_1)$	$p_{X,Y}(x_h,y_2)$	$\cdots$	$p_{X,Y}(x_h,y_l)$	$p_X(x_h)$
$\Pr_y(y)$	$p_Y(y_1)$	$p_Y(y_2)$	$\cdots$	$p_Y(y_n)$	1

where

$$P_X(x_k) = p_{X,Y}(x_k,y_1) + p_{X,Y}(x_k,y_2) + \cdots + p_{X,Y}(x_k,y_l) = \sum_{j=1}^{l} p_{X,Y}(x_k,y_j), \quad \forall 1 \leq k \leq h$$

$$P_Y(y_m) = p_{X,Y}(x_1,y_m) + p_{X,Y}(x_2,y_m) + \cdots + p_{X,Y}(x_h,y_m) = \sum_{i=1}^{h} p_{X,Y}(x_i,y_m), \quad \forall 1 \leq m \leq l$$

and $\sum_{\substack{i=1,\ldots,h \\ j=1,\ldots,l}} p_{X,Y}(x_i,y_j) = 1$.

Expectation of a random variable

The expectation of a random variable X is a weighted (in proportion to probabilities) average of the possible values of X.

Definition 4.4.2. The **expected value** (or **expectation** or **mean**) of a random value X, with probability mass function $p_X(x)$, is

$$E[X] = \sum_x x p_X(x).$$

Example 4.11. *Consider 3 independent coin tosses, each with* $\Pr(H) = 3/4$. *Let* X *be the random variable defined as the number of heads obtained. Then the probability distribution is (why?):*

$$p_X(X = k) = \begin{cases} (1/4)^3 & k = 0 \\ 3 \cdot 3/4 \cdot (1/4)^2 & k = 1 \\ 3 \cdot (3/4)^2 \cdot (1/4) & k = 2 \\ (3/4)^3 & k = 3 \end{cases}$$

From this we can compute the expected value as:

$$E[X] = 0 \cdot (1/4)^3 + 1 \cdot (3 \cdot 3/4 \cdot (1/4)^2) + 2 \cdot (3 \cdot (3/4)^2 \cdot (1/4)) + 3 \cdot (3/4)^3 = 9/4.$$

Conditioning

As we have done before for conditional probability of events, we can talk about the conditional probability of random variables. All the results described in Section 2 can be translated using the new notation.

The conditional probability mass function of a random variable X, conditioned on a particular event E, with $\Pr(E) > 0$, is defined by

$$p_{X|E}(x) = \Pr(X = x|E) = \frac{\Pr(\{X = x\} \cap E)}{\Pr(E)}.$$

Note that the events $\{X = x\} \cap E$ are disjoint for different values of x, their union is E, and, therefore,

$$\Pr(E) = \sum_x \Pr(\{X = x\} \cap E).$$

Combining the above two formulas, we see that

$$\sum_x p_{X|E}(x) = 1.$$

Let X and Y be two random variables associated with the same experiment. If we know that the experimental value of Y is some particular y (with $p_Y(y) > 0$), this provides partial knowledge about the value of X. This knowledge is captured by the conditional probability $p_{X|Y}$ of X given Y , which is defined by specializing the definition of $p_{X|E}$ to events E of the form $\{Y = y\}$:

$$p_{X|Y}(x, y) = \Pr(X = x|Y = y).$$

Using the definition of conditional probability we have seen before, we get

$$p_{X|Y}(x, y) = \frac{\Pr(X = x, Y = y)}{\Pr(Y = y)} = \frac{p_{X,Y}(x, y)}{p_Y(y)}.$$

The conditional probability mass function can be used to compute the joint and the marginal probability. In particular:

$$p_{X,Y}(x,y) = p_Y(y)p_{X|Y}(x|y) \quad \text{and} \quad p_{X,Y}(x,y) = p_X(x)p_{Y|X}(y|x)$$

Note that this is equivalent to the use of the multiplication rule which we have seen before. Moreover,

$$p_X(x) = \sum_y p_{X,Y}(x,y) = \sum_y p_Y(y)p_{X|Y}(x|y).$$

4.5 Looking ahead: Shannon's theorem

We know from Chapter 1 that the year 1948 marks the birth of information theory. In that year, Claude E. Shannon published his epoch making paper [17] on the limits of reliable transmission of data over unreliable channels and methods on how to achieve these limits. Among other things, this paper formalized the concept of information, and established bounds for the maximum amount of information that can be transmitted over unreliable channels.

As we will see in the next chapters, a communication channel is usually defined as a triple consisting of an input alphabet, an output alphabet, and for each pair (i, o) of input and output elements a transition probability $\Pr(i, o)$. Semantically, the transition probability is the probability that the symbol o is received given that i was transmitted over the channel.

Given a communication channel, Shannon proved that there exists a number, called the *capacity* of the channel, such that reliable transmission is possible for rates arbitrarily close to the capacity, and reliable transmission is not possible for rates above capacity.

The notion of capacity is defined purely in terms of information theory. As such it does not guarantee the existence of transmission schemes that achieve the capacity. In the same paper Shannon introduced the concept of *codes* as ensembles of vectors that are to be transmitted. It is clear that if the channel is such that even one input element can be received in at least two possible ways (albeit with different probabilities), then reliable communication over that channel is not possible if only single elements are sent over the channel. This is the case even if multiple elements are sent that are not correlated (in a manner to be made precise). To achieve reliable communication, it is thus imperative to send input elements that are correlated. This leads to the concept of a code, defined as a (finite) set of vectors over the input alphabet. In the previous chapters we assumed that all the vectors have the same length, and call this length the block length of the code. Suppose now that we send a codeword, and receive a vector over the output alphabet. How do we infer the vector that we sent? If the channel allows for errors, then there is no general way of telling which codeword was sent with absolute certainty. However, we can find the most likely codeword that was sent, in the sense that the probability that this codeword was sent given the observed vector is maximized. To see that we really can find such a codeword, simply list all the codewords, and calculate the conditional probability for the individual codewords. Then find the vector or vectors that yield the maximum probability and return one of them. We know that this decoder is called the maximum likelihood decoder. It is not perfect: it takes a lot of time (especially when the code is large) and it may err; but it is the best we can do. Shannon proved the existence of codes of rates arbitrarily close to capacity for which the probability of error of the maximum likelihood decoder goes to zero as the block length of the code goes to infinity. (In fact, Shannon proved that the decoding error of the maximum likelihood decoder goes to zero exponentially fast with the block length, but we will not discuss it here.)

Codes that approach capacity are very good from a communication point of view, but Shannon's theorems are non-constructive and don't give a clue on how to find such codes. More

importantly, even if an oracle gave us sequences of codes that achieve capacity for a certain rate, it is not clear how to encode and decode them efficiently. Design of codes with efficient encoding and decoding algorithms which approach the capacity of the channel is the main goal of coding theory.

4.6 Problems

Exercise 11. A fair coin is tossed, and a fair die is thrown. Write down sample spaces for:

(a) the toss of the coin;

(b) the throw of the die;

(c) the combination of these experiments.

Let A be the event that a head is tossed, and B be the event that an odd number is thrown. Directly from the sample space, calculate $\Pr(A \cap B)$ and $\Pr(A \cup B)$.

Exercise 12. Suppose you roll two fair, six-sided dice, one of which is red and the other of which is green. Define the following random variables:

$X =$ the number shown on the red die

$$Y = \begin{cases} 0 \text{ if the two dice show the same number} \\ 1 \text{ if the numebr of the green die is bigger than the number on the green die} \\ 2 \text{ if the number on thered die is bigger then the number on the green die} \end{cases}$$

1. Write down a table showing the joint probability mass function for X and Y
2. Find the marginal probability mass function for Y, and compute its expected value.
3. Find the conditional probability mass function for X given $Y = 1$.

5 LDPC codes

In this chapter

- Low-density parity-check codes (LDPCC)

Standard references: For this part of the course we mainly follow:

`http://www.cs.toronto.edu/ mackay/itprnn/book.pdf` and [7], [4], [5].

5.1 Introduction

Low-density parity-check (LDPC) codes are linear error-correction codes, first proposed in the 1962 by Gallager. At that time, their incredible potential remained undiscovered due to the computational demands of simulation, and they remained largely neglected for over 35 years.

In the mean time the field of coding theory was dominated by highly structured algebraic block codes. Despite the enormous practical success of these codes, their performance fell well short of the theoretically achievable limits set down by Shannon in his seminal 1948 paper. By the late 1980s, despite decades of attempts, researchers were largely resigned to this seemingly insurmountable theory-practice gap.

The relative quiescence of the coding field was utterly transformed by the introduction of "turbo codes", proposed by Berrou, Glavieux and Thitimajshima in 1993, wherein all the key ingredients of successful error correction codes were replaced: turbo codes involve very little algebra, employ iterative, distributed algorithms, focus on average (rather than worst-case) performance, and rely on soft (or probabilistic) information extracted from the channel. Overnight, the gap to the Shannon limit was all but eliminated, using decoders with manageable complexity.

As researchers struggled through the 1990s to understand just why turbo codes worked as well as they did, two researchers, McKay and Neal, introduced a new class of block codes designed to posses many of the features of the new turbo codes. It was soon recognized that these block codes were in fact a rediscovery of the LDPC codes developed years earlier by Gallager. Indeed, the algorithm used to decode turbo codes was subsequently shown to be a special case of the decoding algorithm for LDPC codes presented by Gallager so many years before.

New generalizations of Gallager's LDPC codes by a number of researchers including Luby, Mitzenmacher, Shokrollahi, Spielman, Richardson and Urbanke, produced new irregular LDPC codes which easily outperform the best turbo codes, as well as offering certain practical advantages and an arguably cleaner setup for theoretical results.

Today, design techniques for LDPC codes exist which enable the construction of codes which approach the Shannon's capacity to within hundredths of a decibel. So rapid has progress been in this area that coding theory today is in many ways unrecognizable from its state just a decade ago. In addition to the strong theoretical interest in LDPC codes, such codes have already been adopted in satellite-based digital video broadcasting and long-haul optical communication standards, are highly likely to be adopted in the IEEE wireless local area network standard, and are under consideration for the long-term evolution of third generation mobile telephony.

Algorithmic issues

Soon after Shannon's paper researchers found that random codes are capacity achieving. Unfortunately, achieving capacity is only part of the story. If these codes are to be used for communications, one needs fast algorithms for encoding and decoding.

Note that random binary codes of rate R are just 2^{Rn} random vectors of length n over the input alphabet, so we need some description of these vectors to be able to embed information into them, or we need to write all of them down into a so-called codebook describing which sequence of Rn bits gets mapped to which codeword. For example if $n = 1000$ and $R = 0.5$, this would require 2^{500} vectors - too much to be handled.

In the previous lectures we have seen that if the input alphabet has the structure of a field, then one can do better, at least as far as encoding goes. For example if we consider linear codes, then a codebook can be described in a natural way by mapping a vector $(x_1, \ldots, x_k) \in \mathbb{F}^k$, where $\mathbb{F}$ denotes any finite field, to the vector obtained by taking linear combinations of the basis vectors given by the coefficients $x_1, \ldots, x_k$.

The class of linear codes is very rich. Shannon's arguments can be used (almost verbatim) to show that there are sequences of linear codes with rates arbitrarily close to capacity and for which the error probability of the maximum likelihood decoder approaches zero (exponentially fast) as the block length goes to infinity. Moreover, it can also be shown that random linear codes achieve capacity. Unlike their non-linear brethren, linear codes can be encoded in polynomial time, rather than exponential time. This is good news.

How about decoding? The decoding problem seems much tougher. The maximum likelihood problem on the BSC has been shown to be NP-hard for many classes of linear codes (e.g., general linear codes over $\mathbb{F}_q$ for any q). It is therefore unlikely to find polynomial time algorithms for maximum likelihood decoding of general linear codes. One way to get around this negative result is to try to repeat the success story for the encoding problem and to specialize to subclasses of general linear codes. However, we have not been able to find subclasses of linear codes for which maximum likelihood decoding is polynomial time and which achieve capacity.

Another possibility is to look at sub-optimal algorithms that are polynomial time by construction. This is the path we will follow in the next section.

5.2 Low-density parity-check (LDPC) codes

In this section we will show two different ways to represent LDPC codes.

5.2.1 Matrix representation

Although LDPC codes can be generalized to non-binary alphabets, for the sake of simplicity we only consider binary LDPC codes.

As their name suggests, LDPC codes are linear block codes with parity-check matrices that contain only a very small number of non-zero entries. It is the sparseness of H which guarantees both a decoding complexity which increases only linearly with the code length and a minimum distance which also increases linearly with the code length.

Aside from the requirement that H be sparse, an LDPC code itself is no different to any other linear code. Indeed existing linear codes can be successfully used with the LDPC iterative decoding algorithms if they can be represented by a sparse parity-check matrix. Generally, however, finding a sparse parity-check matrix for an existing code is not practical. Instead LDPC codes are designed by constructing a sparse parity-check matrix first and then determining a generator matrix for the code afterwards.

The biggest difference between LDPC codes and classical linear codes is how they are decoded. Classical linear codes are generally decoded with ML like decoding algorithms and so are usually short and designed algebraically to make this task less complex. LDPC codes however are decoded iteratively using a graphical representation of their parity-check matrix and so are designed with the properties of H as a focus.

We recall that given a binary $[n, k]$ linear code with parity-check matrix H, each row of H corresponds to a parity-check equation and each column of H corresponds to a bit in the codeword. H is an $(n-k) \times n$ matrix, so we $(n-k)$ parity-check constraints. Moreover an element $\mathbf{c} \in \{0,1\}^n$ is a codeword if and only if it satisfies the matrix equation

$$H\mathbf{c}^T = 0.$$

Definition 5.2.1. An LDPC code is called (w_c, w_r)**-regular** if each its parity-check matrix contains a fixed number, w_c, of 1's per column, and a fixed number, w_r, of 1's per row.

Example 5.1. *The code C with parity-check matrix*

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

is a $(2, 3)$-regular code.

A regular LDPC code will have

$$(n - k) \cdot w_c = n \cdot w_r,$$

ones in its parity-check matrix.

5.2.2 Tanner's graph representation

Tanner considered LDPC codes (and generalization) and showed how they may be represented effectively by a so-called bipartite graph, now call a *Tanner graph*.

A *bipartite graph* is a graph (nodes with connected with edges) whose nodes may be separated into two types, and edges may only connect two nodes of different types. The two types of nodes in a Tanner's graph of a code is are the *variable nodes* and the *check nodes*, which in the following we will call *v-nodes* and *c-nodes*, respectively.

The Tanner graph of a code is drawn accordingly the following rule:

- check-node j is connected to variable node i whenever element h_{ij} in the parity-check matrix H is 1.

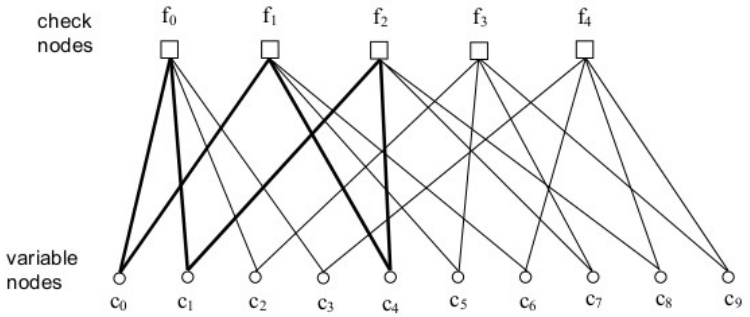
From this it is clear that there are $n - k$ check-nodes, one for each check equation, and n variable nodes, one for each code bit c_i.

Example 5.2. *Consider a $(10, 5)$ linear code C with $w_c = 2$ and $w_r = 4$, with the following parity-check matrix*

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Figure 5.1 shows the Tanner's graph corresponding to H. Note that the Tanner's graph of C

Figure 5.1: Tanner's graph for the code in the Example 5.2



is regular: each v-node has two edge connections and eac c-node has four edge connections; this is in accordance with the fact that $w_c = 2$ and $w_r = 4$.

5.3 LDPC constructions

The construction of binary LDPC codes involves assigning a small number of the values in an all-zero matrix to be 1 so that the rows and columns have the required degree distribution.

5.3.1 Gallager's construction

The original LDPC codes presented by Gallager are regular and defined by a banded structure in H. The rows of Gallager's parity-check matrices are divided into w_c sets with M/w_c rows in each set. The first set of rows contains w_r consecutive ones ordered from left to right across the columns. Every other set of rows is a randomly chosen column permutation of this first set. Consequently every column of H has a '1' entry once in every one of the w_c sets.

Example 5.3. *A length $n = 12$ $(3, 4)$-regular Gallager parity-check matrix is*

$$H = \left[\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Gallager codes were generalized by Tanner in 1981 [19], and were studied for applications to code-division multiple-access communication channel. Gallager codes were extended by MacKay and others ([14], [12].

5.3.2 MacKay codes

Another common construction for LDPC codes is a method proposed by MacKay and Neal. In this method columns of H are added one column at a time from left to right. The weight of each column is chosen to obtain the correct bit degree distribution and the location of the non-zero entries in each column chosen randomly from those rows which are not yet full. If at any point there are rows with more positions unfilled then there are columns remaining to be added, the row degree distributions for H will not be exact. The process can be started again or back tracked by a few columns, until the correct row degrees are obtained.

One drawback of such a design is that is lacks sufficient structure to enable low-complexity encoding.

5.4 LDPC decoding - Message-passing algorithm

The class of decoding algorithms used to decode LDPC codes are collectively termed *message-passing algorithms* since their operation can be explained by the passing of messages along the edges of a Tanner graph. Each Tanner graph node works in isolation, only having access to the information contained in the messages on the edges connected to it. The messag-passing algorithms are also known as iterative decoding algorithms as the messages pass back and forward between the bit and check nodes iteratively until a result is achieved (or the process halted). Different message-passing algorithms are named for the type of messages passed or for the type of operation performed at the nodes. In some algorithms, such as bit-flipping decoding, the

messages are binary and in others, such as belief propagation decoding, the messages are probabilities which represent a level of belief about the value of the codeword bits.

A detailed description of the message-passing algorithms is beyond the scope of this course.

6 Information Theory I

In this chapter:

- Entropy
- Conditional entropy

6.1 Entropy function

In this lesson we will see how to quantitatively express the information carried by a random variable.

Let X be a given random variable over a finite set, say $\{1, \ldots, m\}$. Let us denote

$$X(1) = x_1, \ldots, X(m) = x_m,$$

such that

$$\Pr(X = x_i) = p(x_i),\ i = 1, \ldots, m, \quad \text{and} \quad \sum_i p(x_i) = 1.$$

The random variable X can be seen as defining an m-ary source $\mathcal{S}$ consisting of independent observations $X_1, X_2, \ldots$, where $X_1, X_2, X_3, \ldots$ are independent random variables (Memoryless Source) with the same probability mass function as X. In particular this means

$$Pr(X_1 = x_{i_1}, X_2 = x_{i_2}, \ldots, X_k = x_{i_k}) = p(x_{i_1}) \cdots p(x_{i_k}).$$

We want to measure the amount of "information" in the source. In order to motivate the definitions we will give later, we start with some observations. First of all, notice that the smaller the probability $p(x_i)$, the more uncertain we should be that an observation of X will be x_i.
The main point here is that this "uncertainty" is strictly related to the information carried by x_i: suppose for example that $p(x_i) < p(x_j)$, then, when x_i occurs, we obtain more information about $\mathcal{S}$ than that we obtain in case x_j has occurred. More concretely, assume a binary source, for example a flip coin, then the question is "How much information do we receive when we are told that the outcome is H (head)? To answer this question we consider three different situations:

1. if $\Pr(H) = P(T) = 1/2$, the amount of information is 1 bit;

2. if $\Pr(H) = 1$, the amount of information is 0;

3. if $\Pr(H) = 0.7$, the amount of information is something between 0 and 1.

Thus it is clear that the information obtained from the outcome x_i is not function of itself, but rather of $p(x_i)$. So we could denote this information by $I(p(x_i))$, or more generally, $I(p)$. We can list some reasonable properties we would like to be satisfied by $I(p)$:

a) $I(p) \geq 0$

b) $I(1) = 0$, as the event that occurs with probability 1 does not carry any information.

c) $\lim_{p \to 0} I(p) = \infty$

d) Since the events of x_i and x_j occurring are independent, the information, obtained from the knowledge that both x_i and x_j have occurred, should be the sum $I(p(x_i)) + I(p(x_j))$. More formally,

$$I(\Pr(X = x_i) \cap \Pr(X = x_j)) = I(p(x_i) \cdot p(x_j)) = I(p(x_i)) + I(p(x_j)).$$

So we can regard $1/p(x_i)$ as a measure of the uncertainty of x_i. Note that if $p(x_i) = 1$, then $1/p(x_i) = 1$ as well, and if $p(x_i) = 0$ then $1/p(x_i) = \infty$ (and we are certain that the observation will not be x_i). Using a logaritmic scale, so that also property d) is satisfied, we can define the uncertainty of x_i to be

$$h(x_i) = I(p(x_i)) = \log_2 \frac{1}{p(x_i)} = -\log_2 p(x_i).$$

Thus uncertainty is measured in bits. Shannon proved that this is the only function that satisfies all the properties we have listed above.

Example 6.1. *Consider as before a single toss of coin. Then according to the definition above, we have that*

$$I(\Pr(H)) = \begin{cases} \log_2 \frac{1}{1} = 0 & if \quad \Pr(H) = 1 \\ 1 & if \quad \Pr(H) = 1/2 \\ \approx 0.51 & if \quad \Pr(H) = 0.7 \end{cases}$$

Definition 6.1.1. The **entropy** of a random variable X is defined to be the expected value,

$$H(X) = \sum_{i=1}^{m} p(x_i) \log_2 \frac{1}{p(x_i)} = -\sum_{i=1}^{m} p(x_i) \log_2 p(x_i),$$

of the uncertainty in a single observation of X.

Here we are using the fact that $\lim_{p \to 0} p \log_2(1/p) = 0$.

Example 6.2. *Let X_b be a binary random variable such that $\Pr(X_b = 0) = p$ and $\Pr(X_b = 1) = 1 - p$, where $0 \leq p \leq 1$, i.e.*

$$X_b = \begin{cases} 1 & \textit{with probability } \ p, \\ 0 & \textit{with probability } \ 1 - p. \end{cases}$$

The entropy of X_b is

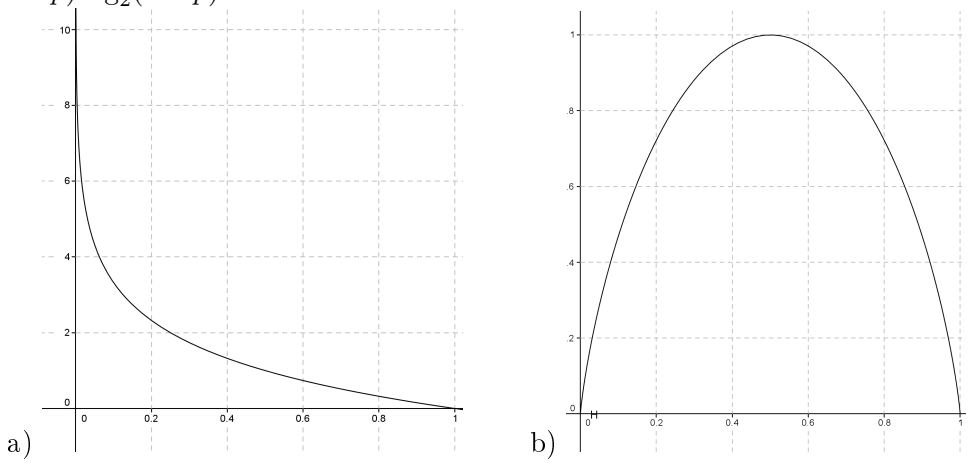
$$H(X_b) = -(p \log_2 p) - (1 - p) \log_2 (1 - p).$$

The function $H(X_b)$, often denoted as $H(p)$, is called **binary entropy function**, *and we have that:*

$$H(0) = H(1) = 0, \quad 0 \leq H(p) \leq 1, \quad \textit{and} \quad H(p) = 1 \Leftrightarrow p = 1/2.$$

Clearly the case $p = 1/2$ is the most uncertain, as the two outcomes are equally liked and there is no way to choose one over the other (See Figure 6.1). The figure illustrates some of the basic properties of entropy: it is a concave function of the distribution and equals 0 when $p = 0$ or $p = 1$. This is very intuitive because when $p = 0$ or 1, the variable is not random and there is no uncertainty.

Figure 6.1: a) $h(p) = \log_2 1/p$; b) the binary entropy function $H(p) = p\log_2 1/2 - (1-p)\log_2(1-p)$



Example 6.3 ([16] Example 3.1.2). *Consider two q-ary sources X and Y, such that*

$$\Pr(X = x_i) = 1/q \quad \forall i = 1, \ldots, q,$$

and

$$\Pr(Y = y_1) = 1, \quad \Pr(Y = y_i) = 0 \quad \forall i = 2, \ldots, q.$$

Then we have that

$$H(X) = \sum_{i=1}^{q} 1/q \log_2 q = \log_2 q \quad \text{and} \quad H(Y) = 0.$$

This example shows how the entropy of a source is a measure of the uncertainty of the source. Here, X, which emits all symbols with equal probability, is in a much state of uncertainty than Y, which emits always the same symbol. Thus the greater the entropy, the greater the uncertainty from each sample and the more information is obtained from the sample.

We list three properties of the entropy:

- It can be shown that $0 \leq H(X) \leq \log_2 m$.
- The *maximum entropy*, $H(X) = \log_2 m$, is reached when all the outcomes are equiprobable, i.e. $p(x_i) = 1/m$.
- The difference $\log_2 m - H$ is called *redundancy* of the source.

6.2 Conditional entropy

When we send information over a communication channel, we need to consider two random variables: X which gives the information to be sent over the channel, and Y which is the corruption of X that results from the noise in the channel. To deal with both of them we need to introduce two new concepts, those of *conditional entropy* and *mutual information*.

More precisely, let X and Y be two random variables over the same set $\{1, \ldots, m\}$. Note that the results we are going to present are still valid when the sets on which the random variables

are defined are different.
We can proceed as before and denote the values of Y by $Y(1) = y_1, \ldots, Y(m) = y_m$ with probabilities $p(y_1), \ldots, p(y_m)$. Let $p_{X,Y}(x, y)$ be the joint probability of X and Y, as defined in Lecture IV, Definition 3.1, and $p_{X|Y}(x, y)$ their conditional probability. Then we have:

Definition 6.2.1. The **joint entropy** of X and Y is

$$H(X, Y) = -\sum_{i=1}^{m}\sum_{j=1}^{m} p(x_i, y_j)\log_2(p(x_i, y_j)).$$

According to Lecture IV, we can define the probability of the conditional random variable $X|Y = y_k$ as

$$p(x_i|y_k) = \frac{p(x_i, y_k)}{p(y_k)}, \quad i = 1, \ldots, m,$$

where $p(x_i|y_k)$ means the probability that $X = x_i$ given that $Y = y_k$. Then the entropy of $X|Y = y_k$, that is the entropy of X conditioned to the specific event $Y = y_k$, is

$$H(X|Y = y_k) = -\sum_{i=1}^{m} p(x_i|y_k)\log_2(p(x_i|y_k)).$$

From this we can define the **conditional entropy** as the expected value

$$\begin{aligned} H(X|Y) &= \sum_{k=1}^{m} p(y_k)H(X|Y = y_k) \\ &= -\sum_{k=1}^{m} p(y_k)\sum_{i=1}^{m} p(x_i|y_k)\log_2(p(x_i|y_k)) \\ &= -\sum_{k=1}^{m}\sum_{i=1}^{m} p(x_i, y_k)\log_2(p(x_i|y_k)). \end{aligned}$$

It is straightforward to prove that

Theorem 6.2.2 (Chain rule for entropy). The joint entropy, conditional entropy and marginal entropy are related by:

$$H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y),$$

Proof We have that

$$\begin{aligned}
H(X,Y) &= -\sum_{i=1}^{m}\sum_{j=1}^{m} p(x_i,y_j)\log_2(p(x_i,y_j)) \\
&= -\sum_{i=1}^{m}\sum_{j=1}^{m} p(x_i,y_j)\log_2(\frac{p(x_i)}{p(y_j|x_i)}) \\
&= -\sum_{i=1}^{m}\sum_{j=1}^{m} p(x_i,y_j)\log_2(p(x_i)) - \sum_{i=1}^{m}\sum_{j=1}^{m} p(x_i,y_j)\log_2(p(y_j|x_i)) \\
&= -\sum_{i} p(x_i)\log_2(p(x_i)) - \sum_{i=1}^{m}\sum_{j=1}^{m} p(x_i,y_j)\log_2(p(y_j|x_i)) \\
&= H(X) + H(Y|X).
\end{aligned}$$

In a similar way can be proved the other equality. ■

Corollary 6.2.3. Let X, Y, Z be three random variables, then

$$H(X,Y|Z) = H(X|Z) + H(Y|X,Z).$$

Example 6.4 (Exercise 8.6 MacKay Book). *Let us consider two random variables X, Y such that their joint probability are given by the following table (Table 6.1).*

We want to compute $H(X), H(Y), H(X,Y), H(X|Y), H(Y|X)$. To compute $H(X), H(Y)$, we

Table 6.1: Joint probability of X and Y

Y \ X	1	2	3	4	$\Pr_Y(y)$
1	1/8	1/16	1/32	1/32	1/4
2	1/16	1/8	1/32	1/32	1/4
3	1/16	1/16	1/16	1/16	1/4
4	1/4	0	0	0	1/4
$\Pr_X(x)$	1/2	1/4	1/8	1/8	1

can use the definition of entropy:

$$H(X) = 1/2\log_2 2 + 1/4\log_2 4 + 1/8\log_2 8 + 1/8\log_2 8 = 7/4 \text{ (bits)}$$

Similarly, we obtain $H(Y) = 2$ bits. Then

$$\begin{aligned}
H(X|Y) =& 1/4H(X|Y=1) + 1/4H(X|Y=2) + 1/4H(X|Y=3) + 1/4H(X|Y=4) \\
=& 1/4\cdot(7/4 + 7/4 + 2 + 0) = 11/8 \text{ bits}
\end{aligned}$$

$$H(X,Y) = H(Y) + H(X|Y) = 2 + 11/8 = 27/8 \text{ bits}$$

and

$$H(Y|X) = H(X,Y) - H(X) = 13/8 \text{ bits}.$$

Another very used concept is the following.

Definition 6.2.4. The **mutual information** $I(X;Y)$ of the random variables X and Y is defined by

$$I(X;Y) = \sum_{i=1}^{m}\sum_{j=1}^{m} p(x_i, y_j) \log_2 \frac{p(x_i, y_j)}{p(x_i)p(y_j)}.$$

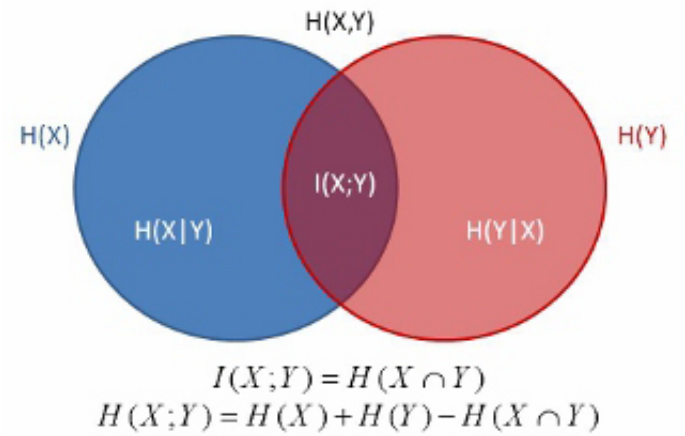
Roughly speaking, the mutual information measures in some sense the dependence between X and Y. It is symmetric in X and Y, and always non-negative. If X and Y are independent then $p(x, y) = p(x)p(y)$ and $I(X;Y) = 0$. On the contrary, we can see that this value increases as the dependence between X and Y increases. The mutual information is related to the entropy, as shown by the following result.

Theorem 6.2.5.

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

In words, this means that the mutual information $I(X;Y)$ equals the reduction in the uncertainty of X due to the knowledge of Y, and similarly, it equals the reduction in the uncertainty of Y due to the knowledge of X.
Intuitively it is possible to represent the relation between entropy and mutual information using a Venn diagram.

Figure 6.2: Mutual information



6.3 Channel capacity

We consider a *discrete communication channel*, i.e. a system in which the output depends probabilistically on the input. Let $\mathcal{X} = \{x_1, \ldots, x_m\}$ be the input alphabet and $\mathcal{Y} = \{y_1, \ldots, y_m\}$ be the output alphabet. We define the *channel statistics* as an $m \times m$ probability matrix P such that

$$P = [p_{i,j}],\ i, j = 1, \ldots, m, \quad p_{ij} = p(y_j|x_i),$$

i.e. the matrix with $p_{i,j}$ in the position (i, j).
Given a discrete memoryless channel (DMC, for short), we say that the channel is noisy provided

that $p(y_j|x_i) \neq 0$ for at least one $i \neq j$. Let X be a random variable which takes on values $x_1, \ldots, x_m$ with probability $p(x_i), i = 1, \ldots, m$. Then X and the DCM determine a random variable Y such that

$$p(y_j) = \sum_{i=1}^{m} p(x_i)p(y_j|x_i).$$

Note that $\sum_{i=1}^{m} p(y_j) = \sum_j \sum_i p(x_i)p(y_j|x_i) = \sum_i \left(\sum_j p(y_j|x_i)\right) p(x_i) = \sum_i p(x_i) = 1$.

Formally, we can define a **communication channel** as a system composed by

- a random variable X over an input alphabet $\mathcal{X} = \{x_1, \ldots, x_m\}$;
- a random variable Y over an output alphabet $\mathcal{Y} = \{y_1, \ldots, y_m\}$;
- $p(y_j|x_i)$, $i, j = 1, \ldots, m$, i.e. the probability that y_j is received whenever x_i is sent.

We would like to determine the maximum information that can be sent over a channel. At the beginning, when no symbol has been received, the uncertainty of X is its entropy $H(X)$. When a symbol is received, this uncertainty is reduced to $H(X|Y)$. Roughly we can say that the information across the channel has been

$$H(X) - H(X|Y) = I(X;Y).$$

This motivates the following definition.

Definition 6.3.1. The **capacity** of a DMC is

$$C = max_{p(x)} I(X;Y).$$

The capacity is the maximum rate at which we can send information over the channel and recover the information with a negligible error probability. C is only function of the probabilities defining the channel.

Example 6.5 (Noiseless Binary Channel). *Suppose that we have a channelwhose binary input are reproduced exactly at the output (Figure 6.3). In this case any transmitted bit is received without error. hence one error-free bit can be transmitted per use of the channel, and the capacity is* 1 *bit. We can also calculate the information capacity* $C = \max I(X;Y) = 1$ *bit, which is achieved by using the input distribution* $p(X = 0) = p(X = 1) = 1/2$.

Figure 6.3: Noiseless binary channel, $C = 1$ bit.

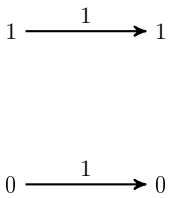
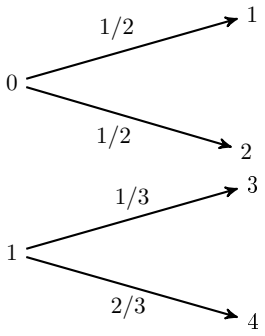


Figure 6.4: Noisy channel with nonoverlapping outputs, $C = 1$ bit.



Example 6.6 (Noisy Channel with Nonoverlapping Outputs). *This channel (see Figure 6.4) has two possible outcomes corresponding to each ot the two inputs. The channel appears to be noisy, but really is not. Even though the output of the channel is a random consequence of the input, the input can be determined form the output, and hence every transmitted bit can be recovered without error. The capacity of this channel is 1bit per transmission.*

Example 6.7 (BSC). *Consider a binary symmetric channel, with probability matrix*

$$P = \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}$$

We have seen that this means that the probability of correct transmission is

$$p(x_i|y_i) = p, \quad i = 1, 2,$$

and the probability of incorrect transmission of each symbol is

$$p(x_i|y_j) = 1 - p, \quad i, j = 1, 2, \quad i \neq j.$$

a. Let the input distribution be $\Pr(X = 0) = \alpha$ *and* $\Pr(X = 1) = 1 - \alpha$, $\alpha \in [0, 1]$. *Compute* $P(Y = 0), P(Y = 1), H(Y)$, *and* $H(Y|X)$.

b. Derive $I(X;Y)$ *in terms of* α.

c. Compute the capacity of this channel.

Solution:

a.

$$\Pr(Y = 0) = \sum_x \Pr(X = x)P(Y = 0|X = x) = \alpha \cdot p + (1-\alpha) \cdot (1-p) = 1 - p - \alpha \cdot (1-2p).$$

$$\Pr(Y = 1) = \sum_x \Pr(X = x)P(Y = 1|X = x) = (1-\alpha) \cdot p + \alpha \cdot (1-p) = p + \alpha \cdot (1-2p).$$

Therefore we have:

$$\begin{aligned} H(Y) &= -\sum_y \Pr(Y = y) \log_2(\Pr(Y = y)) = \\ &= -\big[(1 - p - \alpha \cdot (1-2p)) \cdot log_2(1 - p - \alpha \cdot (1-2p)) + \\ &\quad +(p + \alpha \cdot (1-2p)) \cdot log_2(p + \alpha \cdot (1-2p))\big] = \\ &= H(p + \alpha \cdot (1-2p)) \quad \textit{(the binary entropy function)} \end{aligned}$$

Next we compute the conditional entropy:

$$H(Y|X) = \sum_{i=1,2} p_X(x_i) H(Y|X = x_i) = -p \log_2 p - (1-p) \log_2 (1-p) = H(p).$$

b. We derive $I(X;Y)$ in terms of α: Thus we have:

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H(p + (1-p)\alpha) - H(p) \end{aligned}$$

c. We have that $C = \max_X I(X;Y)$. Note that $H(Y|X) = H(p)$ does not depend on α, therefore to compute the maximum of $I(X;Y)$, we only need to find the value of α which maximizes $H(p + (1-2p)\alpha)$. Further, we know that $H(x)$ takes its maximum value when $x = 1/2$ and that when $\alpha = 1/2$, the value $p + (1-2p)\alpha = 1/2$. Hence $I(X;Y)$ reaches its maximum value when $\alpha = 1/2$. Then

$$\begin{aligned} C &= max_\alpha H(p + \alpha \cdot (1-2p)) - H(p) \\ &= H(1/2) - H(p) = 1 - H(p) \end{aligned}$$

Properties of the channel capacity

1. $C \geq 0$ since $I(X;Y) \geq 0$
2. $C \leq \log |\mathcal{X}|$ since $C = \max I(X;Y) \leq \max H(X) = \log |\mathcal{X}|$
3. $C \leq \log |\mathcal{Y}|$ for the same reason
4. $I(X;Y)$ is a continuos function of $p(x)$

We conclude this section with one of the most important result in information theory, the Shannon's second theorem, which gives an operational meaning to the definition of the capacity as the number of bits we can transmit reliably over the channel. Recall that the transmission rate of an m-ary (n, M) code is

$$R = \frac{\log_m M}{n} \text{ (bit/transmission)} ,$$

and that it measures the efficiency of the transmission.

Theorem 6.3.2 (Channel coding theorem). , The maximum rate R of information over a channel with arbitrarily low error probability is given by its channel capacity C.

This theorem, also known as **Noisy coding theorem** or **Shannon II Theorem**, states that as long as $R \leq C$, then the error probability can be made arbitrarily small. In other words it asserts that *good code exists*, but unfortunately its proof is not constructive and we do not know how to construct it.

6.4 Problems

Exercise 13. Consider the language of 1-character strings over $\{A, B, C, D\}$ with associated probabilities $1/3, 1/12, 1/4$, and $1/3$. What is its corresponding entropy?

Exercise 14. Let X_1 and X_2 be identically distributed, but not necessarily independent. Let

$$\rho = 1 - \frac{H(X_2|X_1)}{H(X_1)}$$

1. Show that $\rho = \frac{I(X_1;X_2)}{H(X_1)}$
2. Show that $0 \leq \rho \leq 1$
3. When is $\rho = 0$?

7 Information Theory II - Source codes

In this chapter

- Source codes
- Source coding theorem
- Huffman encoding

In this lecture we consider the problem of compressing data. Generally speaking, data compression techniques, such as those used in common compression utilities, allow reducing file sizes by exploiting redundancies in the data contained in them. Note that the redundancy we are considering here is different from the redundancy that is added for correcting errors caused by the noise in the channel and which is very well structured. *Lossless coding* techniques permit to achieve compression without compromising any information stored in the file. Lossy techniques may achieve even greater compression, but only by providing an approximate reconstruction of the original data.

In particular we consider *variable-length source codes*, which encode one source symbol at a time, instead of encoding strings of symbols, as block codes. These codes are lossless: unlike block codes, they are guaranteed to compress and decompress without any errors. The high level idea here is that we can achieve compression, on average, by assigning shorter encodings to the more probable outcomes and longer encodings to the less probable.

Example 7.1. *Given three different symbols a, b, c, such that*

$$\Pr(a) = 0.5 \qquad \Pr(b) = 0.25 \qquad \Pr(c) = 0.25.$$

We could save space by encoding a using a single bit:

$$\begin{aligned} a &\longrightarrow 0 \\ b &\longrightarrow 01 \\ c &\longrightarrow 10 \end{aligned}$$

In this way the expected number of bits for codeword will be

$$1 \cdot \Pr(a) + 2 \cdot \Pr(b) + 2 \cdot \Pr(c) = 1.5,$$

so we save 0.5 *bits per codeword.*

Notice that we have a problem with the encoding technique used in the previous example. More precisely some bit sequences are ambiguous, that is, they will match more than one possible symbol sequence. For example, does the sequence 010 correspond to ac or ba? One possible way of preventing this problem is to require that the binary code of each symbol cannot be a prefix of any other. Any encoding that satisfies this property is known as a *prefix code*.

In what follows, we formalize all these concepts, using source codes, Shannon's lower bound on the code length, and the Huffman coding algorithm.

7.1 Source codes

Let $\mathcal{A}$ be a set of symbols, we denote by $\mathcal{A}^N$ the set of all strings of length N composed of elements from $\mathcal{A}$, and $\mathcal{A}^+$ the set of all strings of finite length of elements of $\mathcal{A}$.

Definition 7.1.1. Let X be an m-ary source with input alphabet $\mathcal{A} = \{a_1, \ldots, a_m\}$ and probabilities $p(a_1), \ldots, p(a_m)$, we define a **binary source code** C for X as an encoding map:

$$\begin{aligned} \mathcal{A} &\longrightarrow \{0,1\}^+ \\ a &\longmapsto c(a). \end{aligned}$$

The **extended code** C^+ is a mapping:

$$\begin{aligned} \mathcal{A}^N &\longrightarrow \{0,1\}^+ \\ \mathbf{a} = (a_1 x_2 \dots a_N) &\longmapsto c(a_1)c(a_2)\dots c(a_N) = c^+(\mathbf{a}), \end{aligned}$$

obtained from C by concatenation.
Let $c(\mathbf{a})$ be the codeword corresponding to $\mathbf{a} \in \mathcal{A}^N$ in C, then we will denote its length by $l(\mathbf{a})$ (instead of $l(c(\mathbf{a}))$), and $l_i = l(a_i)$, for $a_i \in \mathcal{A}$.

The three properties that a source code has to achieve are:

1. any encoded string must have a unique decoding;
2. the decoding procedure has to be easy/efficient;
3. the code should achieve as much compression as possible.

Example 7.2. *Let X be a source consisting of three different symbols, a, b, c. And consider the following encoding:*

$$a \mapsto 1 \qquad b \mapsto 00 \qquad c \mapsto 11.$$

Clearly this code does not satisfy the first property above, as both the strings "aaaa" and "cc" are encoded as 1111*.*

To formalize this argument, we introduce the following definitions.

Definition 7.1.2. A source code is said to be **nonsingular** if every element of $\mathcal{A}$ is mapped into a different string of $\{0,1\}^+$, i.e.

$$\text{if } a_i \neq a_j \text{ then } c(a_i) \neq c(a_j).$$

Since we are interested in sending sequences of symbols, we need to generalize the previous definition.

Definition 7.1.3. A code C is said to be **uniquely decodable** if its extension C^+ is nonsingular, i.e. $\forall \mathbf{x}, \mathbf{y} \in \mathcal{A}^+$, such that $\mathbf{x} \neq \mathbf{y}$, we have $c^+(\mathbf{x}) \neq c^+(\mathbf{y})$.

Example 7.3. *Keeping the same source of Example 7.2, consider the following encoding:*

$$a \mapsto 1 \qquad b \mapsto 00 \qquad c \mapsto 10.$$

Then it is possible to show that this code is uniquely decodable.

Example 7.4. *The code* $C_1 = \{0, 01, 001\}$ *is not uniquely decodable since* 001 *represents either the single codeword* 001 *or the concatenation of* 0 *and* 01. *It can be proved that the code* $C_2 = \{0, 10, 110\}$ *is uniquely decodable. For example the string* 0110010 *correspond to the string* $\mathbf{c}_1\mathbf{c}_3\mathbf{c}_1\mathbf{c}_2$, *where we set* $\mathbf{c}_1 = 0, \mathbf{c}_2 = 10, \mathbf{c}_3 = 110$.

Roughly speaking, if a code C is uniquely decodable, then it cannot have many short codewords. For example if 0110 is a codeword then 01 and 10 cannot be codewords. In general it is true that being uniquely decodable strictly depends on the codeword lengths.

Theorem 7.1.4 (McMillan Theorem). For each uniquely decodable binary code $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$, the codeword lengths must satisfy

$$\sum_{i=1}^{m} 2^{-l_i} \leq 1. \tag{7.1}$$

The inequality (7.1) is usually called **Kraft inequality**.

In order to have an easy decoding procedure, we need to easily identify the end of each codeword we receive. This motivates the following:

Definition 7.1.5. A code C is called **instantaneous** if, for each transmitted codeword c, c can be interpreted as a codeword as soon it is received.

Example 7.5. *The code* $C = \{1, 10, 000, 100\}$ *is uniquely decodable, but not instantaneous. When* 1 *is received we do not know if it is a codeword or the first part of* 10 *or* 100.

Note that an instantaneous code is also uniquely decodable, but not the other way around.

Definition 7.1.6. A code C is called **prefix code** if no codeword is a prefix of any other codeword.

Example 7.6. *The code* C_2 *in Example 7.4 is a prefix code.*

Example 7.7. *Consider the source consisting of the three symbols* a, b *and* c, *and the following encoding of this source:*

$$a \mapsto 0 \qquad b \mapsto 10 \qquad c \mapsto 11.$$

This is a prefix code.

Proposition 7.1.7. A code is instantaneous if and only if is a prefix code.

Example 7.8. *A very simple example of prefix code is the* **comma code**. *An example of comma code is*

$$\{0, 10, 110, 1110\}.$$

The name comes from the fact that "0" is actually used as a comma: it tells the receiver when the codeword ends.

Exercise: Consider a source X over $\{1, 2, 3, 4\}$, such that

$$\Pr(X = 1) = 0.5 \qquad \Pr(X = 2) = 0.25 \qquad \Pr(X = 3) = 0.125 \qquad \Pr(X = 4) = 0.125,$$

with the following encoding:

$$1 \mapsto 0 \qquad 2 \mapsto 10 \qquad 3 \mapsto 110 \qquad 111.$$

1. Is this code uniquely decodable? Why?
2. Encode the message 1422324
3. Decode the string 100111010110

It is very common to represent binary prefix codes using *binary trees*. We recall that a binary tree consists of a root with some branches, nodes, and leaves in such a way that the root and each node has exactly two "children" (See Figure 7.1).

In general, any binary code can be represented as a binary tree and each codeword is a path in the tree.

Definition 7.1.8. A binary code C is said to be **complete** if there is no unused leaf in the corresponding binary tree.

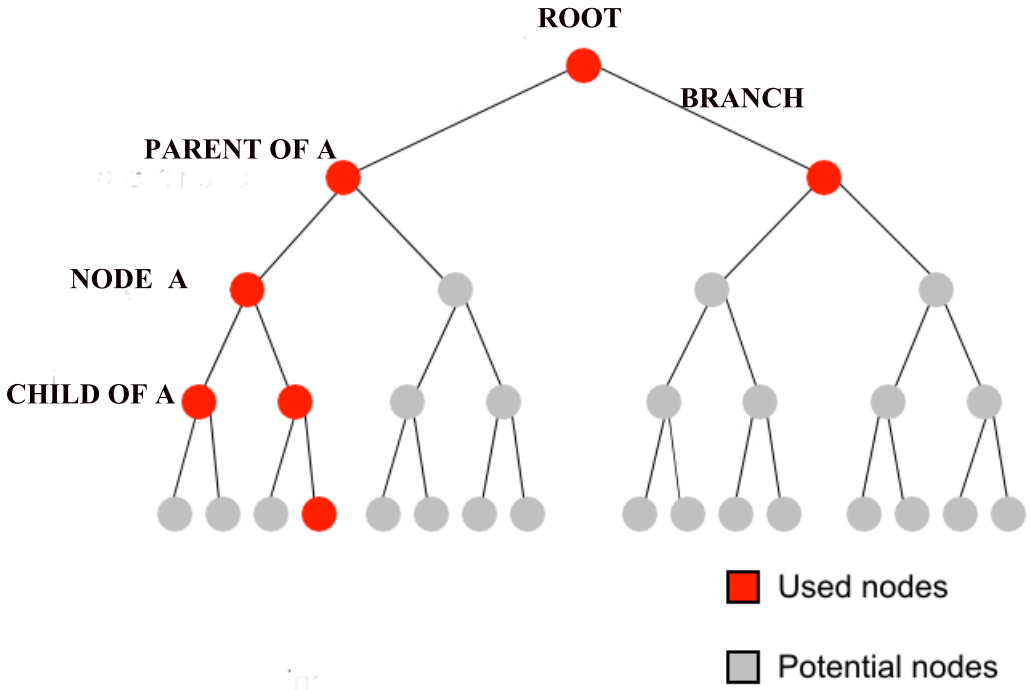
Lemma 7.1.9. A binary code C is a prefix code if and only if, in its corresponding tree, each codeword is a leaf.

We conclude this section with the following theorem, that contains the converse part of the McMillan Theorem.

Theorem 7.1.10 (Kraft-McMillan). Given a set of codeword lengths $\{l_1, \ldots, l_m\}$, there exits a binary prefix code with these codeword lengths if and only if $l_i, i = 1, \ldots, m$, satisfy the Kraft inequality

$$\sum_{i=1}^{m} 2^{-l_i} \leq 1.$$

Figure 7.1: Binary tree



This theorem proves that any codewords set that satisfies the prefix condition has to satisfies the Kraft inequality and, viceversa, that Kraft inequality is a sufficient condition for the existence of a codewords set with the specified set of codewords lengths. In other words this theorem ensures that instantaneous codes exist. Notice however that this theorem does not answer the question if a given code is indeed a prefix code.

Moreover we can always complete the codes that satisfy the Kraft inequality, to make these codes **complete**, i.e. satisfying $\sum_{i=1}^{m} 2^{-l_i} = 1$. Graphically this can be represented with binary trees without unused branches.

Example 7.9. *Does a binary prefix code with codeword lengths* $l_1 = 2, l_2 = 2, l_3 = 2, l_4 = 3, l_5 = 4$ *exist? The answer is yes, since* $\sum_{i=1}^{5} 2^{-l_i} = 15/16$.

7.2 How much can we compress?

We come back now to our original question: how can we decide the codeword lengths in order to achieve the best possible compression? And also, what is the best achievable compression?

As we have noticed before, it is important to consider the probabilities associated to the source symbols in order to construct the best possible code, i.e. the one with the smallest possible expected length. Recall that the **expected length** of a code C is given by:

$$L(C, X) = \sum_{i=1}^{m} p_i \cdot l_i.$$

The first result we are going to present is rather intuitive. Suppose to encode a source X, described as usual as a random variable. Then it seems reasonable that to encode the source symbols we need at least as many bits of information as there are in the source. As the source information is given by its entropy $H(X)$, we see that we need at least $H(X)$ bits per symbol.

Proposition 7.2.1 (Lower bound on $L(X, C)$). The expected length of a uniquely decodable code is bounded by the entropy of the source:

$$H(X) \leq L(C, X).$$

Example 7.10. *Consider the source X with the following probabilities:*

$$\Pr(X = 1) = 1/2 \qquad \Pr(X = 2) = 1/4 \qquad \Pr(X = 3) = 1/8 \qquad \Pr(X = 4) = 1/8$$

and the encoding map C given by

$$1 \longrightarrow 0 \qquad 2 \longrightarrow 10 \qquad 3 \longrightarrow 110 \qquad 4 \longrightarrow 111.$$

It is easy to see that C is uniquely decodable. The entropy of X is 1.75 bits, and the expected length is also 1.75 bits.

Corollary 7.2.2 (Optimal source codelength). The expected length is equal to $H(X)$ if and only if $l_i = \log_2 \frac{1}{p_i}$.

Theorem 7.2.3 (Source coding theorem). Given a random variable X, then there exists a prefix code C with expected length $L(X, C)$, satisfying

$$H(X) \leq L(X, C) \leq H(X) + 1.$$

The previous theorem is also known as **Noiseless coding theorem** or **Shannon I theorem**.

Example 7.11. *Consider the source X with the following probabilities:*

$$\Pr(X = 1) = 0.5 \qquad \Pr(X = 2) = 0.25 \qquad \Pr(X = 3) = 0.125 \qquad \Pr(X = 4) = 0.125$$

and the encoding map C given by

$$1 \longrightarrow 0 \qquad 2 \longrightarrow 10 \qquad 3 \longrightarrow 110 \qquad 4 \longrightarrow 111.$$

What is the expected code length? Is this encoding optimal from the expected code length point of view?

We can compute the expected length:

$$L(X, C) = 1.75,$$

and compare this value with the source entropy, that is $H(X) = 1.75$. This means that from the point of view of expected code length, the proposed code is optimal: there cannot exists another prefix code with a strictly shorter expected code length.

7.3 Huffman encoding

The optimal (shortest expected length) prefix code for a given distribution can be constructed by a simple algorithm due to Huffman.

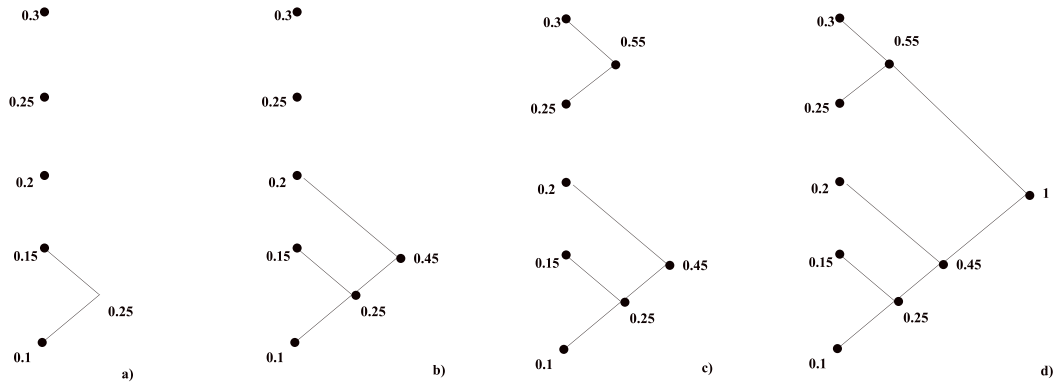
We can easily define the Huffman code $C : X \to \{0,1\}^+$ as the coding scheme that builds a binary tree from leaves up: it takes the two symbols having the least probabilities, assigns them equal lengths, merges them, and then reiterates the entire process.

Example 7.12. *Consider for example the following source: We can start the construction of*

symbols	a	b	c	d	e
p_i	0.3	0.25	0.2	0.15	0.1

the tree merging the two least probable values d and e in a new symbol de with probability $0.1 + 0.15 = 0.25$*. At the next iteration, we merge the two least probable symbols de and c, and so on. The resulting Huffman tree is given in Figure 7.2 d). Now given the tree, we can label*

Figure 7.2: Huffman tree example



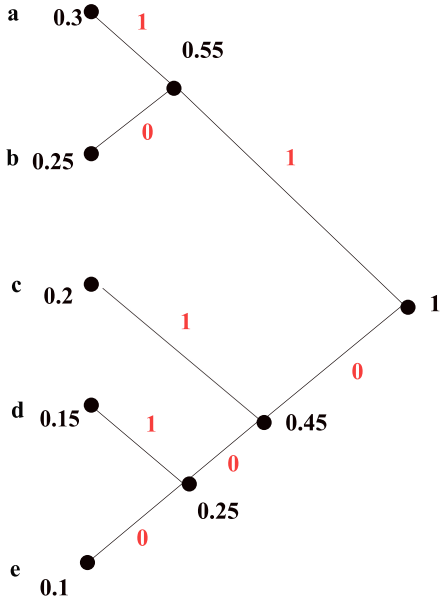
branches with $1s$ *and* $0s$ *to the get the code. In this way the Huffman encoding for a symbol* $x \in \{a, b, c, d, e\}$ *can now be read from the root of the tree and following down the tree until x is reached. The resulting code is*

$$\begin{aligned} a &\longrightarrow 11 \\ b &\longrightarrow 10 \\ c &\longrightarrow 01 \\ d &\longrightarrow 001 \\ e &\longrightarrow 000 \end{aligned}$$

It is easy to see that the expected code length is close to the information content: $H(X) \approx 2.23$ *bits and* $L(X, C) = 2,25$ *bits.*

Note that if there are more symbols with the same probability, then the corresponding Huffman code is not unique, but it depends on the order in which the symbols are merged. However all the codes that we can obtain from a given alphabet are optimal, i.e. they all have the same expected codelength. In particular:

Figure 7.3: Huffman encoding in Example 7.12



Proposition 7.3.1. Huffman codes are optimal prefix codes.

7.4 Problems

Exercise 15. A fair coin is flipped until the first head occurs. Let X denote the number of flips required.

a) Find the entropy $H(X)$ in bits. The following expressions may be useful:

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r} \qquad \sum_{n=0}^{\infty} nr^n = \frac{1}{(1-r)^2}$$

b) A random variable X is drawn according to this distribution. Find and "efficient" sequence of *yes/no* questions of the form, " Is X contained in the set S?" Compare $H(X)$ to the expected number of questions required to determine X.

8 Perfect secrecy

In this chapter

- Perfect secrecy
- One-time pad

We formalise what we mean for a symmetric encryption scheme to be secure. We introduce the perfect secrecy introduced by Shannon in his work, and finally we define the one-time pad scheme, proving that it is actually perfectly secure.

8.1 Encryption schemes: information-theoretic security

In this lecture we consider a different scenario: suppose a sender (Alice) wants to send a secret message to a receiver (Bob), and she does not want a potential eavesdropper (Eve) to violate privacy of its contents. Here we assume no computational bound on Eve's resources (this is why the security is referred as *information theoretic*). In other words a system is information theoretic secure (or *unconditionally secure*) if cannot be broken even with infinite computing power.
Also, according to the Kerckoffs principles, the system we are going to model is composed by algorithms and key(s), and all the secrecy of the message depends totally on the secrecy of the key, while the algorithms are publicly known.

We recall that an *encryption scheme* is composed by three algorithms as follows:

- Gen: it outputs a key k in some finite set $\mathcal{K}$;
- $\mathsf{Enc}_k(m)$: given a key k and a message $m \in \mathcal{M}$, it outputs a ciphertext $c \in \mathcal{C}$;
- $\mathsf{Dec}_k(c)$: given a key k and a ciphertext $c \in \mathcal{C}$, it outputs a message $m \in \mathcal{M}$.

For every $k \in \mathcal{K}$ and $m \in \mathcal{M}$, it should be that

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m.$$

Notice that Gen can not be deterministic, otherwise Eve could just run this algorithm and learn the key.

At this point our question is: when an encryption scheme is secure? And how should we formalize this notion? Intuitively we could say that an encryption scheme is secure if a ciphertext c does not leak *any* information, i.e. if guessing the plaintext m given c is as hard as without c. In other words if knowing the ciphertext is the same that knowing nothing at all. In 1949 Shannon formalized this idea using the theory of probability.

First we discuss a bit about the notation we are going to use. As we have seen, we are using $\mathcal{M}, \mathcal{K}, \mathcal{C}$ for the message-space, the key-space, and the ciphertext-space, respectively. Moreover we use M, K, C to denote random variables taking values over these sets. So M is a random variable over $\mathcal{M}$, K over $\mathcal{K}$ and C over $\mathcal{C}$.

Here we can note that $|\mathcal{C}| > |\mathcal{M}|$ since Enc has to be an injective map.

We can also assume that M and K are independent.

Example 8.1. *Let us consider the following encryption scheme:*

- *Let* $\mathcal{M} = \{m_1, m_2, m_3\}$*, with*

$$\Pr(M = m_1) = 1/2, \quad \Pr(M = m_2) = 1/3, \quad \Pr(M = m_3) = 1/6;$$

- *Let* $\mathcal{K} = \{k_1, k_2, k_3\}$*, with* $\Pr(K = k_i) = 1/3,\ \forall i = 1, 2, 3$*;*

	m_1	m_2	m_3
k_1	1	2	3
k_2	2	3	4
k_3	3	4	1

- *Let* $\mathcal{C} = \{1, 2, 3, 4\}$, *with encryption table*

From Lecture IV, we have:

$$\Pr(M = m, K = k) = \Pr(K = k)\Pr(M = m|K = k) = \Pr(K = k)\Pr(M = m).$$

So, using the encryption table, we can compute

$$\begin{aligned}\Pr(C = 1) &= \Pr(M = m_1, K = k_1) + \Pr(M = m_3, K = k3) = \\ &= \Pr(K = k_1)\Pr(M = m_1) + \Pr(K = k_3)\Pr(M = m_3) = \\ &= 1/3 \cdot 1/2 + 1/6 \cdot 1/3 = 2/9\end{aligned}$$

and, similarly,

$$\begin{aligned}\Pr(C = 2) &= 5/18 \\ \Pr(C = 3) &= 1/3 \\ \Pr(C = 4) &= 1/6.\end{aligned}$$

Using Bayes' rule

$$\Pr(M = m|C = c) = \frac{\Pr(C = c|M = m)\Pr(M = m)}{\Pr(C = c)},$$

we can compute:

$$\Pr(M = m_1|C = 1) = \frac{\Pr(C = 1|M = m_1)\Pr(M = m_1)}{\Pr(C = 1)} = \frac{1/3 \cdot 1/2}{2/9} = 3/4,$$

and, similarly, for other values:

$$\begin{array}{lll}\Pr(M = m_1|C = 1) = 3/4 & \Pr(M = m_2|C = 1) = 0 & \Pr(M = m_3|C = 1) = 1/4 \\ \Pr(M = m_1|C = 2) = 3/5 & \Pr(M = m_2|C = 2) = 2/5 & \Pr(M = m_3|C = 2) = 0 \\ \Pr(M = m_1|C = 3) = 1/2 & \Pr(M = m_2|C = 3) = 1/3 & \Pr(M = m_3|C = 3) = 1/6 \\ \Pr(M = m_1|C = 4) = 0 & \Pr(M = m_2|C = 4) = 2/3 & \Pr(M = m_3|C = 4) = 1/4\end{array}$$

So, if Eve intercepts for example the ciphertext $C = 1$, *she knows that* $m \neq m_2$ *(why?), and also that it is more likely to be* m_1 *than* m_3 *(why?). This is exactly what we do not want for an encryption scheme!*

Definition 8.1.1 (Shannon secrecy). An encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message-space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is **Shannon-secret** if

$$\Pr(M = m|C = c) = \Pr(M = m),$$

where M and C are random variable taking values over $\mathcal{M}$ and $\mathcal{C}$.

In other terms we can say that an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is Shannon secure if the random variables M and C are independent, or also if the a *posteriori* probability that m has been sent given that c has been received, does not provide any further information, except what was already known from a *priori* probabilities that m has been sent. From Bayes'rule we have:

$$\Pr(M|C) = \frac{\Pr(C|M) \cdot \Pr(M)}{\Pr(C)},$$

then $\Pr(M|C) \cdot \Pr(C) = \Pr(C|M) \cdot \Pr(M)$, and, as $\Pr(M|C) = \Pr(M)$, it follows that

$$\Pr(C|M) = \Pr(C).$$

This definition, which is very intuitive, is often difficult to prove in practice. We consider now another one:

Definition 8.1.2 (Perfect secrecy). An encryption system $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message-space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is **perfectly secure** if

$$\Pr(M = m_1)|C) = \Pr(M = m_2|C), \quad \forall m_1, m_2 \in \mathcal{M},$$

i.e. if a ciphertext is an equally likely output for any two messages in $\mathcal{M}$.

Although the previous two definitions look different, it can be proved that they are equivalent.

Proposition 8.1.3. An encryption scheme is Shannon secure if and only if it is perfectly secure.

8.2 A perfectly secure scheme. One-time pad

We describe the so called *one-time pad* encryption scheme, or *Vernam cipher* after his creator. The idea is that a randomly generated key is used to completely randomize each symbol composing the message.

Definition 8.2.1 (One-time pad). Let $n \geq 1$ be an integer and $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^n$:

- Gen: it outputs a uniformly random $k \in \{0,1\}^n$
- $\mathsf{Enc}_k(m)$: given a message $m \in \{0,1\}^n$, it outputs $c = m \oplus k \in \{0,1\}^n$ (where $\oplus$ is the bitwise XOR)
- $\mathsf{Dec}_k(c)$: it computes $m = c \oplus k \in \{0,1\}^n$

Example 8.2. *If* $m = 100100$ *and* $k = 010001$*, then* $c = m \oplus k = 110101$.

Proposition 8.2.2. The one-time pad is perfectly secure.

Proof First note that each key $k \in \mathcal{K}$ has probability $\frac{1}{|\mathcal{K}|} = \frac{1}{2^n}$ to be randomly chosen. And that, given $c = m \oplus k$, $\Pr(K = k|M = m) = \Pr(K = k) = 1/2^n$, because K and M are independent. Then using Bayes' rule:

$$\Pr(M = m|C = c) = \frac{\Pr(M = m)\Pr(C = c|M = m)}{\Pr(C = c)} = \frac{\Pr(M = m) \cdot 1/2^n}{1/2^n} = \Pr(M = m).$$

■

Note that, despite its efficiency (it involves just XOR operations), the scheme presents a major drawback: it requires a very long key which can be used only once (from this the term one-time pad!). For example, let us suppose that $c_1 = m_1 \oplus k$ and $c_2 = m_2 \oplus k$; if Eve can intercept c_1 and c_2, then she can obtain a partial information about m_1 and m_2 by computing: $c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$.

In the one-time pad scheme keys have the same length as the messages. This is an implication of perfect secrecy, as we can see from the following theorem due to Shannon.

Theorem 8.2.3. For each perfectly secret encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, if $\mathcal{M}$ and $\mathcal{K}$ denote the message-space and the key-space, respectively, then

$$|\mathcal{K}| \geq |\mathcal{M}|$$

Proof Without loss of generality we assume that every ciphertext can occur, i.e. $\Pr(C = c) > 0$ for all $c \in C$, otherwise we can just modify our definition of $\mathcal{C}$. As the encryption scheme is perfectly secret, then for any message m and any ciphertext c we have

$$\Pr(C = c|P = m) = \Pr(C = c) > 0,$$

i.e. for each $c \in \mathcal{C}$ there must be one key k such that $\mathsf{Enc}_k(m) = c$. Hence $|\mathcal{K}| \geq |\mathcal{C}|$ as required.
■

9 Solutions

Chapter 1

Solution 1. $n = 8;\ k = 3;\ d = 2$

Solution 2. If C can detect any single error, then it cannot have any codewords that differ by one bit. Thus if there is the zero word, then it cannnot have any words that have weight 1. The best you can do is to have either words of even weight, or all the words of odd weight.

Solution 3. Consider the code C' with distance $d-s$, such that $s \leq d-1$. Then C' can correct t errors if $t \leq \frac{(d-s)-1}{2}$.

Solution 4. When 000000 has been transmitted, the receiver will not detect transmission errors if the error pattern is $001111, 110011$ or 111100. This happens with probability

$$p^4(1-p)^2 + p^4(1-p)^2 + p^4(1-p)^2.$$

One finds the same probability of undetected error when one of the other three words has been transmitted. So the overall probability of undetected error is equal to $3p^4(1-p)^2$.

Chapter 2

Solution 5. Let $\mathbf{x} = (x_1, \ldots, x_n) \in C$, we add a parity check, obtaining

$$\hat{\mathbf{x}} = \left(x_1, \ldots, x_n, \sum_{i=1}^{n} x_i\right)$$

Since the field is $\mathbb{F}_2$, this last term is just a parity check, as now the sum of the terms of $\hat{x}$ is

$$x_1 + \ldots x_n + \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} 2x_i = 0.$$

Now the code with a parity check is $\hat{C} = \{\hat{\mathbf{x}} | \mathbf{x} \in C\}$. Because this is a binary code, the subspace criterion requires only that we check addition (since scalar multiplication has only two choices, 0 and 1). Taking $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}} \in \hat{C}$, their sum is

$$\hat{\mathbf{x}} + \hat{\mathbf{y}} = \left(x_1 + y_1, \ldots, x_n + y_n, \sum_{i=1}^{n}(x_i + y_i)\right) = \widehat{\mathbf{x} + \mathbf{y}} \in \hat{C}.$$

So $\hat{C}$ is linear.

Solution 6. The solution is not unique. An example is $\{11100, 11111, 01001\}$.

Solution 7. If they all have even weight, then we are done. Otherwise, there exists a codeword with odd weight; call it $\mathbf{v}$, call the set of all odd codewords C_o and that of all even codewords C_e. Now, because C is linear, we have that

$$C_e + v = \{\mathbf{e} + \mathbf{v} | \mathbf{e} \in C_e\} \subseteq C.$$

However, we in a binary code, the sum of an even word and an odd word is odd, since the sum is simply the exclusive-or of the summands. Also, all the words in the above set are distinct, so

$$C_e + \mathbf{v} \subseteq C_o \quad \text{and} \quad |C_e| = |C_e + \mathbf{v}| \leq |C_o|.$$

Similarly,

$$C_o + \mathbf{v} \subseteq C_e \quad \text{and} \quad |C_o| = |C_o + \mathbf{v}| \leq |C_e|.$$

Therefore $|C_e| = |C_o|$. Since C is the disjoint union of these two sets, it follows that

$$|C_e| = |C_o| = \frac{1}{2}|C|.$$

Solution 8.

1. $d = 1$. $A_2(8, 1) = 2^8$. C contains all the binary strings of length 8. parity bit added.

2. $A_2(n, 4)$
   - $n = 4$. $A_2(4, 4) = 2$. $C = \{0000, 1111\}$.
   - $n = 5$. $A_2(5, 4) = 2$. Because C contains the word 00000, then it can contain only one word with four or five ones. C cannot contain any other word because of the given minimum distance $d = 4$.

3. $A_q(4, 3)$
   - $q = 2$. $A_2(4, 3) = 2$. Indeed, because $0000 \in C$, there is no word in C with less then three ones and there can be only one word with three or four ones in C. One of $A_2(4, 3)$-codes is code $C = \{0000, O111\}$.
   - $q = 3$. $A_3(4, 3) \leq q^{n-d+1} = 3^2$ according to Singleton's bound. And we can find a ternary $(4, 9, 3)$-code $C = \{0000, 0111, 0222, 1012, 1120, 1201, 2021, 2102, 2210\}$ that reaches the Singleton's bound.

Chapter 3

Solution 9. The answers are not unique; it is possible to reorder the columns (except for the first one) and to choose different coset leaders in the last two rows for C_3.

$$C_1 = 00\ 01\ 10\ 11,$$

Since every vector is a codeword.

$$C_2 = \begin{matrix} 000 & 101 & 011 & 110 \\ 100 & 001 & 111 & 010 \end{matrix}$$

$$C_3 = \begin{matrix} 00000 & 10110 & 01011 & 11101 \\ 10000 & 00110 & 11011 & 01101 \\ 01000 & 11110 & 00011 & 10101 \\ 00100 & 10010 & 01111 & 11001 \\ 00010 & 10100 & 01001 & 11111 \\ 00001 & 10111 & 01010 & 11100 \\ 11000 & 01110 & 10011 & 00101 \\ 01100 & 11010 & 00111 & 10001 \end{matrix}$$

Solution 10.

(a) Since 11111 is in column 4, the answer is 11101. (It does not depend on the choice of coset leaders since 11111 is in a coset with a unique leader.) 01011 is a codeword, and so is unchanged by the decoding map.

(b) If you sent 00000 and the errors occur in the first two places then 11000 is correctly decoded to 00000. However, if the errors occur in the third and fourth places then we could send 00000, receive 00110 and we would "wrongly" decode it to 10110.

Chapter 4

Solution 11.

(a) $\{Head, Tail\}$

(b) $\{1, 2, 3, 4, 5, 6\}$

(c) $\{(H, 1), (T, 1), \ldots (H, 6), (T, 6)\}$

Clearly $\Pr(A) = 1/2 = \Pr(B)$. We assume that the two events are independent, so

$$\Pr(A \cap B) = \Pr(A) \Pr(B) = 1/4.$$

Alternatively, we can examine the sample space above and deduce that three of the twelve equality likely events comprise $A \cap B$.
Also, $\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B) = 3/4$, where this probability can be deduced by noting from the sample space that nine of twelve equally likely events comprise $A \cup B$.

Solution 12.

1.

Y \ X	1	2	3	4	5	6
0	1/36	1/36	1/36	1/36	1/36	1/36
1	5/36	4/36	3/36	2/36	1/36	0
2	0	1/36	2/36	3/36	4/36	5/36

2.

$$E(Y) = 0 \cdot (6/36) + 1 \cdot (15/36) + 2 \cdot (15/36) = 45/36.$$

3.

y	$\Pr(Y=y)$
0	6/36
1	15/36
2	15/36

1	2	3	4	5	6
5/15	4/15	3/15	2/15	1/15	0

Chapter 6

Solution 13. The entropy of the language is

$$\begin{aligned} H(X) =& 1/3\log_2(3) + 1/12\log_2(12) + 1/4\log_2(4) + 1/3\log_2(3) \\ =& 2/3\log_2(3) + 1/12(2+\log_2(3)) + 1/2 = 2/3 + 3/4\log_2(3), \end{aligned}$$

which is approximately 1.855.

Solution 14. Note that X_1 and X_2 are identically distributed, so $H(X_1) = H(X_2)$

1. $$\rho = 1 - \frac{H(X_2|X_1)}{H(X_1)} = \frac{H(X_1) - H(X_2|X_1)}{H(X_1)} = \frac{H(X_2) - H(X_2|X_1)}{H(X_1)} = \frac{I(X_1;X_2)}{H(X_1)}$$

2. $I(X_1;X_2) = H(X_1) - H(X_1|X_2)$, but $H(X_1|X_2) \geq 0 \Rightarrow I(X_1;X_2) \leq H(X_1) \Rightarrow \rho \leq 1$, $I(X_1;X_2) \geq 0 \Rightarrow 0 \Rightarrow \rho \geq 0$. Hence $0 \leq \rho \leq 1$.

3. X_1, X_2 are independent so $H(X_1|X_2) = H(X_1)$, so $I(X_1;X_2) = 0$. Thus $\rho = 0$ when X_1 and X_2 are i.i.d.

Chapter 7

Solution 14.

a) The number X of tosses till the first head appears has the geometric distribution with parameter $p = 1/2$, where $\Pr(X = n) = pq^{n-1}, n \in \{1, \ldots, \}$. Hence the entropy of X is

$$H(X) = -\sum_{n=1}^{\infty} pq^{n-1}\log(pq^{n-1}) = H(p)/p \text{ bits.}$$

If $p = 1/2$, then $H(X) = 2$ bits.

b) Intuitively, it seems clear that the best questions are those that have equally likely chances of receiving a yes or a no answer. Consequently, one possible guess is that the most "efficient" series of questions is: "Is $X = 1$?" If not, "Is $X = 2$"? If not, " Is $X = 3$?" ... with a resulting expected number of questions equal to $\sum_{n=1}^{\infty} n(1/2^n) = 2$. This should reinforce the intuition that $H(X)$ is a measure of the uncertainty of X. Indeed in this case, the entropy is exactly the same as the average number of questions needed to define X, and in general $E(\text{number of questions}) \geq H(X)$. This problem has an interpretation as a source coding problem. Let $0 = NO, 1 = YES$, $X = Source$, and $Y = EncodedSource$. Then the set of questions in the above procedure can be written as a collection of (X, Y) pairs: $(1,1), (2,01), (3,001)$, etc. In fact, this intuitively derived code is the optimal (Huffman) code minimizing the expected number of questions.

Bibliography

[1] http://www.cs.cmu.edu/ venkatg/teaching/codingtheory/.

[2] https://docs.switzernet.com/1/people/emin-gabrielyan/060413-coupled-checksums/Shokrolla

[3] http://www.cs.toronto.edu/ mackay/itprnn/book.pdf.

[4] sigpromu.org/sarah/SJohnsonLDPCintro.pdf.

[5] www.telecom.tuc.gr/ alex/papers/ryan.pdf.

[6] D. Augot, E. Betti, and E. Orsini. An introduction to linear and cyclic codes. In M. Sala, S. Sakata, T. Mora, C. Traverso, and L. Perret, editors, *Gröbner Bases, Coding, and Cryptography*, pages 47–68. Springer Berlin Heidelberg, 2009.

[7] D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to Probability.* Lectures Notes MIT, 2000.

[8] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal 29*, pages 147–160, 1950.

[9] W. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes.* ITPro collection. Cambridge University Press, 2003.

[10] W. C. Huffman and R. A. Brualdi. *Handbook of Coding Theory.* Elsevier Science Inc., New York, NY, USA, 1998.

[11] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series).* Chapman & Hall/CRC, 2007.

[12] D. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Trans. Information Theory*, pages 399–431, 1999.

[13] D. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[14] D. MacKay and R. Neal. Good codes based on very sparse matrices. *Cryptography and Coding, 5th IMA Conf., Lecture Notes in Computer Science, LNCS*, pages 100–111, 1995.

[15] F. MacWilliams and N. Sloane. *The Theory of Error Correcting Codes.* North-Holland Mathematical Library. North-Holland, 1978.

[16] S. Roman. *Introduction to coding and information theory.* Undergraduate texts in mathematics. Springer, 1997.

[17] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal 27*, pages 379–423, 623–656, 1948.

[18] N. Smart. *Cryptography: An Introduction, 3rd Edition.* Mcgraw-Hill College, 2004.

[19] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Information Theory*, pages 533–547, 1981.

[20] J. van Lint. *Introduction to Coding Theory.* Graduate Texts in Mathematics, 86, (Third Edition). Springer-Verlag, 1999.