About Chapter 1

In the first chapter, you will need to be familiar with the binomial distribution. And to solve the exercises in the text – which I urge you to do – you will need to know *Stirling's approximation* for the factorial function, $x! \simeq x^x\, e^{-x}$, and be able to apply it to $\binom{N}{r} = \frac{N!}{(N-r)!\,r!}$. These topics are reviewed below.

Unfamiliar notation? See Appendix A, p.598.

The binomial distribution

Example 1.1. A bent coin has probability f of coming up heads. The coin is tossed N times. What is the probability distribution of the number of heads, r? What are the mean and variance of r?

Solution. The number of heads has a binomial distribution.

$$P(r \mid f, N) = \binom{N}{r} f^r (1-f)^{N-r}. \tag{1.1}$$

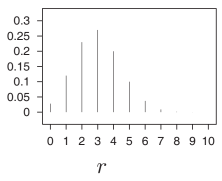


Figure 1.1. The binomial distribution $P(r \mid f = 0.3,\ N = 10)$.

The mean, $\mathcal{E}[r]$, and variance, $\mathrm{var}[r]$, of this distribution are defined by

$$\mathcal{E}[r] \equiv \sum_{r=0}^{N} P(r \mid f, N)\, r \tag{1.2}$$

$$\begin{aligned} \mathrm{var}[r] &\equiv \mathcal{E}\left[(r - \mathcal{E}[r])^2\right] & (1.3)\\ &= \mathcal{E}[r^2] - (\mathcal{E}[r])^2 = \sum_{r=0}^{N} P(r \mid f, N) r^2 - (\mathcal{E}[r])^2 . & (1.4)\end{aligned}$$

Rather than evaluating the sums over r in (1.2) and (1.4) directly, it is easiest to obtain the mean and variance by noting that r is the sum of N *independent* random variables, namely, the number of heads in the first toss (which is either zero or one), the number of heads in the second toss, and so forth. In general,

$$\begin{aligned} \mathcal{E}[x+y] &= \mathcal{E}[x] + \mathcal{E}[y] && \text{for any random variables } x \text{ and } y;\\ \mathrm{var}[x+y] &= \mathrm{var}[x] + \mathrm{var}[y] && \text{if } x \text{ and } y \text{ are independent.}\end{aligned} \tag{1.5}$$

So the mean of r is the sum of the means of those random variables, and the variance of r is the sum of their variances. The mean number of heads in a single toss is $f \times 1 + (1-f) \times 0 = f$, and the variance of the number of heads in a single toss is

$$\left[f \times 1^2 + (1-f) \times 0^2\right] - f^2 = f - f^2 = f(1-f), \tag{1.6}$$

so the mean and variance of r are:

$$\mathcal{E}[r] = Nf \qquad \text{and} \qquad \mathrm{var}[r] = Nf(1-f). \qquad \square \tag{1.7}$$

Approximating $x!$ and $\binom{N}{r}$

Let's derive Stirling's approximation by an unconventional route. We start from the Poisson distribution with mean λ,

$$P(r \mid \lambda) = e^{-\lambda}\frac{\lambda^r}{r!} \qquad r \in \{0, 1, 2, \ldots\}. \tag{1.8}$$

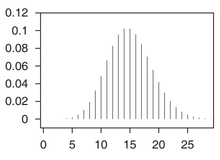


Figure 1.2. The Poisson distribution $P(r \mid \lambda = 15)$.

For large λ, this distribution is well approximated – at least in the vicinity of $r \simeq \lambda$ – by a Gaussian distribution with mean λ and variance λ:

$$e^{-\lambda}\frac{\lambda^r}{r!} \simeq \frac{1}{\sqrt{2\pi\lambda}}\, e^{-\frac{(r-\lambda)^2}{2\lambda}}. \tag{1.9}$$

Let's plug $r = \lambda$ into this formula, then rearrange it.

$$e^{-\lambda}\frac{\lambda^\lambda}{\lambda!} \simeq \frac{1}{\sqrt{2\pi\lambda}} \tag{1.10}$$

$$\Rightarrow \quad \lambda! \simeq \lambda^\lambda\, e^{-\lambda}\sqrt{2\pi\lambda}. \tag{1.11}$$

This is Stirling's approximation for the factorial function.

$$x! \simeq x^x\, e^{-x}\sqrt{2\pi x} \quad \Leftrightarrow \quad \ln x! \simeq x\ln x - x + \tfrac{1}{2}\ln 2\pi x. \tag{1.12}$$

We have derived not only the leading order behaviour, $x! \simeq x^x\, e^{-x}$, but also, at no cost, the next-order correction term $\sqrt{2\pi x}$. We now apply Stirling's approximation to $\ln \binom{N}{r}$:

$$\ln\binom{N}{r} \equiv \ln\frac{N!}{(N-r)!\, r!} \simeq (N-r)\ln\frac{N}{N-r} + r\ln\frac{N}{r}. \tag{1.13}$$

Since all the terms in this equation are logarithms, this result can be rewritten in any base. We will denote natural logarithms ($\log_e$) by 'ln', and logarithms to base 2 ($\log_2$) by 'log'.

Recall that $\log_2 x = \dfrac{\log_e x}{\log_e 2}$.

Note that $\dfrac{\partial \log_2 x}{\partial x} = \dfrac{1}{\log_e 2}\dfrac{1}{x}$.

If we introduce the *binary entropy function*,

$$H_2(x) \equiv x\log\frac{1}{x} + (1-x)\log\frac{1}{(1-x)}, \tag{1.14}$$

then we can rewrite the approximation (1.13) as

$$\log\binom{N}{r} \simeq N H_2(r/N), \tag{1.15}$$

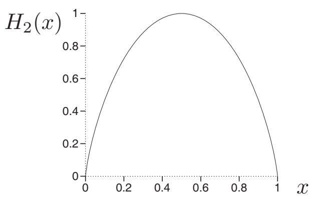


Figure 1.3. The binary entropy function.

or, equivalently,

$$\binom{N}{r} \simeq 2^{N H_2(r/N)}. \tag{1.16}$$

If we need a more accurate approximation, we can include terms of the next order from Stirling's approximation (1.12):

$$\log\binom{N}{r} \simeq N H_2(r/N) - \tfrac{1}{2}\log\left[2\pi N\,\frac{N-r}{N}\,\frac{r}{N}\right]. \tag{1.17}$$

1

Introduction to Information Theory

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

(Claude Shannon, 1948)

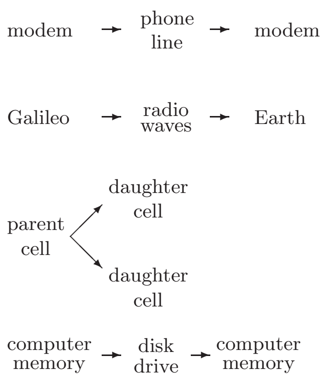
In the first half of this book we study how to measure information content; we learn how to compress data; and we learn how to communicate perfectly over imperfect communication channels.

We start by getting a feeling for this last problem.

▶ 1.1 How can we achieve perfect communication over an imperfect, noisy communication channel?

Some examples of noisy communication channels are:

- an analogue telephone line, over which two modems communicate digital information;
- the radio communication link from Galileo, the Jupiter-orbiting spacecraft, to earth;
- reproducing cells, in which the daughter cells' DNA contains information from the parent cells;
- a disk drive.

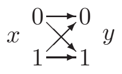


The last example shows that communication doesn't have to involve information going from one *place* to another. When we write a file on a disk drive, we'll read it off in the same location – but at a later *time*.

These channels are noisy. A telephone line suffers from cross-talk with other lines; the hardware in the line distorts and adds noise to the transmitted signal. The deep space network that listens to Galileo's puny transmitter receives background radiation from terrestrial and cosmic sources. DNA is subject to mutations and damage. A disk drive, which writes a binary digit (a one or zero, also known as a *bit*) by aligning a patch of magnetic material in one of two orientations, may later fail to read out the stored binary digit: the patch of material might spontaneously flip magnetization, or a glitch of background noise might cause the reading circuit to report the wrong value for the binary digit, or the writing head might not induce the magnetization in the first place because of interference from neighbouring bits.

In all these cases, if we transmit data, e.g., a string of bits, over the channel, there is some probability that the received message will not be identical to the transmitted message. We would prefer to have a communication channel for which this probability was zero – or so close to zero that for practical purposes it is indistinguishable from zero.

Let's consider a noisy disk drive that transmits each bit correctly with probability $(1-f)$ and incorrectly with probability f. This model communication channel is known as the *binary symmetric channel* (figure 1.4).



$$\begin{aligned} P(y=0\,|\,x=0) &= 1-f; & P(y=0\,|\,x=1) &= f; \\ P(y=1\,|\,x=0) &= f; & P(y=1\,|\,x=1) &= 1-f. \end{aligned}$$

Figure 1.4. The binary symmetric channel. The transmitted symbol is x and the received symbol y. The noise level, the probability that a bit is flipped, is f.

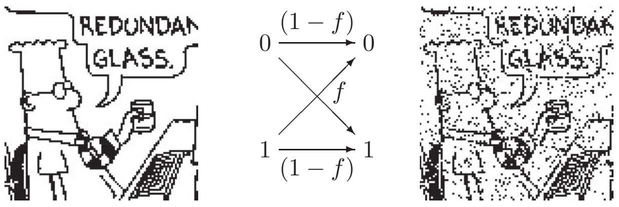


Figure 1.5. A binary data sequence of length 10 000 transmitted over a binary symmetric channel with noise level $f = 0.1$. [Dilbert image Copyright©1997 United Feature Syndicate, Inc., used with permission.]

As an example, let's imagine that $f = 0.1$, that is, ten per cent of the bits are flipped (figure 1.5). A useful disk drive would flip no bits at all in its entire lifetime. If we expect to read and write a gigabyte per day for ten years, we require a bit error probability of the order of 10^{-15}, or smaller. There are two approaches to this goal.

The physical solution

The physical solution is to improve the physical characteristics of the communication channel to reduce its error probability. We could improve our disk drive by

1. using more reliable components in its circuitry;
2. evacuating the air from the disk enclosure so as to eliminate the turbulence that perturbs the reading head from the track;
3. using a larger magnetic patch to represent each bit; or
4. using higher-power signals or cooling the circuitry in order to reduce thermal noise.

These physical modifications typically increase the cost of the communication channel.

The 'system' solution

Information theory and coding theory offer an alternative (and much more exciting) approach: we accept the given noisy channel as it is and add communication *systems* to it so that we can detect and correct the errors introduced by the channel. As shown in figure 1.6, we add an *encoder* before the channel and a *decoder* after it. The encoder encodes the source message $\mathbf{s}$ into a *transmitted* message $\mathbf{t}$, adding *redundancy* to the original message in some way. The channel adds noise to the transmitted message, yielding a received message $\mathbf{r}$. The decoder uses the known redundancy introduced by the encoding system to infer both the original signal $\mathbf{s}$ and the added noise.

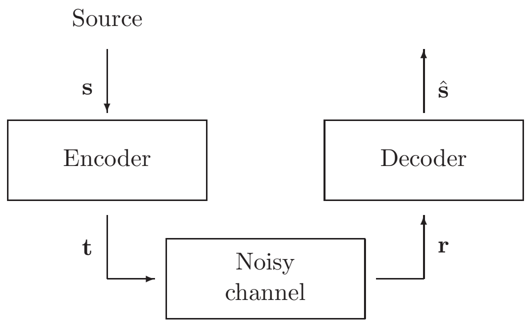


Figure 1.6. The 'system' solution for achieving reliable communication over a noisy channel. The encoding system introduces systematic redundancy into the transmitted vector **t**. The decoding system uses this known redundancy to deduce from the received vector **r** *both* the original source vector *and* the noise introduced by the channel.

Whereas physical solutions give incremental channel improvements only at an ever-increasing cost, system solutions can turn noisy channels into reliable communication channels with the only cost being a *computational* requirement at the encoder and decoder.

Information theory is concerned with the theoretical limitations and potentials of such systems. 'What is the best error-correcting performance we could achieve?'

Coding theory is concerned with the creation of practical encoding and decoding systems.

▶ 1.2 Error-correcting codes for the binary symmetric channel

We now consider examples of encoding and decoding systems. What is the simplest way to add useful redundancy to a transmission? [To make the rules of the game clear: we want to be able to detect *and* correct errors; and retransmission is not an option. We get only one chance to encode, transmit, and decode.]

Repetition codes

A straightforward idea is to repeat every bit of the message a prearranged number of times – for example, three times, as shown in table 1.7. We call this *repetition code* 'R_3'.

Source sequence **s**	Transmitted sequence **t**
0	000
1	111

Table 1.7. The repetition code R_3.

Imagine that we transmit the source message

$$\mathbf{s} = 0\ 0\ 1\ 0\ 1\ 1\ 0$$

over a binary symmetric channel with noise level $f = 0.1$ using this repetition code. We can describe the channel as 'adding' a sparse noise vector $\mathbf{n}$ to the transmitted vector – adding in modulo 2 arithmetic, i.e., the binary algebra in which 1+1=0. A possible noise vector $\mathbf{n}$ and received vector $\mathbf{r} = \mathbf{t} + \mathbf{n}$ are shown in figure 1.8.

$$\begin{array}{cccccccc}
\mathbf{s} & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
\mathbf{t} & \overbrace{0\,0\,0} & \overbrace{0\,0\,0} & \overbrace{1\,1\,1} & \overbrace{0\,0\,0} & \overbrace{1\,1\,1} & \overbrace{1\,1\,1} & \overbrace{0\,0\,0} \\
\mathbf{n} & 0\,0\,0 & 0\,0\,1 & 0\,0\,0 & 0\,0\,0 & 1\,0\,1 & 0\,0\,0 & 0\,0\,0 \\
\hline
\mathbf{r} & 0\,0\,0 & 0\,0\,1 & 1\,1\,1 & 0\,0\,0 & 0\,1\,0 & 1\,1\,1 & 0\,0\,0
\end{array}$$

Figure 1.8. An example transmission using R_3.

How should we decode this received vector? The optimal algorithm looks at the received bits three at a time and takes a majority vote (algorithm 1.9).

Received sequence $\mathbf{r}$	Likelihood ratio $\frac{P(\mathbf{r}\,|\,s=1)}{P(\mathbf{r}\,|\,s=0)}$	Decoded sequence $\hat{\mathbf{s}}$
000	γ^{-3}	0
001	γ^{-1}	0
010	γ^{-1}	0
100	γ^{-1}	0
101	γ^{1}	1
110	γ^{1}	1
011	γ^{1}	1
111	γ^{3}	1

Algorithm 1.9. Majority-vote decoding algorithm for R_3. Also shown are the likelihood ratios (1.23), assuming the channel is a binary symmetric channel; $\gamma \equiv (1-f)/f$.

At the risk of explaining the obvious, let's prove this result. The optimal decoding decision (optimal in the sense of having the smallest probability of being wrong) is to find which value of $\mathbf{s}$ is most probable, given $\mathbf{r}$. Consider the decoding of a single bit s, which was encoded as $\mathbf{t}(s)$ and gave rise to three received bits $\mathbf{r} = r_1r_2r_3$. By Bayes' theorem, the *posterior probability* of s is

$$P(s\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s)P(s)}{P(r_1r_2r_3)}. \tag{1.18}$$

We can spell out the posterior probability of the two alternatives thus:

$$P(s=1\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s=1)P(s=1)}{P(r_1r_2r_3)}; \tag{1.19}$$

$$P(s=0\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s=0)P(s=0)}{P(r_1r_2r_3)}. \tag{1.20}$$

This posterior probability is determined by two factors: the *prior probability* $P(s)$, and the data-dependent term $P(r_1r_2r_3\,|\,s)$, which is called the *likelihood* of s. The normalizing constant $P(r_1r_2r_3)$ needn't be computed when finding the optimal decoding decision, which is to guess $\hat{s}=0$ if $P(s=0\,|\,\mathbf{r}) > P(s=1\,|\,\mathbf{r})$, and $\hat{s}=1$ otherwise.

To find $P(s=0\,|\,\mathbf{r})$ and $P(s=1\,|\,\mathbf{r})$, we must make an assumption about the prior probabilities of the two hypotheses $s=0$ and $s=1$, and we must make an assumption about the probability of $\mathbf{r}$ given s. We assume that the prior probabilities are equal: $P(s=0) = P(s=1) = 0.5$; then maximizing the posterior probability $P(s\,|\,\mathbf{r})$ is equivalent to maximizing the likelihood $P(\mathbf{r}\,|\,s)$. And we assume that the channel is a binary symmetric channel with noise level $f < 0.5$, so that the likelihood is

$$P(\mathbf{r}\,|\,s) = P(\mathbf{r}\,|\,\mathbf{t}(s)) = \prod_{n=1}^{N} P(r_n\,|\,t_n(s)), \tag{1.21}$$

where $N = 3$ is the number of transmitted bits in the block we are considering, and

$$P(r_n\,|\,t_n) = \begin{cases} (1-f) & \text{if} \quad r_n = t_n \\ f & \text{if} \quad r_n \neq t_n. \end{cases} \tag{1.22}$$

Thus the likelihood ratio for the two hypotheses is

$$\frac{P(\mathbf{r}\,|\,s=1)}{P(\mathbf{r}\,|\,s=0)} = \prod_{n=1}^{N} \frac{P(r_n\,|\,t_n(1))}{P(r_n\,|\,t_n(0))}; \tag{1.23}$$

each factor $\frac{P(r_n|t_n(1))}{P(r_n|t_n(0))}$ equals $\frac{(1-f)}{f}$ if $r_n = 1$ and $\frac{f}{(1-f)}$ if $r_n = 0$. The ratio $\gamma \equiv \frac{(1-f)}{f}$ is greater than 1, since $f < 0.5$, so the winning hypothesis is the one with the most 'votes', each vote counting for a factor of γ in the likelihood ratio.

Thus the majority-vote decoder shown in algorithm 1.9 is the optimal decoder if we assume that the channel is a binary symmetric channel and that the two possible source messages `0` and `1` have equal prior probability.

We now apply the majority vote decoder to the received vector of figure 1.8. The first three received bits are all `0`, so we decode this triplet as a `0`. In the second triplet of figure 1.8, there are two `0`s and one `1`, so we decode this triplet as a `0` – which in this case corrects the error. Not all errors are corrected, however. If we are unlucky and two errors fall in a single block, as in the fifth triplet of figure 1.8, then the decoding rule gets the wrong answer, as shown in figure 1.10.

$$\begin{array}{rccccccc}
\mathbf{s} & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
\mathbf{t} & \overbrace{000} & \overbrace{000} & \overbrace{111} & \overbrace{000} & \overbrace{111} & \overbrace{111} & \overbrace{000} \\
\mathbf{n} & 000 & 001 & 000 & 000 & 101 & 000 & 000 \\ \hline
\mathbf{r} & \underbrace{000} & \underbrace{001} & \underbrace{111} & \underbrace{000} & \underbrace{010} & \underbrace{111} & \underbrace{000} \\
\hat{\mathbf{s}} & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
\text{corrected errors} & & \star & & & & & \\
\text{undetected errors} & & & & & \star & &
\end{array}$$

Figure 1.10. Decoding the received vector from figure 1.8.

Exercise 1.2.[2, p.16] Show that the error probability is reduced by the use of R_3 by computing the error probability of this code for a binary symmetric channel with noise level f.

The exercise's rating, e.g.'[2]', indicates its difficulty: '1' exercises are the easiest. Exercises that are accompanied by a marginal rat are especially recommended. If a solution or partial solution is provided, the page is indicated after the difficulty rating; for example, this exercise's solution is on page 16.

The error probability is dominated by the probability that two bits in a block of three are flipped, which scales as f^2. In the case of the binary symmetric channel with $f = 0.1$, the R_3 code has a probability of error, after decoding, of $p_b \simeq 0.03$ per bit. Figure 1.11 shows the result of transmitting a binary image over a binary symmetric channel using the repetition code.

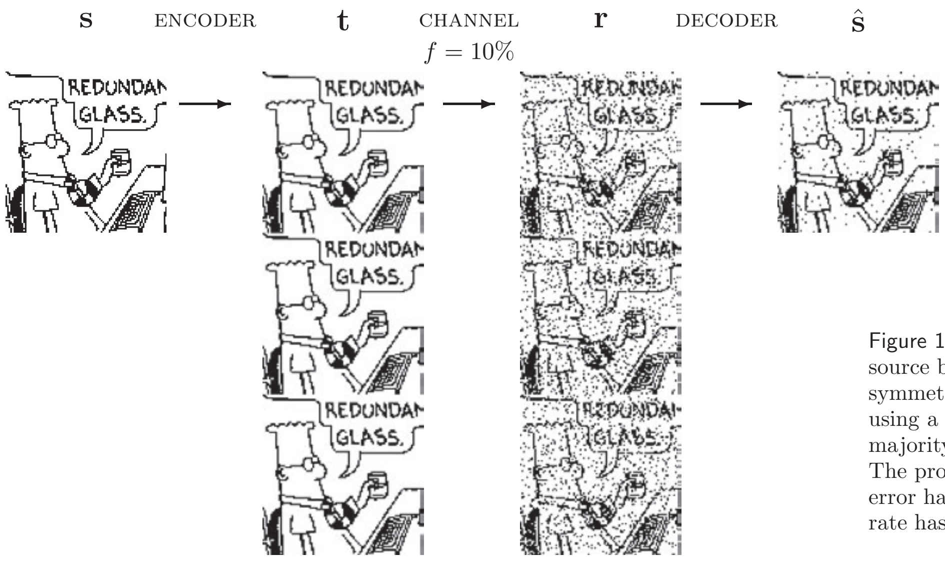


Figure 1.11. Transmitting 10 000 source bits over a binary symmetric channel with $f = 10\%$ using a repetition code and the majority vote decoding algorithm. The probability of decoded bit error has fallen to about 3%; the rate has fallen to 1/3.

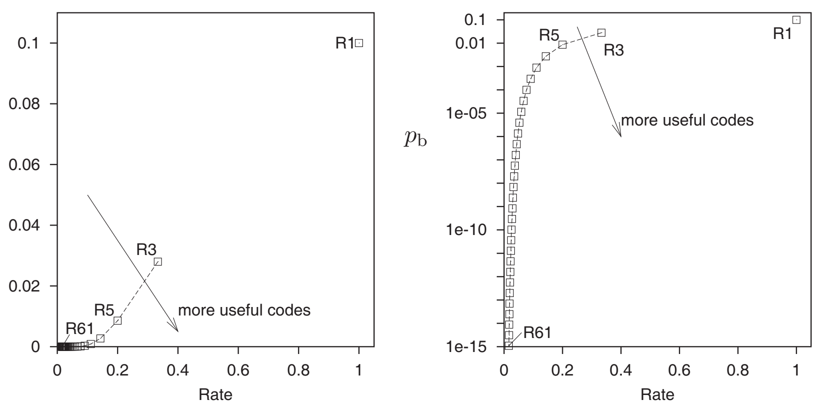


Figure 1.12. Error probability p_b versus rate for repetition codes over a binary symmetric channel with $f = 0.1$. The right-hand figure shows p_b on a logarithmic scale. We would like the rate to be large and p_b to be small.

The repetition code R_3 has therefore reduced the probability of error, as desired. Yet we have lost something: our *rate* of information transfer has fallen by a factor of three. So if we use a repetition code to communicate data over a telephone line, it will reduce the error frequency, but it will also reduce our communication rate. We will have to pay three times as much for each phone call. Similarly, we would need three of the original noisy gigabyte disk drives in order to create a one-gigabyte disk drive with $p_b = 0.03$.

Can we push the error probability lower, to the values required for a sellable disk drive – 10^{-15}? We could achieve lower error probabilities by using repetition codes with more repetitions.

Exercise 1.3.[*3*, p.16] (a) Show that the probability of error of R_N, the repetition code with N repetitions, is

$$p_b = \sum_{n=(N+1)/2}^{N} \binom{N}{n} f^n (1-f)^{N-n}, \tag{1.24}$$

for odd N.

(b) Assuming $f = 0.1$, which of the terms in this sum is the biggest? How much bigger is it than the second-biggest term?

(c) Use Stirling's approximation (p.2) to approximate the $\binom{N}{n}$ in the largest term, and find, approximately, the probability of error of the repetition code with N repetitions.

(d) Assuming $f = 0.1$, find how many repetitions are required to get the probability of error down to 10^{-15}. [Answer: about 60.]

So to build a *single* gigabyte disk drive with the required reliability from noisy gigabyte drives with $f = 0.1$, we would need *sixty* of the noisy disk drives. The tradeoff between error probability and rate for repetition codes is shown in figure 1.12.

Block codes – the (7, 4) *Hamming code*

We would like to communicate with tiny probability of error *and* at a substantial rate. Can we improve on repetition codes? What if we add redundancy to *blocks* of data instead of encoding one bit at a time? We now study a simple *block code.*

A *block code* is a rule for converting a sequence of source bits $\mathbf{s}$, of length K, say, into a transmitted sequence $\mathbf{t}$ of length N bits. To add redundancy, we make N greater than K. In a *linear* block code, the extra $N-K$ bits are linear functions of the original K bits; these extra bits are called *parity-check bits*. An example of a linear block code is the $(7,4)$ *Hamming code*, which transmits $N=7$ bits for every $K=4$ source bits.

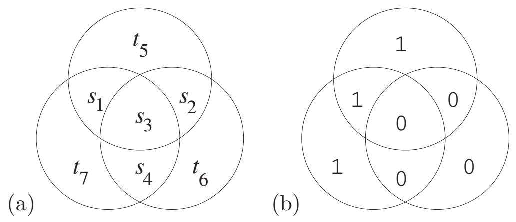


Figure 1.13. Pictorial representation of encoding for the $(7,4)$ Hamming code.

The encoding operation for the code is shown pictorially in figure 1.13. We arrange the seven transmitted bits in three intersecting circles. The first four transmitted bits, $t_1t_2t_3t_4$, are set equal to the four source bits, $s_1s_2s_3s_4$. The parity-check bits $t_5t_6t_7$ are set so that the *parity* within each circle is even: the first parity-check bit is the parity of the first three source bits (that is, it is 0 if the sum of those bits is even, and 1 if the sum is odd); the second is the parity of the last three; and the third parity bit is the parity of source bits one, three and four.

As an example, figure 1.13b shows the transmitted codeword for the case $\mathbf{s}=$ 1000. Table 1.14 shows the codewords generated by each of the $2^4=$ sixteen settings of the four source bits. These codewords have the special property that any pair differ from each other in at least three bits.

$\mathbf{s}$	$\mathbf{t}$	$\mathbf{s}$	$\mathbf{t}$	$\mathbf{s}$	$\mathbf{t}$	$\mathbf{s}$	$\mathbf{t}$
0000	0000000	0100	0100110	1000	1000101	1100	1100011
0001	0001011	0101	0101101	1001	1001110	1101	1101000
0010	0010111	0110	0110001	1010	1010010	1110	1110100
0011	0011100	0111	0111010	1011	1011001	1111	1111111

Table 1.14. The sixteen codewords $\{\mathbf{t}\}$ of the $(7,4)$ Hamming code. Any pair of codewords differ from each other in at least three bits.

Because the Hamming code is a linear code, it can be written compactly in terms of matrices as follows. The transmitted codeword $\mathbf{t}$ is obtained from the source sequence $\mathbf{s}$ by a linear operation,

$$\mathbf{t}=\mathbf{G}^{\mathsf{T}}\mathbf{s}, \tag{1.25}$$

where $\mathbf{G}$ is the *generator matrix* of the code,

$$\mathbf{G}^{\mathsf{T}}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \tag{1.26}$$

and the encoding operation (1.25) uses modulo-2 arithmetic ($1+1=0$, $0+1=1$, etc.).

In the encoding operation (1.25) I have assumed that $\mathbf{s}$ and $\mathbf{t}$ are column vectors. If instead they are row vectors, then this equation is replaced by

$$\mathbf{t}=\mathbf{s}\mathbf{G}, \tag{1.27}$$

where

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{1.28}$$

I find it easier to relate to the right-multiplication (1.25) than the left-multiplication (1.27). Many coding theory texts use the left-multiplying conventions (1.27–1.28), however.

The rows of the generator matrix (1.28) can be viewed as defining four basis vectors lying in a seven-dimensional binary space. The sixteen codewords are obtained by making all possible linear combinations of these vectors.

Decoding the (7, 4) *Hamming code*

When we invent a more complex encoder $\mathbf{s} \rightarrow \mathbf{t}$, the task of decoding the received vector $\mathbf{r}$ becomes less straightforward. Remember that *any* of the bits may have been flipped, including the parity bits.

If we assume that the channel is a binary symmetric channel and that all source vectors are equiprobable, then the optimal decoder identifies the source vector $\mathbf{s}$ whose encoding $\mathbf{t}(\mathbf{s})$ differs from the received vector $\mathbf{r}$ in the fewest bits. [Refer to the likelihood function (1.23) to see why this is so.] We could solve the decoding problem by measuring how far $\mathbf{r}$ is from each of the sixteen codewords in table 1.14, then picking the closest. Is there a more efficient way of finding the most probable source vector?

Syndrome decoding for the Hamming code

For the (7, 4) Hamming code there is a pictorial solution to the decoding problem, based on the encoding picture, figure 1.13.

As a first example, let's assume the transmission was $\mathbf{t} = 1000101$ and the noise flips the second bit, so the received vector is $\mathbf{r} = 1000101 \oplus 0100000 = 1100101$. We write the received vector into the three circles as shown in figure 1.15a, and look at each of the three circles to see whether its parity is even. The circles whose parity is *not* even are shown by dashed lines in figure 1.15b. The decoding task is to find the smallest set of flipped bits that can account for these violations of the parity rules. [The pattern of violations of the parity checks is called the *syndrome*, and can be written as a binary vector – for example, in figure 1.15b, the syndrome is $\mathbf{z} = (1, 1, 0)$, because the first two circles are 'unhappy' (parity 1) and the third circle is 'happy' (parity 0).]

To solve the decoding task, we ask the question: can we find a unique bit that lies *inside* all the 'unhappy' circles and *outside* all the 'happy' circles? If so, the flipping of that bit would account for the observed syndrome. In the case shown in figure 1.15b, the bit r_2 lies inside the two unhappy circles and outside the happy circle; no other single bit has this property, so r_2 is the only single bit capable of explaining the syndrome.

Let's work through a couple more examples. Figure 1.15c shows what happens if one of the parity bits, t_5, is flipped by the noise. Just one of the checks is violated. Only r_5 lies inside this unhappy circle and outside the other two happy circles, so r_5 is identified as the only single bit capable of explaining the syndrome.

If the central bit r_3 is received flipped, figure 1.15d shows that all three checks are violated; only r_3 lies inside all three circles, so r_3 is identified as the suspect bit.