Error correcting codes

Lecture 1: motivating examples

Notation I shall carry forward some notation from the crypto part of the course. $\mathbb{Z}_2 = \{0,1\}$, the field with 2 elements. Officially, $\mathbb{Z}_2^n$ is the set of all vectors with n components over the field $\mathbb{Z}_2$ but rather than writing $(a_1, \ldots, a_n) \in \mathbb{Z}_2^n$ we shall usually write $a_1 \ldots a_n$ to simplify notation. All messages between Alice and Bob will be elements of $\mathbb{Z}_2^n$. It is always possible to convert information into sequences of bits (= elements of $\mathbb{Z}_2$) so this is no real restriction.

The goal of this final part of the course is to show you another class of codes. This time, we are not trying to keep secrets, instead we are trying to cope with the fact that communication channels are noisy and so introduce errors into the messages that are transmitted over them.

Example Alice wishes to communicate with Bob about selling or buying some shares as follows:

message	meaning
0	sell
1	buy

We shall also assume that the Communication Channel is such that there is a small but nevertheless non-zero probability that a bit will be zapped by noise so that a 0 is changed into a 1 and vice-versa.

Alice sends Bob the message 0 ("sell") but due to noise it is converted into a 1 ("buy"). Thus Bob does the opposite of what Alice wanted.

How can we deal with this problem?

We send the same bit twice.

message	codeword sent
0	00
1	11

s'pose now Alice wants to tell Bob to sell. She does not send 0 instead she sends 00. Assume again that one of the bits is zapped so that Bob receives 10. What can he deduce? Well, $10 \neq 00, 11$ so he knows that errors have occurred — he has <u>detected</u> errors. However, he still doesn't know whether 00 or 11 was sent.

We can obviously improve matters by sending the same message three times.

message	codeword sent
0	000
1	111

Alice sends 000; an error occurs so Bob receives 010. Again Bob knows that something is wrong AND assuming that no more than one error has occurred he knows that the codeword sent was 000 (and not 111).

Of course, we have paid a price to gain this ability - the codewords are longer than the messages - but there are many circumstances where this is a price worth paying.

The set $\{000, 111\} \subseteq \mathbb{Z}_2^3$ is called a code. We have defined an encoding function

$$\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2^3$$

given by $x \longmapsto xxx$.

This is an example of a 3-fold repetition code.

Example We s'pose this time that Alice wishes to send Bob 4 messages

message	meaning
00	apple
10	orange
01	banana
11	cumquat

We of course have the same problem as before: Alice sends 00 but Bob receives, say, 11: Instead of "apple" she gets "Cumquat". Consider now the following encoding

Message	Codeword
00	000
10	101
01	011
11	110

This time Alice adds a final bit so that the number of 1's in the codeword is even – for this reason it is called a parity check bit (or digit). Formally,

$\mathbb{Z}_2^2 \longrightarrow \mathbb{Z}_2^3$ is given by $xy \longmapsto xy(x+y)$

Suppose now that an odd no. of errors occur. Then Bob can detect them. For example, suppose Alice sends 101 and Bob receives 111. Since $1+1+1 \neq 0$ he knows that errors have occurred. Again, he can detect some errors.

Observe, that he has not paid a very high price to gain this extra information because he has only added one extra bit (rather than repeating the message to the same effect at the price of adding two bits).

The art and science of the theory of error correcting codes is to add the smallest no. of extra bits to a message in order to be able to correct as many errors as possible.

Example This is a more realistic set of examples closely connected to the origins of the subject in the late 1940's and early 1950's.

Supse we wish to send 16 messages from $\mathbb{Z}_2^4$ in such a way that if an error occurs we are guaranteed to be able to correct it. How should we encode the message and what is the smallest no. of bits that we need to add on?

(1) Our first attempt is a 3-fold repetition code. We define the encoding function $\mathbb{Z}_2^4 \longrightarrow \mathbb{Z}_2^{12}$ by

$x_1x_2x_3x_4 \longmapsto x_1x_2x_3x_4\, x_1x_2x_3x_4\, x_1x_2x_3x_4$.

We can certainly correct 1 error but we have had to pay the heavy price of adding an extra 8 bits to our message to achieve this.

(2) Our second attempt is due to Richard Hamming at the end of the 1940's. We define an encoding function $\mathbb{Z}_2^4 \longrightarrow \mathbb{Z}_2^8$ as follows:

(i) Message = $x_1 x_2 x_3 x_4$.

(ii) Write in a square

x_1	x_2	
x_3	x_4	

(iii) Calculate the 4 new bits shown ("partial parity checks")

x_1	x_2	x_1+x_2
x_3	x_4	x_3+x_4
x_1+x_3	x_2+x_4	

(iv) Write the result as a string

$$x_1 x_2 (x_1+x_2)\ x_3 x_4 (x_3+x_4)(x_1+x_3)(x_2+x_4)$$

I shall now show how this method can be used to correct 1 error.

We wish to send the message 1101.

We carry out the above encoding procedure to determine the codeword that is transmitted

1	1	0
0	1	1
1	0	

Codeword = $1\,1\,0\,0\,1\,\overset{*}{1}\,1\,0$

Let's now say an arbitrary bit - the one marked with *.

Thus $1\,1\,0\,0\,1\,0\,1\,0$ is the received vector.

We redraw the square

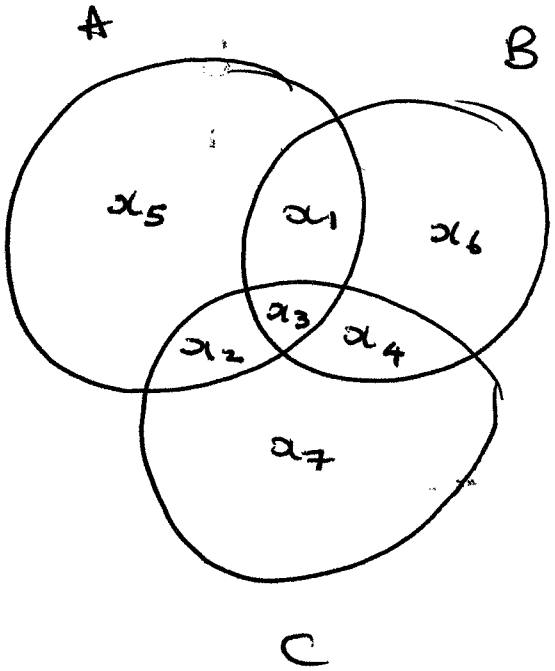
1	1	0 ✓	(row ok)
0	1	0 X	(row wrong)
1	0		
✓	✓	← columns ok	

We see that the 6th bit is in error.

(3) Our final method is again due to Richard Hamming. This will lead to a function $\mathbb{Z}_2^4 \to \mathbb{Z}_2^7$.

message = $x_1 x_2 x_3 x_4$

We now draw up a Venn diagram



where $x_5 = x_1 + x_2 + x_3$
$x_6 = x_1 + x_3 + x_4$
$x_7 = x_2 + x_3 + x_4$

} partial parity checks

Thus each circle sums to 0

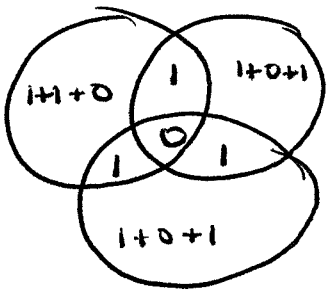
The codeword is

$x_1 x_2 x_3 x_4 x_5 x_6 x_7$

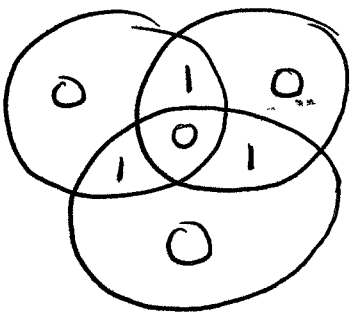
When a vector is received we redraw the Venn diagram. A circle that does not sum to zero must contain a bit that is in error.
By combining the information from the three circles we can locate that bit

Here's an example.

Message = 1 1 0 1
1 2 3 4



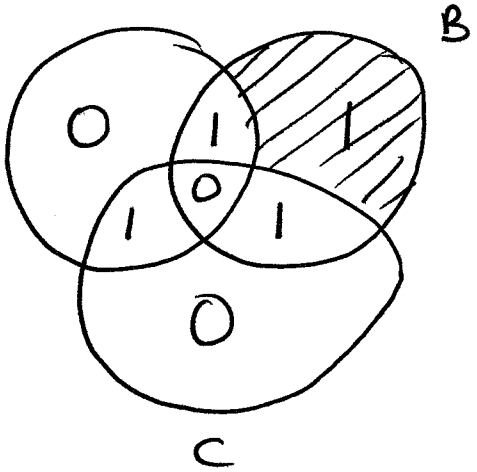
We get



Codeword = 1 1 0 1 0 $\overset{*}{0}$ 0

Suppose now that the bit indicated is flipped and the received vector is 1 1 0 1 0 1 0
1 2 3 4 5 6 7

We redraw the Venn diagram



A ✓ B ✗ C ✓

Thus the error is in

$B \setminus (A \cup B)$

which I have shaded

Our third example is clearly the most efficient so far and you might wonder if we can do any better.

The answer is 'no'.

The reason is that m bits describe 2^m binary no's and so intuitively 2^m pieces of information.

If we added 2 extra bits onto our message we would have a string of length 6. The error can occur in any one of the 6 bits or no errors might have occurred. We would therefore need 7 pieces of information and we only have 4 ($=2^2$).

Thus the code Hamming constructed is the best solution to our original question.