NOTES

Error Correcting Codes

John Hannah

Introduction

In this article, I will describe an elementary approach to error-correcting codes which can be presented to second year (general level) students. In this approach, students can see simple abstract concepts being used to solve an easily described practical problem.

Although most texts give the impression that you need to know some finite field theory in order to learn coding theory, you can in fact get a good grasp of the basic ideas by knowing about little more than matrix multiplication and modulo – 2 arithmetic. Thus, for example, I include such codes as a brief topic in my second year linear algebra course (to help justify looking at abstract vector spaces rather than just spaces over the real numbers). Codes could also be discussed in introductory courses on abstract algebra or discrete mathematics.

From the student's point of view, the need for error-correcting codes is easily appreciated. Digital data occur in many parts of modern life. Information is stored as strings of binary digits (0 and 1) in such diverse areas as computers, satellites and record-players. It is important to be able to transfer such data reliably between different systems, whether it be between the memory and the processor of a computer, or between Earth and Voyager satellite near Uranus. Unfortunately, most communication channels are prone to noise of one sort or another, and errors can appear in the data. Coding theory tries to construct efficient ways of sending digital data while at the same time guarding against these errors.

The Parity-check Code

Perhaps the simplest example of a binary code is the parity-check code. Here original data, a block of n binary digits, has an extra "check digit" attached, and a block of $n+1$ digits is transmitted instead. The check digit is 0 if there are an even number of 1's in the original message, and 1 if there are an odd number. Hence the name "parity check" code. Using addition moduo 2 the encoding procedure can be expressed as:

$$\underset{\text{information digits}}{(a_1, a_2, \ldots, a_n)} \quad \rightarrow \quad \underset{\text{codeword}}{(a_1, a_2, \ldots, a_n, a_{n+1})}$$

where

$$a_{n+1} = a_1 + a_2 + \cdots + a_n.$$

For example, if we were using 4-digit blocks we would have the encodings

$$(1101) \rightarrow (11011)$$

and

$$(1010) \rightarrow (10100).$$

In this code a correctly received codeword must have an even number of 1's. Hence if there is exactly one error during transmission, the receiver will realize that an error has occurred. For example (10101) cannot come from a correctly sent message. So we can say that the parity check code *detects all single errors*. If there is no error detected then the receiver decodes the message simply by removing the redundant digit a_{n+1}.

There are two drawbacks to this simple code. Firstly, if only one error does occur, we still cannot tell which digit is wrong, and so we cannot correct the error. For example (10101) could have come from a single error in either (10100) or (11101). Secondly, double errors go undetected. For instance, errors in the second and fifth digits of (11011) result in a received word (10010) which looks correct since it satisfies the parity check $1 + 0 + 0 + 1 = 0$.

The idea behind coding theory is to look for more sophisticated "parity checks" which will improve the performance of the above code. By introducing extra redundant digits (like a_{n+1} in the parity check code) we can home in on the incorrect digit and detect more errors. Of course, if there are too

many redundant digits the code will become inefficient: the actual message will make up only a small part of the codeword and transmission will become time-consuming and expensive. So good codes have as few redundant digits as possible and detect or correct as many errors as possible.

The ISBN Code

The ISBN numbers, which are used to classify books, are another example of a code. Again just one check digit is involved, but this time the arithmetic is done modulo 11. In a typical ISBN number

$$0 - 474 - 00130 - X$$

the first nine digits $a_1, a_2, \ldots, a_9$ are the information digits, and the check digit a_{10} is calculated from the formula

$$a_{10} = \sum_{n=1}^{9} n a_n \pmod{11}.$$

If, as in the above example, this check digit is 10, it is written as X. It is easy to see that this code again detects all single errors. The coefficients n are used (instead of the 1 used in the parity check code) so that the code will also detect all transposition errors, where two digits are accidentally interchanged. Transpositions are among the most common errors that occur when data are communicated by humans (rather than by electric or magnetic fields!)

I will discuss binary codes in the rest of this article, but obviously a discussion of why modulo-11 arithmetic is used here would fit well into an abstract algebra course.

Constructing an error-correcting code

To illustrate the ideas involved, I shall show how to construct one of the family of Hamming codes. These codes can correct single errors or, alternatively, detect double errors.

Suppose we allow ourselves four check digits, instead of the single check digit in the above parity check code, and suppose that our data consists

of strings of n information digits $(a_1, a_2, \ldots, a_n)$. Each of the check digits $a_{n+1}, \ldots, a_{n+4}$ will come from an equation of the form

$$a_{n+1} = \text{sum of some of } a_1, a_2, \ldots, a_n$$

and these same four equations will be used as check equations for the received message $(a_1, a_2, \ldots, a_n, a_{n+1}, \ldots, a_{n+4})$. We can arrange these check equations in matrix form as

$$Ha = 0$$

where a is the received codeword, 0 is the zero vector and H is a matrix of the form

$$H = [Q \mid I_4]$$

where I_4 is the 4×4 identity matrix and Q is a $4 \times n$ matrix of 0's and 1's. These equations determine whether a is a correctly encoded string of $n+4$ digits. Clearly if the received codeword gives $Ha \neq 0$, then an error has occurred. But what is the precise effect on Ha of an error in the codeword a? We can represent the received codeword as $a + e$, where a is the intended codeword, and the vector e has a 1 in each position where an error occurred but has zero components otherwise. This is because each of the errors $0 \to 1$ and $1 \to 0$ can be obtained by adding 1 (mod 2) to the original entry. When we test the received codeword using the matrix H we get

$$H(a+e) = Ha + He = 0 + He.$$

If there has been exactly one error, in the ith position say, then we have

$$He = i\text{th column of } H.$$

For example, an error in the first entry corresponds to having $e = (1, 0, \ldots, 0)^T$ and the calculation of $H(a+e)$ will yield the first column of H.

Thus to be able to *detect the existence* of one error, we just need to make sure that each column of H is nonzero (that is, not every entry in the column is zero). If we also want to *locate the position* of such an error, then we just need to make each column of H different. Notice that since the only possible entries are 0 and 1, locating the position of an error is the same as being able to correct that error.

In our example H has four rows and so there are $2^4 = 16$ possible columns to choose from, all but one of them being nonzero. Thus if we want to be

able to correct all single errors, then H must have at most 15 columns. This means that 4 check digits can be used to protect strings of up to $11(= 15 - 4)$ information digits. If we use 4 check digits to protect exactly 11 digits then there is esentially only one possible marix H (since swapping the columns of H amounts to relabelling the original digits):

$$H = \left[\begin{array}{cccccccccc|cccc} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right]$$

Once H is found we can write down the equations for the check digits $a_{12}, a_{13}, a_{14}, a_{15}$ in terms of the information digits $a_1, a_2, \ldots, a_{11}$.

From the point of view of hand-calculations (which is all I expect my students to do), correcting single errors is a simple procedure: in the above notation, you search among the columns of H for one that looks like He. Of course, this is not very satisfactory if you intend to use a computer. The searching part of the algorithm can be sidestepped though, if you are willing to rearrange all 15 columns of the above matrix H. The idea is to use for the ith column of H the binary representation of the number i, so that when He is calculated it tells you directly which entry was wrong. (See the article by Levinson [2]).

The same code can also be used to detect double errors, but this time correction is not possible. The same calculation as before shows that with two errors $H(a + e) = He$ is a sum of two columns from H, and since all these columns are different, their sum (modulo 2, of course) must be nonzero. So the error is detected. But with our matrix H the sum of any two columns is another column, since H contains all possible nonzero columns. So this double error would be indistinguishable from some other single error.

This is as far as I take my students. After all, it is a course on linear algebra. In an abstract course, you could go further: to correct two or more errors you really need to construct finite fields of order 2^m. My colleague, Kevin O'Meara, offers such a course to third-year students. My main aim is to whet their appetities by showing what can be achieved using a few simple ideas, and by making them aware that there is still more to be achieved.

References

[1] Clark, G.C. and Cain, J.B. *Error-correction coding for digital communications*, Plenum, 1981 pp.1–7 and 49–62. Their illustration (on pages 59–61), using the problem of identifying one counterfeit coin among a collection of coins, may appeal to you if you wanted to incorporate these ideas in a discrete mathematics course. Another interesting introduction to the subject is offered by

[2] N. Levinson, *Coding Theory: a counterexample to G.H. Hardy's conception of applied mathematics*, Amer. Math. Monthly 77 (1970), 249–258. For a more advanced, but still very readable treatment, you could try:

[3] V. Pless, *Introduction to the theory of error-coding codes*, Wiley, 1982.

University of Canterbury,
Christchurch,
New Zealand.