A BRIEF INTRODUCTION TO CODING THEORY

KIMBALL MARTIN
SEPTEMBER 30, 2002

Coding theory is the theory of sending information accurately over noisy channels. It lies in the intersection of mathematics, computer science and electrical engineering. Coding theory is needed in many diverse applications such as in satellite transmission and CD recording. Let me illustrate the basic ideas of coding theory with the following example. Suppose you're sending emails back and forth with your friend and you receive:
DO YOU WNT TO SEE A OVIE THIS WEEKND?
So you notice there are typos in his email. This is called *error-detection*. You are able to do this because English has a certain structure to it. If he was just typing a random string of characters (say some password), you would have no idea if there were any errors (until you tried it). There are two ways to understand his message. First you could ask him for confirmation, i.e. you could ask
DID YOU ASK IF I WANTED TO SEE A MOVIE THIS WEEKEND?
or
WHAT DID YOU SAY?
Of course in this example, if you did that, he'd think you were stupid. But still, there is such a protocol for *error-detecting codes*. These are codes (to be described more precisely later) by which you can detect errors but not correct them. The simplest example is the "parity check code," where the sum of all the digits in each word is even. If you detect an error in the data you receive, you simply ask the other party to resend it. In some applications, this may be all you want to do–e.g., to make sure a credit card number or license number is valid. Of course, this could be quite inconvenient if you're telling a satellite something and by the time you hear it say "WHAT" it's on the other side of the earth, so you need to wait until tomorrow to transmit again (Perhaps this is a bit of an exaggeration, but you get the point). It would be much better to be able to correct the errors in real-time. I believe that there's not a whole lot of research still to be done on error-detecting codes, so most of the current research focuses on "error-correcting" codes, as we will do.

In our example, this means you would simply interpret his email to be:
DO YOU WANT TO SEE A MOVIE THIS WEEKEND?
Here we simply correct the apparent three errors by common sense. Of course, there's a small possibility that we've made a mistake. For example, there could have been four errors and the original message was
DO YOU WANT TO SEE A BOVINE THIS WEEKEND?
But this is highly unlikely and probably wouldn't merit a response anyway. So in error-correcting codes you correct errors you notice into the most probable intended word. Thus you want to find a code with high probability of proper correction.

We will be concerned with the following general model. First we will be interested in *binary codes* of length n. This means you have a set of codewords C which are strings of length n of 0's and 1's. The idea is to take the original data, encode it into codewords with some encoding algorithm, transmit your encoded data, receive it, correct errors and decode it. We will assume the only possible errors that can occur are that some bits might get flipped across the noisy channel. (This is sufficient for many applications, though in some applications other errors may occur such as bits being deleted or added.)

Let me illustrate the process the repetition code of length 3. Say you want to send the following nibble (string of 4 bits): 1010. To encode, you repeat each bit 3 times to get: 111 000 111 000. You send this over the noisy channel and say you receive: 110 100 111 001. To decode, you just take the majority vote on each set of 3 bits, and get 1010 back. Notice there were three errors total and two of them in a row. Fortunately they were on different codewords. If we had two errors on the same codeword, say we received: 100 000 111 010, then we would incorrectly decode this as 0010. So the repetition code of length 3 can detect up to 2 errors per word but can (rightly) correct only 1 error per word. Now you can define a repetition code of any odd length $2k + 1$ which will detect up to $2k$ errors and correct up to k errors per word.

Generally you choose your code based on what kind of noise you expect in your channel: What percentage of bits will be flipped? Is the noise sparse or well spread out, or does it come in bursts? Often errors come in bursts, i.e. 7 out of 10 bits in a row get flipped, but then out of the next 120 bits, only 2 or 3 get flipped. So you wouldn't want a code of length 10, you might want one of length 128 that could correct up to 15 errors per word to be safe. Now you could get by with a repitition code of length 31, but if you have a lot of data to send then your code isn't very efficient–for every 31 bits you receive, you only get one bit of data. We'll see in the next section how we can do much better than that.

Linear Codes

Most type of codes that are studied are linear codes. I'll start throwing out some terminology with some informal definitions to get us started, but if you don't understand the definitions it's not that important. The main point of learning this notation is to be able to understand the literature. Also, it's a convenient way to speak of things and may serve you well in higher math.

Definition 1. *Let $\mathbb{Z}$ denote the integers. For a positive integer n, let $\mathbb{Z}/n$ denote the integers modulo n. (For an integer a, we let $a \pmod n$ be the remainder r after dividing a by n. Thus $a = kn + r$ for some integer k.) Inotherwords, $\mathbb{Z}/n$ is the set of possible remainders after division by n. As a set $\mathbb{Z}/n$ is $\{0, 1, ..., n-1\}$. Now $\mathbb{Z}/n$ inherits addition and multiplication from $\mathbb{Z}$ as follows. We let $a \pmod n + b \pmod n = a + b \pmod n$ and define multiplication similarly.*

Definition 2. *Let F be a set with 0, 1, addition, subtraction, multiplication and division (by non-zero elements) satisfying the usual properties (commutativity, transitivity, associativity and distributitivity). Then F is called a* field*. If F is a finite set, then F is called a* finite field*.*

Example 1. *The rational numbers $\mathbb{Q}$ form a field.*

Example 2. *The real numbers $\mathbb{R}$ form a field.*

Example 3. *The complex numbers $\mathbb{C}$ form a field.*

Example 4. *The integers $\mathbb{Z}$ do not form a field. You don't always have division. For example 3/2 is not defined as an integer. However $\mathbb{Z}/n$ is a* ring*, that is it has all properties of a field except possibly division.*

Exercise 1. *Show that $\mathbb{Z}/2$ is a field. Show the same for $\mathbb{Z}/3$ and $\mathbb{Z}/p$ where p is a prime number, if you can.*

Exercise 2. *Show that $\mathbb{Z}/4$ is not a field. Show the same for $\mathbb{Z}/n$ where n is a composite number (i.e., a non-prime bigger than 1). (However they are rings.)*

In fact, the following is true:

Theorem 1. *There exists a finite field with n elements if and only if n is a prime power p^k. In fact there is only one finite field of size p^k (up to relabeling the elements), usually denoted $\mathbb{F}_{p^k}$.*

Note by Exercise 2, $\mathbb{F}_{p^k} \neq \mathbb{Z}/p^k$ unless $k = 1$. Rather, it is constructed by considering the polynomials with coefficients in $\mathbb{Z}/p$ modulo an irreducible polynomial of degree k. The study of fields, rings and something called groups forms the subject of algebra (not the high school kind).

Now we turn to fundamental object of study of a specific kind of algebra, linear algebra. This is the notion of a vector space.

Definition 3. *Let F be a field. A* vector space *or* linear space over F *is a set V with a 0, addition, subtraction and scalar multiplication by elements in F.*

These are sometimes called F-vector spaces or F-linear spaces. We will be dealing with *finite-dimensional* vector spaces over $\mathbb{F}_2 = \mathbb{Z}/2$. Hence from now on we will write $\mathbb{F}$ for $\mathbb{F}_2$. You can think of the typical n-dimensional $\mathbb{F}$-linear space as the set of all vectors of length n with coordinates in $\mathbb{F}$. We will denote this by $\mathbb{F}^n$. Since as a set $\mathbb{F} = \{0, 1\}$, you can think of $\mathbb{F}^n$ as all strings of n bits with mod 2 addition. (Scalar multiplication doesn't mean much here, you can only multiply by 0 or 1.)

Exercise 3. *Write out all elements of $\mathbb{F}^2$. Compute the sum of every pair of elements in $\mathbb{F}^2$.*

Exercise 4. *The set of $n \times n$ matrices over $\mathbb{F}$ form an n^2-dimensional vector space over $\mathbb{F}$. Of course, they're more than a vector space because you can multiply them too–this makes them into a ring with noncommutative multiplication for $n > 1$. Show that multiplication is not commutative if $n > 1$.*

Exercise 5. *Look up the term "passing math."*

Now I haven't told you what dimension really means, though you know $\mathbb{F}^n$ has dimension n. Well, I will. We will be concerned with vector spaces contained in $\mathbb{F}^n$, also called *subspaces* of $\mathbb{F}^n$. Let V be a subspace of $\mathbb{F}^n$ (and we write $V \leq \mathbb{F}^n$).

Definition 4. *Let S be a subset of elements of $\mathbb{F}^n$. The linear span of S, denoted by $< S >$, is the set of all possible linear combinations of elements in S. That is*

$$< S >= \{a_1x_1 + a_2x_2 + \cdots + a_nx_n | a_i \in \mathbb{F}, x_i \in S, n \geq 0\}.$$

Definition 5. *Let S be a subset of $\mathbb{F}^n$. We say S is* linearly independent (LI) *if no element of S is a (finite) linear combination of other elements in S. Equivalently, S is LI if and only if for any $x_1, x_2, ..., x_m \in S$ (Notation: $\in$ means "in" or "is an element of") the equation,*

$$a_1x_1 + a_2x_2 + \cdots + a_mx_m = 0, \ \ (a_i \in \mathbb{F})$$

holds only if $a_1 = a_2 = \cdots = a_m = 0$. Otherwise, S is a linearly dependent *set.*

Definition 6. *If $V =< S >$ and S is LI, then S is called a* basis *for V.*

Theorem 2. *Every vector space V has a basis B (in general, it has many bases). In our setting ($V \leq \mathbb{F}^n$), B is always finite. All bases for B have the same cardinality (size) $|B| = k$. The number k is the* dimension *of V.*

Exercise 6. *The standard basis for $\mathbb{F}^2$ is $\{e_1 = [1, 0], e_2 = [0, 1]\}$. Show this is a basis. Do the same for $\mathbb{F}^n$.*

Alright, I'll finally tell you what a linear code is now.

Definition 7. *Let C be a k-dimensional subspace of $\mathbb{F}^n$. Then we say C is a* linear code *of length n and dimension k.*

Linear codes have three fundamental parameters: n, k and d. Here n and k are as above, and I will define d shortly.

Exercise 7. *Prove if the dimension $\dim C = k$, then $|C| = 2^k$. Also show that your encoded data will be n/k times larger than your unencoded data.*

Definition 8. *The* weight *of a vector $v \in \mathbb{F}^n$ is the number of 1's that occur in v. We'll write this as $wt(v)$. Let $v, w \in \mathbb{F}^n$. The* (Hamming) distance *between v and w is $d(v, w) = wt(v - w)$. (Note since we are working mod 2, $v - w = v + w = w - v$.)*

Exercise 8. *Show that $d(v, w)$ is the number of coordinates in which v is different from w.*

Definition 9. *Let C be a linear code of length n and dimension k. The* minimum distance *C is defined $d = \min d(v, w)$, where the minimum is taken over all $v, w \in C$ with $v \neq w$. Then we say C is an (n, k, d)-linear code.*

Exercise 9. *Show that d is the smallest weight of all nonzero codewords $c \in C$ for a linear code C.*

Having minimum distance d means all codewords are at least d distance apart. Let $t = d/2 - 1$ or $(d - 1)/2$ if d is odd. Then around each codeword you can make a ball of radius t (draw a picture) such that none of the balls will intersect. Consider the ball $B(c, t)$ around the codeword c of radius t. Any $v \in B(c, t) \subseteq \mathbb{F}^n$ differs from c in at most t places. Thus if you transmit a codeword c across a noisy channel and there are no more than t errors, the received vector v will lie in $B(c, t)$. Thus the code C can correct t errors per word.

Exercise 10. *Show that the repetition code of length n is a linear code. What are its parameters?*

Exercise 11. *Let C be an (n, k, d)-linear code and let $c \in C$ and t as above. Compute $|B(c, t)|$ and $|\bigcup_{c \in C} B(c, t)|$. If the latter quantity equals 2^n, then the code is called* perfect. *Can you think of why? Perfect codes are quite rare.*

The next exercise will show you how to encode your data.

Exercise 12. *(Encoding) Let C be an (n, k, d)-linear code. Take your stream of unencoded binary data. To encode it you will break it up into words of length k, i.e. you will have a sequence of elements in $\mathbb{F}^k$. Let $e_1, e_2, ..., e_k$ be the standard basis for $\mathbb{F}^k$. Let $b_1, b_2, ..., b_k$ be a basis for C. Construct a matrix G that sends e_i to b_i for each i. Check that this maps $\mathbb{F}^k$ one-to-one onto C. This is the encoding scheme and G is called the* generating matrix. *Note the encoding depends on the basis for C, so when you decode you will need to use the same basis.*

So encoding is efficient (both time and space), though decoding is not quite as efficient for general linear codes. For certain types of linear codes, such as *cyclic codes* decoding is also very efficient. Thus much of the theory of linear codes goes into finding certain classes of linear codes with efficient decoding algorithms, as well as getting good parameters (n, k, d).

I'll skip the general decoding algorithm now. You can find it in pretty much any introduction to coding theory.

The parameters of a linear code satisfy certain properties, such as

Exercise 13. *(Singleton bound) Let C be an (n, k, d)-linear code. Show $d \leq n - k + 1$.*

Codes that attain this bound sharply (i.e., for which $d = n-k+1$) are called *maximum distance separable (MDS)* codes. Inotherwords, for an MDS code, $k = n - d + 1$. So in the example at the end of the last section, I said we might want a code with $n = 128, d = 31$. Then we might like an MDS code with parameters (128,98,31). However, such a code doesn't exist. MDS codes are pretty rare too. The best linear code there is with desired n and d was a (128, 40, 31) code (see *http://www.win.tue.nl/˜aeb/voorlincod.html*). Then by Exercise 7, encoding will only multiply the size of the data by 3.2, instead of 31 as in the repitition code. There is also a well known so-called second-order Reed-Muller code with parameters (128, 29, 32), which will only increase the data size by a factor of about 4.4.

Covering Radius and Lightbulb Codes

Now we're moving on to our problem! The idea of correcting errors is to use *maximum likelihood decoding*, you correct each received word to the codeword closest to it. This applies even when you receive words not lying in any $B(c, t)$ (notation as in previous section). Another sort of measure of effectiveness of the code is to roughly measure how many words don't lie in any of these balls. In a rough sense, this is done with the covering radius of the code, which more precisely measures how far received words can be from the codewords, or equivalently, these balls. The study of the covering radius in coding theory is a fairly recent one.

Definition 10. *Let C be a linear code in $\mathbb{F}^n$ and $x \in \mathbb{F}^n$. The* distance *from x to C is given by $d(x, C) = \min_{c \in C} d(x, c)$. The* covering radius *of C is $R = \max_{x \in \mathbb{F}^n} d(x, C)$.*

The lower the covering radius, the better the code is in the above sense. Some lightbulb and related codes are known to have relatively low covering radius.

Exercise 14. *With notation as above, show that $R \geq t$. If C is a perfect (n, k, d) code, what is R?*

Now we will define the *lightbulb code* L_n of length n^2. We will think of the space of all possible words $\mathbb{F}^{n^2}$ as $n \times n$ matrices over $\mathbb{F}$. Let R_i be the $n \times n$ matrix with all 1's in the i^{th} row and 0's elsewhere. Similarly let C_j be the $n \times n$ matrix with all 1's in the j^{th} column and 0's elsewhere. We'll call the R_i's and the C_j's the row and column matrices, respectively. Then $L_n =< R_i, C_j | 1 \leq i, j, \leq n >$, that is the space spanned by the row and column matrices.

Exercise 15. *Show that* $\dim L_n = 2n - 1$.

Exercise 16. *Show that R_n in the lightbulb game really is the covering radius of L_n.*