Coding Theory: Math 454 Lecture 19 (7/31/2017)

Cole Franks

December 17, 2017

Contents

1 Error correcting codes

Error correcting codes are methods to encode messages for transmission through error prone communications channels.

Example 1. In international Morse code, messages are sent by dashes and dots. To represent a dot, you might turn on a light for a short period of time. To represent a dash,

you might turn it on for a longer period, say 3 dot durations. To represent a space between words, wait for 7 dot durations.

$$SOS = \bullet\bullet\bullet PAUSE - - - PAUSE \bullet\bullet\bullet$$

However, if the transmitting ship is very far from the receiving ship or there is a black hole in between them which slows down the light rays, the above message might be mistaken for

$$DOG = - \bullet\bullet PAUSE - - - PAUSE - - \bullet .$$

Remark 1.1. The type of errors matter a great deal. Perhaps there are deletions, insertions, etc. Perhaps the message is spoken words, and the errors are in forms of syllables.

In general, we think of sending messages in the following model.

$$\text{message } m \xrightarrow[\text{encoding}]{} \text{codeword } c \xrightarrow[\text{channel}]{} \text{received word } c' \xrightarrow[\text{decoding}]{} m'$$

We think of the channel as where the errors occur. Ideally, after decoding, m' would be equal to m, but if there are too many errors, that may not be possible.

1.1 Binary bit-error correcting codes

First, we make the assumption that the *messages* we want to send are binary strings. This is not much of an assumption, because anything can be written down in binary.

Next, we make a stronger assumption that we will also only be transmitting binary strings across the channel and the only errors that occur are *bit-errors*, which consist of replacing a 1 by a 0 or a 0 by a 1.

Definition 1.1 (code)**.** A code is a subset of the binary strings of length n. The parameter n is called the *block-length* of the code.

Example 2. Let $C = \{001, 010, 100, 111\}$. This is a code of block-length 3. Using C, we can send a message of 2 bits. That is, a binary string of length 2. To do this, we come up with a correspondence between messages and codewords, which we call the *encoding*.

$$00 \to 001, 01 \to 010, 10 \to 100, 11 \to 111.$$

Next, we transmit the encoding of the message we wanted to send. A possible situation is

$$11 \xrightarrow[\text{encoding}]{} 111 \xrightarrow[\text{channel}]{} 110 \xrightarrow[\text{decoding}]{} m'.$$

In this example, the transmitted codeword was 111, and a bit-error occurred in the last position.

If in addition we know that our channel was only capable of producing one error, we can deduce that an error occurred in this situation, because 110 is not in the code C. However, we cannot figure out which codeword it came from, because 110 can be made from $100, 010$ and 111 by one bit-error, so the original message could have been any of $00, 01, 11$.

Definition 1.2 (Hamming distance)**.** The *Hamming distance* $d(x, y)$ between two strings is the number of positions in which they differ. For example, $d(001, 100) = 2$.

Fact 1.2 (Key fact)**.** If a codeword c is transmitted across a channel capable of producing at most r bit-errors, then

$$d(c, c') \leq r$$

for the received word c'.

Fact 1.3. The Hamming distance is symmetric and satisfies the triangle inequality, that is $d(x, y) = d(y, x)$, and $d(x, y) \leq d(x, z) + d(y, z)$. The proof is left as an exercise.

Definition 1.3 (Distance of a code)**.** In coding theory, we have the conflicting goals of sending lots of information (that is, being able to send many possible messages) in a short message, and preventing confusion due to errors. First we tackle the "avoiding confusion" angle, and then we talk about trying to send lots of information.

The distance of a code is the minimum Hamming distance between two distinct codewords in the code. For example, the distance of $\{001, 010, 100, 111\}$ is 2.

1.2 Correcting, Detecting Errors

Detecting errors just involves finding out if an error occurred.

Definition 1.4. We say a code can *detect* r errors if the following holds:
Whenever a codeword is transmitted across a channel capable of producing r bit-errors, it is possible to deduce whether an error occurred using only the received word.

Using our diagram from before, error detection consists of reading c' and deciding whether an error occurred.

$$\text{message } m \xrightarrow[\text{encoding}]{} \text{codeword } c \xrightarrow[\text{channel}]{} \text{received word } c' \xrightarrow[\text{decoding}]{} ?$$

We don't find out what m or c is just using error correction.

Fact 1.4. If the distance of a code is d, then the code can detect exactly $d - 1$ errors.

Proof. If a channel can produce at most $d - 1$ bit-errors, it cannot turn one codeword into another by flipping bits, because the minimum Hamming distance between any two

codewords is at least d. Thus, if the received word c' is not in the code, we can deduce that some errors occurred.

The code is not guaranteed to detect d errors, because one codeword can be transformed into another by d bit-errors. □

Definition 1.5. We say a code can *correct* r errors if the following holds:

Whenever a codeword is transmitted across a channel capable of producing r bit-errors, one can deduce which codeword was sent using only the received word. Using our diagram from before, error correction consists of reading c' and deciding which codeword c must have been sent, and then decoding that codeword c to get the original message m.

$$\text{message } m \xrightarrow[\text{encoding}]{} \text{codeword } c \xrightarrow[\text{channel}]{} \text{received word } c' \xrightarrow[\text{decoding}]{} m' = m$$

Fact 1.5. If the distance of a code is d, then the code can correct exactly $\lfloor (d-1)/2 \rfloor$ errors.

Proof. Suppose C is a code of distance d, and a codeword c is sent across a channel capable of producing at most $\lfloor (d-1)/2 \rfloor$ bit-errors. Then $d(c, c') \leq \lfloor (d-1)/2 \rfloor$. We claim that c is the only codeword within distance $\lfloor (d-1)/2 \rfloor$ of c' - if there were another codeword, say c'', then

$$d(c, c'') \leq 2\lfloor (d-1)/2 \rfloor \leq d-1$$

by the triangle inequality for Hamming distance, which is a contradiction. Thus, we simply output the nearest codeword c to c' in Hamming distance. In fact, we only need to look nearby to c' for c. □

Example 3. The code in Example 2 has distance 2, so it can detect 1 error, but it can correct 0 errors.

Example 4. Consider the code that consists of codewords of length $3k$ formed by taking all strings of length k and repeating each character 3 times. For example, 101 corresponds to 111000111. The code, which is of block-length $3k$, has distance 3, and so it can correct 1 error.

1.3 How much information we can send: Rate

We measure how much information we can send essentially by the number of possible messages we can send. Suppose there is a collection of N messages. We can put these in one to one correspondence with a subset of the binary strings some length - we would need strings of length $\lceil \log_2 N \rceil$ to do this, since there are 2^k strings of length k. So, for convenience, we just think of $\log_2 N$ as the amount of information we can send, motivated by the fact that you need strings of length roughly $\log_2 N$ to have a correspondence between messages and strings.

Remark 1.6. For our discussion, the encoding itself doesn't matter. All that matters is the size of the code. This is because any two sets of the same cardinality can be put in 1-1 correspondence, so you can really just think that if you can find an error correcting code C, then you can send $|C|$ different messages without confusion, and hence you can send $\log_2 |C|$ bits of information.

It is easy to find a code with very good error correction properties - just repeat every bit of your message a million times. This will be able to correct about half a million errors, but it's also very wasteful. For this reason, we define a quantity that measures the information *per symbol* of the codeword.

Definition 1.6 (Rate)**.** The rate of a code C of block-length n is defined to be

$$rate(C) = \frac{\log_2 |C|}{n}.$$

That is, the amount of information the code can send over the block-length of the code.

2 The Hamming bound

A code that can correct errors cannot be too big. This makes sense, because it seems that a very large set of strings of length n must contain two nearby strings. After all, the maximum Hamming distance between two strings is n, and there isn't all the room in the world. This section formalizes that reasoning.

In the proof that a code of distance d can correct $(d-1)/2$ errors, we used the useful concept of "all nearby strings" of another string x.

Definition 2.1 (Hamming ball)**.** If x is a binary string of length n, then the Hamming ball of radius r around x, denoted $B_r(x)$, is all strings of Hamming distance at most r from x.

Fact 2.1.

$$|B_r(x)| = \sum_{i=0}^{r} \binom{n}{i}.$$

Fact 2.2. A code can correct r errors if and only if the Hamming balls of radius r around the codewords are pairwise disjoint.

The below bound is also sometimes called the sphere-packing bound or the volume bound.

Theorem 2.3 (Hamming bound)**.** *If a code C of block-length n can correct r errors, then*

$$|C| \leq \frac{2^n}{\sum_{i=0}^{r} \binom{n}{i}}.$$

Proof. The Hamming balls of radius r around the $|C|$ codewords are pairwise disjoint, and each has size $\sum_{i=0}^{r} \binom{n}{i}$, so $|C| \left(\sum_{i=0}^{r} \binom{n}{i}\right) \leq 2^n$. □

Algorithm 1 (Algorithm to decode a code)**.** Suppose C can correct r errors, and a codeword c is transmitted across a channel capable of producing at most r errors. If the received word is c', we can deduce c by looking at all elements of $B_r(c')$ and picking out the only element of the code.

Example 5. Suppose Alice encodes a message with the repetition code from Example 4 and sends it to Bob across a channel is capable of producing 1 error. Suppose Bob receives the word 001111. Then the only words it could have come from are

$$B_1(001111) = \{001111, 101111, 011111, 000111, 001011, 001101, 001110\}.$$

However, Bob knows Alice is using the repetition code, and the only element of the repetition code among $B_1(001111)$ is 000111. Thus, the original codeword sent was 000111, and so the message was 01.

2.1 Linear codes

Definition 2.2. If x is a number, then x mod 2 is the remainder of x when you divide by 2. If x is odd, then we write $x \pmod 2 = 1$ and if x is even then $x \pmod 2 = 0$.

Fact 2.4. $(x + y)$ mod $2 = (x \bmod 2 + y \bmod 2) \bmod 2$. All this is really saying is that if exactly one of x and y are odd, then $x + y$ is odd, and otherwise, $x + y$ is even.
Further, $xy \bmod 2 = (x \bmod 2)(y \bmod 2)$; this just reflects that the product of an even number with any number is even and the product of two odd numbers is odd.

In this section, it should be understood that all additions and multiplications of 1's and $0'$'s is performed mod2.

Example 6.

$$\begin{aligned} 1 + 0 = 0 + 1 = 1 \\ 1 + 1 = 0 + 0 = 0 \\ \text{and} \\ 1 \cdot 0 = 0 \cdot 1 = 0 \cdot 0 = 0 \\ 1 \cdot 1 = 1. \end{aligned}$$

Note that our "+" operation is the familiar XOR from computer science, and " $\cdot$ " is the AND operation.

Definition 2.3. We can also easily add vectors mod2. For example,

$$[0,1,0,0,1,1,0]+[1,1,0,0,1,1,1]=[1,0,0,0,0,0,1].$$

We can also do scalar multiplications.

$$0[0,1,0,0,1,1,0]=[0,0,0,0,0,0,0],\ \text{and}\ 1[0,1,0,0,1,1,0]=[0,1,0,0,1,1,0].$$

This means we can multiply matrices. For example,

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

$$=0\begin{bmatrix}1\\1\\0\end{bmatrix}+1\begin{bmatrix}1\\0\\1\end{bmatrix}+0\begin{bmatrix}0\\1\\1\end{bmatrix}+0\begin{bmatrix}1\\1\\1\end{bmatrix}+1\begin{bmatrix}1\\0\\0\end{bmatrix}+1\begin{bmatrix}0\\1\\0\end{bmatrix}+0\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$=\begin{bmatrix}1\\0\\1\end{bmatrix}+\begin{bmatrix}1\\0\\0\end{bmatrix}+\begin{bmatrix}0\\1\\0\end{bmatrix}=\begin{bmatrix}0\\1\\1\end{bmatrix}.$$

Definition 2.4. A *linear code* C is one formed by taking all products of a fixed matrix G with all vectors mod2. The matrix G is called a *generator matrix*. Usually we consider the multiplication from the left.

Example 7. Let $G=\begin{bmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{bmatrix}$. Then the code is all products yG, where y is any of the 2^4 row vectors of length 4. For example,

$$[0,0,1,1]G=[0,0,1,1]\begin{bmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{bmatrix}$$

$$=[0,0,1,0,0,1,1]+[0,0,0,1,1,1,1]$$

$$=[0,0,1,1,1,0,0]$$

is a codeword of C. $[0,0,0,0]G=[0,0,0,0,0,0,0]$ is also a codeword of G. There will be 2^4 codewords, because $y\to yG$ is one-to-one since y appears as the first 4 coordinates of yG.

Fact 2.5. The real reason such codes are called linear is that they have a nice property: If c_1 and c_2 are in a linear code C, then so is $c = c_1 + c_2$. This is because $c_1 = yG$ and $c_2 = xG$, so

$$c_1 + c_2 = yG + xG = (x + y)G \in C.$$

Define the *Hamming weight* of a vector v, denoted $w(v)$ to be the number of 1's it has. For example,

$$w([0, 0, 1, 0, 1]) = 2$$

. Then we have another key fact:

Fact 2.6 (Distance of a linear code)**.** *The distance of a linear code is the minimum Hamming weight of any nonzero codeword.*

Proof. Note that $d(c_1, c_2) = w(c_1 + c_2)$, where addition is mod2. (there will be a one exactly where the vectors c_1 and c_2 disagree.) □

Definition 2.5 (Standard form)**.** The generator matrix G for a linear code of block length n and message length k is in *standard form* if it looks like

$$\left[\begin{array}{c|c} I_k & P \end{array}\right].$$

where P is a $k \times n$ matrix.

Example 8. The matrix G from Example 7 is in standard form, because

$$G = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right].$$

Using this matrix G it is easy to check, using case-by-case analysis, that the minimum distance of the code it generates is 3.

Definition 2.6. If C is a linear code with generator matrix $G = \left[\begin{array}{c|c} I_k & P \end{array}\right]$, then the *parity-check matrix* of C is the $n - k \times n$ matrix

$$H = \left[\begin{array}{c|c} -P^T & I_{n-k} \end{array}\right].$$

For us, the minus part doesn't matter because $x \equiv -x \pmod 2$, but we write it because it would matter if we were working with symbols other than 0 and 1.

Fact 2.7. If the code C has generator matrix G and parity-check matrix H, then C is exactly the vectors

$$\{c : cH^T = 0\}.$$

That is, the code is exactly the vectors sent to 0 by the parity-check matrix.

Proof. If c is a codeword $cH^T = yGH^T$. But one can check that

$$GH^T = \left[\begin{array}{c|c} I_k & P \end{array}\right] \left[\begin{array}{c} -P \\ \hline I_{n-k} \end{array}\right] = -P + P = 0,$$

so $yGH^T = 0$. On the other hand, if $cH^T = 0$, then if c_1 is the vector that is the first k bits of c and c_2 is the second $n-k$ bits of c then $cH^T = -Pc_1 + c_2 = 0$, so $c_2 = Pc_1$. In other words, $c = c_1G$. Thus, c is in the code. □

2.2 The Hamming code

Definition 2.7. A Hamming$(2^k - 1, 2^k - k - 1)$ code is an encoding of strings of length $2^k - k - 1$ of block-length $2^k - 1$ that has distance 3 and corrects 1 error. We can define a Hamming$(2^k - 1, 2^k - k - 1)$ code by the parity check matrix as follows:

The parity-check matrix of a Hamming$(2^k - 1, 2^k - k - 1)$ code is any whose columns are all possible vectors of length $n - k$. We typically order them so that the last part of the matrix is the identity, which puts things in standard form.

Example 9. A parity check matrix H of a Hamming(7,4) code is ($k = 3$ in this case)

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

because as you can see, all strings appear and the last part of the matrix is the identity. To get the generator matrix, apply the definition of the parity check matrix to obtain

$$G = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right].$$

Fact 2.8. For $k < n$, a Hamming$(2^k - 1, 2^k - k - 1)$ code has distance at least 3. Thus, it can correct 1 error.

Proof. The distance of a linear code is the least Hamming weight of any nonzero codeword. However, we know that the codewords are the ones passing the parity-check, that is

$$cH^T = 0.$$

However, no row of H^T is zero, so no codeword of Hamming weight 1 can pass the parity-check. Also, no two rows are the same, so no codeword of Hamming weight 2 can pass the parity-check. Thus, all codewords are of Hamming weight at least 3, the code has distance at least 3! □

Algorithm 2 (Decoding the Hamming Code)**.** The following method works to decode a linear code in standard form:

Input: A transmitted message c' with at most 1 error, that is, a message c at Hamming distance 1 from some codeword c of a Hamming$(2^k - 1, 2^k - k - 1)$ code.

Output: The original message m.

- If c' has at most 1 errors, then $x = c' + e$ where e is a string of Hamming weight at most 1.
- Compute $c'H^T$. However, by Fact 2.1.4,

$$c'H^T = (c' + e)H^T = eH^T.$$

- eH^T is just the row of H^T where the error occurred, which if transposed, will be the column of H corresponding to the bit that erred. Deduce e by looking at where this column came from.
- Set $c = c' - e$.
- Since the generator matrix is in standard form, the first k bits of c are the message m.

Example 10. Let's use the parity check H for a Hamming(7,4) code to decode some words. If a received word is $c' = [1, 1, 1, 1, 0, 1, 0]$, then we compute $c'H^T$ to obtain

$$[1, 1, 1, 1, 0, 1, 0] \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [1, 0, 1].$$

This corresponds to the column

$$\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

of H, which is the 2^{nd} column. Thus, $e = [0, 1, 0, 0, 0, 0, 0]$, and so

$$c = [1, 1, 1, 1, 0, 1, 0] - e = [1, 0, 1, 1, 0, 1, 0],$$

and so the message was $[1, 0, 1, 1]$.

Theorem 2.9. *The rate of a Hamming*$(2^k - 1, 2^k - k - 1)$ *code is, of course,*

$$\frac{2^k - k - 1}{2^k - 1} \to 1$$

as $k \to \infty$*. Further, Hamming codes are perfect codes - they exactly match the Hamming bound.*

Proof. Since the block-length of a Hamming$(2^k - 1, 2^k - k - 1)$ code is $2^k - 1$, the Hamming bound says the number of codewords is at most

$$\frac{2^{2^k - 1}}{1 + (2^k - 1)} = 2^{2^k - 1 - k}.$$

However, this is exactly the number of codewords of Hamming$(2^k - 1, 2^k - k - 1)$, because it encodes all binary strings of length $2^k - k - 1$. □

Remark 2.10. Compare this to the repetition code which had rate **??**3.

We can extend this algorithm for linear codes that correct more errors.

Algorithm 3 (Syndrome decoding)**.** The following method works to decode a linear code in standard form:

Input: A transmitted message x with at most $(d-1)/2$ errors, that is, a message x at Hamming distance $\leq (d-1)/2$ from some codeword $y^T G$.

Output: The original message y^T.

- If x has at most $(d-1)/2$ errors, then $x = y^T G + e$ where e is a string of Hamming weight at most $(d-1)/2$.
- Compute Hx^T. By construction of the parity-check matrix H,

$$Hx^T = HG^T y + He = He.$$

- For over all vectors e' has Hamming weight at most $(d-1)/2$, find the unique one where $He' = He$. Then $e' = e$, so you have found e.
- $y^T G = x - e$, so output the first k bits of $x - e$, which will be equal to y.

The above algorithm is correct.

Proof. □

Example 11.

2.3 Better Examples

Definition 2.8 (Rate)**.**

Hamming code corrects a single error.

Definition 2.9 (Mod 2 Arithmetic)**.**

Definition 2.10 (Generator matrix)**.**

Definition 2.11 (Parity-check matrix)**.**

3 Upper Bounds on the sizes of Error correcting codes

3.1 The Hamming Bound

Theorem 3.1 (Maximum rate for detecting multiple errors)**.**

Theorem 3.2 (Maximum rate for detecting a single error)**.**

Definition 3.1 (Perfect codes)**.**

Definition 3.2 (For fun: Golay Code)**.**

4 Lower Bounds on the sizes of Error correcting codes

4.1 Gilbert-Varshamov nonconstructive lower bound

5 Random Errors

Definition 5.1 (Binary symmetric channel)**.**

Probability of getting certain number of errors.

5.1 Proof sketch for Shannon, random linear codes.

Theorem 5.1 (Shannon's Theorem)**.**

What are they? Constructively what can be done?