LINEAR ERROR-CORRECTING CODES

NANDANA MADHUKARA

1. Introduction

If humans look at the phrases

Helo evryne!

or

Plse kep readng ths papr

they can easily decipher what they say, and this is why companies can get away with replacing letters in their name with symbols that don't quite match the letter and why texting is so easy. However, detecting these "errors" is hard for a computer that doesn't have human intuition, and this is where error-correcting codes come in [Bay98].

When data is transmitted through channels, it is very susceptible to having errors when it reaches the receiver. This is clearly a problem, so having a machine be able to detect and correct errors is very useful. One obvious way of doing this is by just repeating the message, so that if there is an error, we can easily detect it and correct it with the copy. However, this has two problems: it takes up a lot of space, and there could be an error in the original and the copy. To fix the second problem, we can keep making copies, and whenever there is a discrepancy, we take the data that is consistent in most of the copies, but as mentioned in the first problem, this takes up a lot of space. Therefore, the interesting question is to ask how much data can be corrected while giving up as little space as possible for redundancy.

One of the earliest examples of error-correcting codes were Hamming codes. The story goes that Richard Hamming was working for Bell Labs, and he was doing work on an expensive punch-card computer to which he had limited access. However, his programs kept failing since a bit would get swapped or changed, so he got so frustrated that he created the first error-correcting code. There has been a lot of work done after this by many mathematicians, such as the Reed-Solomon code which is what CDs and DVDs use. Furthermore, this field has been a surprisingly rich source of mathematics, such as sphere packing [LS93] and Galois Extensions [you20].

2. Basics of Coding Theory

The setup is that Alice wants to send a message to Bob over a noisy channel that could introduce some errors. Alice splits up her message into *message blocks* which consist of symbols in some alphabet A. Each message word $\mathbf{m}$ is converted into a *code word* $\mathbf{c}$ in an *alphabet* A.

Definition 2.1. A *code word* $\mathbf{c}$ of length n over A is a subset $\mathbf{c} \subseteq A^n$. A *code* C of word length n over A is the set of all code words of length n.

Date: March 18, 2023.

Notice that we can make A a field by defining operators $+$ and $\times$ such that they follow the field axioms. With this, we can consider A^n to be a vector space by defining some scalar multiplication with A and term-wise vector addition.

Definition 2.2. Let $\mathbf{c} \subseteq A^n$. An *encoding function* is the injective map $E : [|\mathbf{c}|] \to A^n$.

Definition 2.3. Let $\mathbf{c} \subseteq A^n$. A *decoding function* is the map $D : A^n \to [|\mathbf{c}|]$.

Over the noisy channel, a code word $\mathbf{c}$ could have an error $\mathbf{e}$ and be incorrectly converted to another word d and the job of the decoding function is to

1. detect $\mathbf{d}$ is not a code word and identify the error $\mathbf{e}$ is made,
2. fix the error and obtain $\mathbf{c}$, and
3. go from $\mathbf{c}$ to $\mathbf{m}$.

We can define some other properties of codes:

Definition 2.4. For two code words $\mathbf{c}, \mathbf{c}' \in A^n$, the *Hamming distance* denoted by $\Delta(\mathbf{c}, \mathbf{c}')$ is number of positions $\mathbf{c}$ and $\mathbf{c}'$ differ in.

Definition 2.5. The *minimum distance* for a code $C \subseteq A^n$ denoted by $\Delta(C)$ is the smallest Hamming distance between any two code words in C. In other words

$$\Delta(C) = \{\min(\mathbf{c}, \mathbf{c}') : \mathbf{c}, \mathbf{c}' \in C\}.$$

If all messages were not encoded, then C is just all possible words so $\Delta(C) = 1$. This is not ideal since it is very hard to distinguish an error word $\mathbf{d}$ from an actual code word $\mathbf{c}$. Therefore, we want to maximize the minimum distance so that we can easily distinguish code words $\mathbf{c}$ and error words $\mathbf{d}$.

One example of this is the famous Hamming(7,4) code C which encodes a message of length n as a message of length 7 and has $\Delta(C) = 3$. This means that if there is exactly one error in any code word $\mathbf{c} \in C$, we can be sure that the error word $\mathbf{d} \notin C$ since $\Delta(\mathbf{c}, \mathbf{d}) = 1$ but the minimum distance is 3. Therefore, a single error is easy to detect. Furthermore, if there is only a single error, we can match it to the nearest code word. More specifically, if match $\mathbf{d}$ to the $\mathbf{c} \in C$ such that $\Delta(\mathbf{c}, \mathbf{d}) < 3/2$. In general, this *nearest-neighbor decoding* is how we decode in any error correcing code but in this case, it is only unambiguous when there is a single error. If there are two errors or $\Delta(\mathbf{c}, \mathbf{d})$, the nearest-neighbor decoding does not work because in Figure 1, we can see that the nearest code word for $\mathbf{d}$ is $\mathbf{c}'$ and not $\mathbf{c}$. This means that that Hamming(7,4) can detect two errors but not correct them.

Definition 2.6. A code C is *t-error-correcting* if the code can correct at most t errors.

3. Basics of Linear Codes

Now let us delve a little more into the math and talk about how we can mathematically encode and decode. When we start discussing linear codes it is useful to consider $F_q = \{0, 1, ..., q-1\}$ instead of A (assuming $|A| = q$) since they are isomorphic. We also make q prime such that F_q is a finite field. Therefore, words are just row vectors $\mathbf{w}$ in the vector space F_q^n.

Definition 3.1. A *linear code* C of word length n over F_q is a vector subspace of F_q^n. If the dimension of C is k, then the code C is called an $[n, k]$.

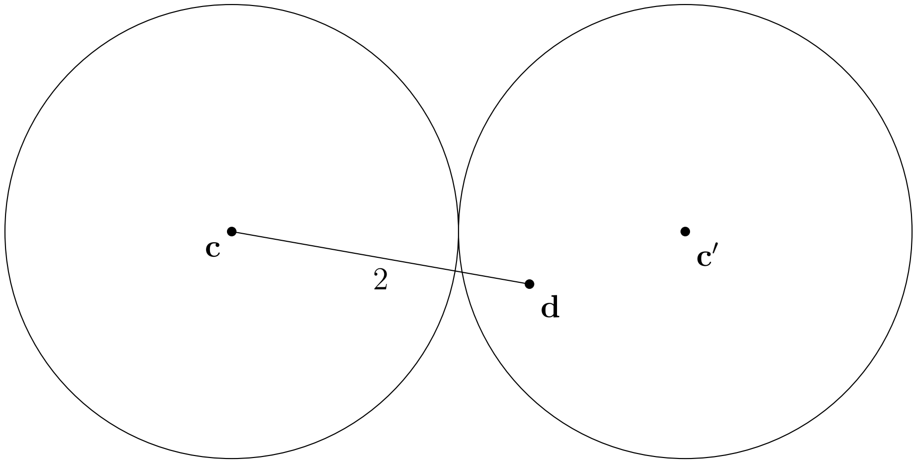

Figure 1. The case where nearest-neighbor decoding does not work

A simple result we can see is that if $dim(C) = k$, then $|C| = q^k$. This is because, if $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k\}$ is the basis of C, then each code word is just $\sum_{i=0}^{k} \lambda_i \mathbf{b}_i$. We know that the λ_i's can take on q values so the total number of code words is q^k.

Definition 3.2. The *weight* of a word $\mathbf{w} \in C$, denoted by $w(\mathbf{w})$, is the number of nonzero letters $\mathbf{w}$ has.

A very useful fact from this is that $\Delta(\mathbf{w}_1, \mathbf{w}_2) = w(\mathbf{w}_1 - \mathbf{w}_2)$ and this works because a letter in $\mathbf{w}_1 - \mathbf{w}_2$ is only zero when letters are the same so $w(\mathbf{w}_1 - \mathbf{w}_2)$ will just count the number of spots with differing letters. We can use this to prove the following

Lemma 3.3. *If C be a linear code, then smallest weight of any nonzero code word, denoted by w_{min}, is $\Delta(C)$.*

Proof. Let $\mathbf{c}_1$ be the nonzero code word with the smallest weight. This means that

$$w_{\min} = w(\mathbf{c}_1) = w(\mathbf{c}_1 - \mathbf{0} = \Delta(\mathbf{c}_1, \mathbf{0}) \geq \Delta(C).$$

Now, let $\mathbf{c}_2$ and $\mathbf{c}_3$ be two code words such that $\Delta(\mathbf{c}_2, \mathbf{c}_3) = \Delta(C)$. This means that

$$\Delta(C) = \Delta(\mathbf{c}_2, \mathbf{c}_3) = w(\mathbf{c}_2 - \mathbf{c}_3) \geq w_{\min}$$

since $\mathbf{c}_1 - \mathbf{c}_2$ is a code word. Putting this all together means that $\Delta(C) = w_{\min}$. ■

4. Encoding and Decoding

Now we are ready to talk about encoding and decoding messages. Let the message $\mathbf{m}$ be k letters long so $\mathbf{m} \in F_q^k$. The way we encode $\mathbf{m}$ as a code word $\mathbf{c}$ is with a matrix:

Definition 4.1. A *generator matrix G* over F_q for an $[n, k]$ code C is a $k \times n$ matrix whose rows are the code words of any basis of C.

Example. Consider the following generator matrix that defines a [7, 4] code over F_2

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The rows are clearly linearly independent so they can form a basis for C. Now we can create a code word by multiplying a message word $\mathbf{m}$ by G:

$$\mathbf{c} = \mathbf{m}G = \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Notice that we are doing modular addition so $1 + 1 = 0$. Additionally, G is said to be in *standard form* since it is in the form $(I_k \mid A)$ and this is very useful since the first k letters is the message we are encoding and we are only adding on $n - k$ more letters. Therefore, this saves a lot of time for the decoders since we can go from a code word to its message word very easily

After encoding, it is useful to find the minimum distance of the code since we can deduce the number of errors the code can correct as we saw in the Hamming(7, 4) code and which we will see later. We can do this with a parity check matrix:

Definition 4.2. A *parity check matrix* for a linear $[n, k]$ code C is a $(n - k) \times n$ matrix H such that

(1) all of the rows are linearly independent and
(2) $\mathbf{c} \in C$ if and only if $\mathbf{c}H^T = \mathbf{0}$.

We can actually calculate the parity check matrix of a linear code with the following theorem:

Theorem 4.3. *Let C be a linear $[n, k]$ code with a standard form generator $G = (-I_k \mid A)$. The parity check matrix for C is $H = (-A^T \mid I_{n-k})$.*

Proof. Let

$$G = \begin{pmatrix} 1 & \cdots & 0 & a_{11} & \cdots & a_{1,n-k} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & a_{k1} & \cdots & a_{k,n-k} \end{pmatrix}$$

be the standard form generator of C and let

$$H = \begin{pmatrix} -a_{11} & \cdots & -a_{k1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -a_{1,n-k} & \cdots & -a_{k,n-k} & 0 & \cdots & 1 \end{pmatrix}.$$

Now H indeed is a $(n - k) \times n$ matrix and its rows are linearly independent. All that is left to check is $\mathbf{c} \in C \iff \mathbf{c}H^T = \mathbf{0}$.

First if $\mathbf{c} \in C$, then it can be written as $\mathbf{c} = \sum_{i=0}^{k} m_i \mathbf{b}_i$ where the $\mathbf{b}_i$'s are rows of G and the m_i are the letters of the message word. This means we have

$$\mathbf{c}H^T = \left(\sum_{i=0}^{k} m_i \mathbf{b}_i \right) H^T = m_i \sum_{i=0}^{k} \mathbf{b}_i H^T.$$

Notice that the expression inside the sum is just a vector of inner product of row i of G and row i of H. Now this inner product is just $-a_{ij} + a_{ij} = 0$ (where we have omitted the 0's) so

$$\mathbf{c}H^T = m_i \sum_{i=0}^{k} \mathbf{b}_i H^T = m_i \sum_{i=0}^{k} \mathbf{0} = \mathbf{0}.$$

Next if $cH^T = \mathbf{0}$, let $\mathbf{c} = \sum_{i=0}^{k} m_i \mathbf{b}_i$ where the $\mathbf{b}_i$ are some n dimension vectors and the m_i's are some scalars. We have

$$\mathbf{c}H^T = m_i \sum_{i=0}^{k} \mathbf{b}_i H^T = \mathbf{0} \implies \mathbf{b}_i H^T = \mathbf{0}$$

for all i. We can see that the only way this can happen is if the b_i's are rows of G so we are done. ■

Therefore, every linear code as a parity check matrix which can be found using this theorem.

Example. From the previous example, we would have

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

after using our formula. Since we are working in binary of F_2, we just have $-1 = 1$ which is why all the entries are positive.

Now we can find the minimum distance with the following theorem:

Theorem 4.4. *The minimum distance $\Delta(C)$ of a linear code C is the size of the smallest linearly dependent set of columns of its parity check matrix H.*

Proof. Let $\mathbf{c} \in C$ be a code word with weight $w_{\min}$ which is $\Delta(C)$ by Lemma 3.3. This means that there are only $\Delta(C)$ nonzero letters in $\mathbf{c}$. Now let $H = (\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_n)$ be the parity check matrix so by definition,

$$\mathbf{c}H^T = c_1\mathbf{h}_1 + c_2\mathbf{h}_2 + \cdots + c_n\mathbf{h}_n = \mathbf{0}.$$

However, this means that the set $\{\mathbf{h}_i \mid c_i \neq 0\}$ must be linearly independent since we can think of the c_i's as weights. Additionally, the size of this set is $\Delta(C)$ so all that is left to do is to prove that this is indeed the smallest set.

Let $\{\mathbf{h}_{\sigma(1)}, ..., \mathbf{h}_{\sigma(t)}\}$ be a set of t linearly independent columns. By the definition of linear dependence there must exist λ_i's such that

$$\lambda_1\mathbf{h}_{\sigma(1)} + \cdots + \lambda_2\mathbf{h}_{\sigma(2)} = \mathbf{0}.$$

Now let $\mathbf{c}'$ be the word with letters λ_i and 0's everywhere else. We know that $\mathbf{c}'H^T = \mathbf{0}$ so $\mathbf{c}'$ is a code word. My definition, we know that the weight of $\mathbf{c}'$ is less than or equal to t which means that $t \geq w_{\min} = \Delta(C)$ and we are done. ■

Now we can start discussing decoding. Unlike encoding, decoding is not as simple as multiply by a matrix since recall the way we decode is matching error word with the closest code word so if $\mathbf{c}$ is the code word (which we want to find) that is erroneously outputted as $\mathbf{d}$, we know that $\Delta(\mathbf{d}, \mathbf{c}) \leq \Delta(\mathbf{d}, \mathbf{c}')$ for all code words $\mathbf{c}' \neq \mathbf{c}$ in C. Of course, we can find $\mathbf{d}$ just by looking through all possible code words in C and finding the closest one but just considering the sheer size of the set of possibilities should discourage us from doing this. Additionally, this is not only slow but we must store the whole code space C which is a storage issue meaning that we must find a better way.

There is indeed a better way by using cosets from abstract algebra. First let $\mathbf{e}$ be the error such that $\mathbf{d} = \mathbf{c} + \mathbf{e}$ so $\mathbf{c} = \mathbf{d} - \mathbf{e}$. This means that $\mathbf{d} + \mathbf{c}_1 = \mathbf{e} + \mathbf{c}_2$ if $\mathbf{c}_2 - \mathbf{c}_1 = \mathbf{c}$ for $\mathbf{c}_1, \mathbf{c}_2 \in C$. Therefore this means that

$$\mathbf{d} + C = \mathbf{e} + C$$

where

$$\mathbf{a} + C = \{\mathbf{a} + \mathbf{c} : \mathbf{c} \in C\}$$

is a left coset for some $\mathbf{a} \in F_q^n$. Now since the $\mathbf{d}$ and $\mathbf{e}$ left cosets are equal, they must be in the same coset since F_q^n is partitioned into a finite number of disjoint cosets. Now since we are given $\mathbf{d}$, we are looking for the right $\mathbf{e}$ that is in the same coset. However, we know that we want to have $\Delta(\mathbf{d}, \mathbf{c}) \leq \Delta(\mathbf{d}, \mathbf{c}')$ so we want to minimize $w(\mathbf{e})$ or the weight of $\mathbf{e}$. Therefore, we can go through $\mathbf{d} + C$ and find the word with the minimum weight. After we find the error $\mathbf{e}$, we can just decode as $\mathbf{c} = \mathbf{d} - \mathbf{e}$ and we can drop the last $n - k$ letters to get the message word $\mathbf{m}$ if C has a standard form generator.

Notice that this is better than our original way but we still need to store the whole word space F_q^n and know the cosets. Then we need to browse over this word space to find $\mathbf{d}$ and its coset $\mathbf{d} + C$ which will both take some time. Fortunately, we can actually find a better way than both of these. This can be done using the parity check matrix.

Definition 4.5. Let $\mathbf{w}$ be any word in F_q^n and let H be a parity check matrix for a linear code. The *syndrome* of $\mathbf{w}$ is

$$s(\mathbf{w}) = \mathbf{w}H^T$$

With this definition, we have the following result:

Lemma 4.6. *The words $\mathbf{v}$ and $\mathbf{w}$ belong to the same coset if and only if $s(\mathbf{v}) = s(\mathbf{w})$*

Proof. First let $\mathbf{v}$ and $\mathbf{w}$ be words that belong to the same coset. This means that $\mathbf{v} + C = \mathbf{w} + C$ so $\mathbf{v} - \mathbf{w} \in C$. By the definition of a parity check matrix, we have

$$(\mathbf{v} - \mathbf{w})H^T = \mathbf{0} \implies \mathbf{v}H^T = \mathbf{w}H^T \implies s(\mathbf{v}) = s(\mathbf{w}).$$

These implications also go the other way so

$$s(\mathbf{v}) = s(\mathbf{w}) \implies (\mathbf{v} - \mathbf{w})H^T = \mathbf{0}$$

which means that $\mathbf{v} - \mathbf{w} \in C$ and the only way this can happen is if $\mathbf{v}$ and $\mathbf{w}$ are in the same coset. ■

Now this can spark some ideas in a better decoding algorithm. Since we know that $\mathbf{d}$ and $\mathbf{e}$ are in the same coset, they must have the same syndrome. Therefore, can find $\mathbf{e}$ by finding the error that has the same syndrome as $\mathbf{d}$. This means our algorithm to decode a t-error-correcting code $[n, k]$ code C with a standard form generator and parity check matrix H is

(1) Create a table with the correctable errors $\mathbf{e}_i$ (so errors with weight less than or equal to t) in one column and $s(\mathbf{e}_i)$ in another column.
(2) Identify the error $\mathbf{e}$ that has the same syndrome as $\mathbf{d}$. If none exists, then $\mathbf{d}$ is undecodable by C.
(3) Decode to the code word $\mathbf{c} = \mathbf{d} - \mathbf{e}$ and drop the last $n - k$ letters to get the message word $\mathbf{m}$

Now the only way this will not work is if there are two correctable errors that have the same syndrome as $\mathbf{d}$ but we will discuss this in the next section.

5. Correcting t errors

So far, we have shown how we can encode messages to form a code C and how we can decode them. We also saw how we can find $\Delta(C)$ and in the Hamming(7, 4) code, we saw that the number of errors a code can correct depends on its minimum distance so the question is: what must $\Delta(C)$ be such that C is t-error-correcting? Before this, we must first further describe the nature of the Hamming Distance.

Lemma 5.1. *The Hamming distance is a metric on A^n. That is, the function $\Delta : A^n \times A^n \to F$ satisfies for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in A^n$*

(1) $\Delta(\mathbf{x}, \mathbf{y}) \geq 0$ *and* $\Delta(\mathbf{x}, \mathbf{y}) = 0 \implies \mathbf{x} = \mathbf{y}$
(2) $\Delta(\mathbf{x}, \mathbf{y}) = \Delta(\mathbf{y}, \mathbf{x})$
(3) $\Delta(\mathbf{x}, \mathbf{z}) \leq \Delta(\mathbf{x}, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{z})$.

Proof. By the definition of $\Delta(\mathbf{x}, \mathbf{y})$, we can trivially see that (1) and (2) are true so we are left with (3). Let $\mathbf{x} = (x_1, x_2, ..., x_n)$, $\mathbf{y} = (y_1, y_2, ..., y_n)$, $\mathbf{z} = (z_1, z_2, ..., z_n)$. Now we go through the words and compare x_i, y_i, and z_i and we have two cases.

Case 1: $x_i = z_i$. This means that the ith letters are the same so the ith letter contributes contributes 0 to $\Delta(\mathbf{x}, \mathbf{z})$. However, the ith letter contributes a non negative number to $\Delta(\mathbf{x}, \mathbf{y})$ and $\Delta(\mathbf{y}, \mathbf{z})$ so the ith letter never contribute more to $\Delta(x, z)$ than $\Delta(\mathbf{x}, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{z})$ in this case

Case 2: $x_i \neq z_i$. This means that the ith letters are different so the ith letter contributes 1 to $\Delta(\mathbf{x}, \mathbf{z})$. The ith letter contributes a non negative number to $\Delta(\mathbf{x}, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{z})$ but never 0. This is because y_i cannot be the same as x_i and z_i since $x_i \neq z_i$. Therefore the ith letter never contribute more to $\Delta(x, z)$ than $\Delta(\mathbf{x}, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{z})$ in this case so we are done. ■

Theorem 5.2. *A code C is t-error-correcting if and only if $\Delta(C) \geq 2t + 1$.*

Proof. We first prove the forward direction so we assume a code C is t-error-correcting and we must prove that $\Delta(C) \geq 2t + 1$. We can equivalently prove the contrapostive: If $\Delta(C) < 2t + 1$, then C is not t-error-correcting. In other words, we must find an error that the code cannot detect.

Let $\mathbf{c}$ and $\mathbf{c}'$ be code words such that $\Delta(\mathbf{c}, \mathbf{c}') = \Delta(C)$. Now we introduce some errors to $\mathbf{c}$. If $\Delta(C)$ is even, out of the $\Delta(C)$ letters that are different in $\mathbf{c}$ and $\mathbf{c}'$, we change $\Delta(C)/2$ of them to be the corresponding letters in $\mathbf{c}'$. If $\Delta(C)$ is odd, we change $(\Delta(C) + 1)/2$ of the differing letters to be the corresponding letters in $\mathbf{c}'$. Like usual, we call this error word $\mathbf{d}$. Notice that since $\Delta(C) < 2t + 1$, the number of errors we introduce is always at most t.

In the even case, we know that $\Delta(\mathbf{c}, \mathbf{d}) = \Delta(C)/2$ by definition but because of the way we defined the error, we have $\Delta(\mathbf{c}', \mathbf{d}) = \Delta(C)/2$ so $\mathbf{d}$ does not have a unique closest neighbor so the decoder does not know to pick $\mathbf{c}$ or $\mathbf{c}'$. In the odd case we have $\Delta(\mathbf{c}, \mathbf{d}) = (\Delta(C) + 1)/2$ so

$$\Delta(\mathbf{c}', \mathbf{d}) = \Delta(C) - \frac{\Delta(C) + 1}{2} = \frac{\Delta(C) - 1}{2}$$

so the decoder would make a mistake and return $\mathbf{c}'$ rather than $\mathbf{c}$.

For the reverse direction, let $\mathbf{c} \in C$ be a code word and let $\mathbf{d}$ be the error word with at most t errors so $\Delta(\mathbf{c}, \mathbf{d}) \leq t$. We assume $\Delta(C) \geq 2t + 1$ so we have

$$2t + 1 \leq \Delta(C) \leq \Delta(\mathbf{c}, \mathbf{c}')$$

for some other code word $\mathbf{c}' \in C$. By the triangle inequality, we have

$$2t + 1 \leq \Delta(\mathbf{c}, \mathbf{c}') \leq \Delta(\mathbf{c}, \mathbf{d}) + \Delta(\mathbf{d}, \mathbf{c}') \leq t + \Delta(\mathbf{d}, \mathbf{c}').$$

This can be rewrite this as

$$\Delta(\mathbf{d},\mathbf{c}') \geq t+1 \geq t \geq \Delta(\mathbf{d},\mathbf{c})$$

so the decoder would correctly match $\mathbf{d}$ with $\mathbf{c}$ since $\mathbf{d}$ is closer to $\mathbf{c}$ than any other code word. ■

Now recall that in the previous section, our decoding algorithm wouldn't work if two errors had the same syndrome as $\mathbf{d}$ but now we can show that this cannot happen.

Corollary 5.3. *Let C be a t-error-correcting code and let $\mathbf{e}_1$ and $\mathbf{e}_2$ be correctable errors. Then $s(\mathbf{e}_1) \neq s(\mathbf{e}_2)$.*

Proof. For the sake of contradiction, assume that $s(\mathbf{e}_1) = s(\mathbf{e}_2)$. By Lemma 4.6, this means that $\mathbf{e}_1$ and $\mathbf{e}_2$ are in the same coset so $\mathbf{e}_1 - \mathbf{e}_2 \in C$. We know that $w(\mathbf{e}_1), w(\mathbf{e}_2) \leq t$ so $\mathbf{e}_1 - \mathbf{e}_2$ can have no more than $2t$ nonzero letters. This means that

$$w(\mathbf{e}_1 - \mathbf{e}_2) \leq 2t < 2t+1.$$

By Theorem 5.2, we have

$$w(\mathbf{e}_1 - \mathbf{e}_2) < 2t+1 \leq \Delta(C)$$

so $w(\mathbf{e}_1 - \mathbf{e}_2) < \Delta(C)$. This is however a contradiction since $w_{\min} = \Delta(C)$ so we are done. ■

Since correctable errors have different syndromes, there must only be one correctable error that has the same syndrome as $\mathbf{d}$.

References

[Bay98] D. J. Baylis. *Error correcting codes: A mathematical introduction.* Routledge, 1998.

[LS93] John Leech and N. J. Sloane. Sphere packing and error-correcting codes. *Grundlehren der mathematischen Wissenschaften*, page 136–156, 1993.

[you20] *How to send a self-correcting message (Hamming codes).* YouTube, Sep 2020.