CS294-226: Advances in Error-Correcting Codes **UC Berkeley, Fall 2022**
Lecture #1: Introduction to Coding Theory August 24, 2022
Lecturer: Venkatesan Guruswami Scribe: Yuval Michael

1 Overview of Error Correcting Codes

Often times, when interacting with the environment around us, we assume that the data we are receiving is ideal. However, that is far from the case, especially when it comes to digital information. Information that we send and receive often can get corrupted and construed from its original source. To overcome this issue we create methods to allow the recipient to reconstruct the original message; this can be as simple as asking someone to repeat a message again, all the way to developing special data structures and encoding methods to pack messages with extra information.

For digital communications there have been developments with implementing Error Correcting Codes (ECCs) to ensure communications are not compromised. The goal of ECCs is to provide a judiciously designed noise-resistant form of coded data. This coded data, is some form of data which not only contains the original data but also additional information to allow reconstruction of the message if it experiences noise.

The redundancy built in to ECCs allows for the code to preform recovery from various effect effects of noise. So noise may corrupt some part of the data, but due to the additional information that came with the encoding of the original message, the original message can be recovered to some limit.

The limit of how much noise a message can experience and still have the original message be distinguishable is a fine line which ECCs try to balance. Effective ECCs seek to create a encoding which grantees the most recovery of data all while having minimal additional information. Additionally, ECCs seek to have a efficient method of encoding and decoding information. This is all to say that ECCs try to optimize effectiveness and minimize overhead.

The following chart shows the process in which error-correcting codes are implemented within data communication.

$$\begin{tikzcd}
\boxed{\text{Input}} \arrow[r, "\text{Coding}"] & \boxed{\text{Coded Data}} \arrow[r, "\text{Noise}"] & \boxed{\text{Noisy Data}} \arrow[r, "\text{Correction}"] & \boxed{\text{Original Data}}
\end{tikzcd}$$

1.1 Basic definitions

Definition 1 (Error-Correcting Code)**.** An Error-correcting code over an **alphabet** Σ is defined by a pair of maps $(\mathrm{Enc}, \mathrm{Dec})$, where $\mathrm{Enc} : \Sigma^k \to \Sigma^n$ is an injective map from k symbols to n symbols of coded form, and a decoding map $\mathrm{Dec} : \Sigma^n \to \Sigma^k$ from n symbols back to k symbols.

The collection of strings Σ^k represents the **message space**, and the elements in Σ^k are the **messages**.

The collection of strings $C := \mathrm{Enc}(\Sigma^k) \subseteq \Sigma^n$ represents the collection of codewords of our error-correcting code (i.e. encoded data of n-bit strings after applying the ECC). n is referred to as the

block length of a code, and k is the **dimension** of the code. Note that $|C| = |\Sigma|^k$ as Enc is an injective map.

Throughout most of the course, we will be dealing almost exclusively with C rather than the pair (Enc, Dec).

With this knowledge in hand we can rewrite the above diagram as:

$$\begin{tikzcd}
m & c & \tilde{c} & m \\
\textbf{Input} \arrow[r, "\text{Enc}"] & \textbf{Coded Data} \arrow[r, squiggly, "\text{Noise}"] & \textbf{Noisy Data} \arrow[r, "\text{Dec}"] & \textbf{Original Data} \\
\Sigma^k & \Sigma^n & \Sigma^n & \Sigma^k
\end{tikzcd}$$

Notation 1. A code C can be defined for the following notation $[n, k, d]_q$, where the length of a codeword c is n, the dimension of the code is k, the minimum distance of the codes is d, and the alphabet size is $q = |\Sigma|$.

1.2 Types of noise

ECCs are built to correct errors that arise when noise acts on our data while transferring it. The way in which noise acts on data is hard to predict and simulate. As such, we have developed different models to represent how noise will affect our data.

Hamming's Worst Case: This noise model says at most this information will experience some number of errors bounded by some constant β. Throughout the whole of this note as well as most of the course we will be dealing with noise according to the hamming worst case noise model.

Shannon's Probabilistic Stochastic: This noise model says that for each bit they will experience noise with some probability α. Unlike Hamming's noise model with this kind of model we cannot grantee to always recover the transmitted code word.

Alike with types of noise models we also different kinds of noise which can act on our data. These types of noise can occur on a bit level or a packet level. This note will focus solely on the bit level.

Erasure: In the erasure model of noise some symbols are erased and the rest are intact. The location of the erasures are known. For example,

$$101101 \to 10?1?1$$

Error: In the Error model of noise some symbols are modified in value, the location of the modification is unknown. For example,

$$101101 \to 101100$$

Deletion: In the deletion model of noise some symbols are erased and the rest are intact. The location of the erasure are unknown. For example,

$$101101 \to 11101$$

2 Single Bit Erasure

Given a code $C \subseteq \{0,1\}^n$, is there a way to make an encoding method such that if one code word c_i experiences an erasure the original message can still be retrieved in whole?

Construction 1 (Parity)**.** One method to corrects this possible single bit erasure is with the addition of a parity bit, through the following encoding scheme.

$$(m_1, m_2, m_3, \dots, m_k) \to (m_1, m_2, m_3, \dots, m_n, m_1 \oplus m_2 \oplus m_3 \oplus \dots \oplus m_k)$$

And so the codewords are going to be $C_{par} = \{(m_1, \dots, m_{n-1}, m_1 \oplus \dots \oplus m_{n-1}) \mid m_i \in \{0,1\}\}$.

Construction 2 (Duplication)**.** Another method to correct this single bit erasure is with a duplication of each bit.

$$(m_1, m_2, \dots, m_k) \to (m_1, m_1, m_2, m_2, \dots, , m_{k-1}, m_{k-1}, m_k, m_k)$$

And so the codewords are going to be $C_{dupe} = \{(m_1, m_1, m_2, m_2, \dots, m_{n/2-1}, m_{n/2-1}, m_{n/2}, m_{n/2} \mid m_i \in \{0,1\}\}$.

While both of these codes successfully give a scheme which can be used to resolve one bit erasure. One is more effective than the other. Both C_{par} and C_{dupe} have algorithmically fast encoding (of running time $O(n)$). However, the two codes greatly differ in size. $|C_{par}| = 2^{n-1}$ and thus contains only 1 redundant bit of information. On the other hand, $|C_{dupe}| = 2^{n/2}$, and so it contains $\frac{n}{2}$ redundant bits. This lesser redundancy is important as it directly relates to the idea of rate.

Definition 2 (Rate)**.** The rate of a code $C \subseteq \Sigma^n$, denoted $R(C)$ is defined by

$$R(C) = \frac{\log |C|}{n \log |\Sigma|}$$

$R(C)$ represents the amount of non-redundant information per bit in codewords of C.

Using the definition earlier, we can see that $R(C_{par}) = \frac{n}{n+1}$ while the $R(C_{dupe}) = \frac{(n/2)}{n} = \frac{1}{2}$. Due to both coding schemes having algorithmically fast encoding, C_{par} having the higher rate makes it the better of the two codes. In fact it is actually the best code possible to correct any 1 erasure due to the Hamming bound.

Definition 3. Hamming Bound
Let C be a **binary code** of block length n and distance d. Then

$$|C| \leq \frac{2^n}{\sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i}}$$

This is the best code possible as for this equality to hold, we would need perfect packing of non-overlapping Hamming spheres that cover every point in $\{0,1\}^n$ once. Codes which meet this bound are considered perfect codes since there is no way to rearrange the hamming spheres to get a better coverage with less overlap. [VAM22]

3 Single Bit Error

The previous code C_{par} does not work in the case of 1 bit error as it would be impossible to differentiate if a c_i had erred or if the parity bit $(c_1 \oplus_2 \oplus c_3 \oplus \dots \oplus c_n)$ had erred.

Example 1. One solution to this is to simply replicate each bit three times.

$$C_{tri} = \{(c_1, c_1, c_1, c_2, c_2, c_2, \dots, c_{n-1}, c_{n-1}, c_{n-1}, c_n, c_n, c_n) \mid c_i \in \{0,1\}\}$$

Given that one bit errors within this scheme we could correct it simply by looking at each triplet and taking the majority element. This encoding method comes at a very heavy cost with $\frac{2}{3}$ of the bits being redundant, giving a rate of $\frac{1}{3}$.

Example 2. Another solution we will call C_{vt} this is implemented in the following way:

$$C_{vt} = \{(c_1, c_2, ...c_n) \in \{0,1\}^n \mid c_1 + 2c_2 + 3c_3... + nc_n \equiv 0 \pmod{B}\}$$

Claim 1. *claim if $B \geq 2n+1$, then C_{vt} can correct one error.*

Proof. Given some $c_i \in C$ experiences an error it can be uniquely identified by this code. For any bit flip the sum of the elements changes by at most n, because the max coefficient of all the elements is bounded by n. Further since each element c_i of the code has a unique weight i that means that there is a one to one map for each possible c_i in mod $2n+1$ space. So we can look at the resulting value and uniquely retrieve which bit had changed. For example, if c_i had erred from $0 \to 1$ then the result mod $2n+1$ would be i, if c_i had erred from $1 \to 0$ then the result mod $2n+1$ would be $2n+1-i$. These values for each index are unique and bounded below n the no flip to a 1 could be confused with a flip to a 0 as have no overlap in mod $2n+1$ space. □

C_{vt} has $\frac{2^n}{n}$ bits with $\log_2(n) + O(1)$ being redundant [Lev66]. Therefore, the rate $R(C_{vt}) = 1 - \frac{\log_2(n)+O(1)}{n}$. This code allows for the same utility but with a much better rate.

3.1 Varshamov-Tenengolts codes

The code C_{vt} is part of a larger family of codes called Varshamov-Tenengolts codes.

Definition 4 (Varshamov-Tenengolts (VT) Codes). For $a \in [0, n]$, the Varshamov-Tenengolts Code $\mathrm{VT}_a(n)$ consists of all binary vectors $(x_1, \ldots, x_n)$ satisfying

$$\sum_{i=1}^{n} ix_i \equiv a \pmod{n+1}$$

3.2 Varshamov-Tenengolts Code Systematic Encoding

Unlike the previous codes that we have seen in this note the systematic encoder to map k-bit message sequences onto codewords in $|\mathrm{VT}_a(n)|$ is more complex. Firstly we will define $k = n - \lceil \log_2(n+1) \rceil$. Now given a k-bit message $M = m_1...m_k$ to be encoded to a codeword $C_{vt} = c_1...c_n \in VT_a(n)$ for some $a \in \{0, 1, \ldots, n\}$. The steps are as follows:

1. Denote the first k non-dyadic positions by $\{j_1 < \ldots < j_k\}$ where indices are in ascending order. Then we set $c_{j_i} = m_i$ for $1 < i < k$. Dyadic positions are which are a tensor of order two and rank one.

2. Set the bits in all the dyadic positions to be 0 and denote the resulting sequence by $C' = c'_1, c'_2 \ldots, c'_n$. In other words we now have $c'_{2^i} = 0$ for $0 \leq i \leq \lceil \log_2(n+1) \rceil - 1$ and $c'_{jl} = m_l$ for $1 \leq l \leq k$.

3. Now we will define the defiency d as the difference between the desired syndrome a and the syndrome of C'. That is,

$$d \equiv a - \text{syndrome}(C') \pmod{n+1}$$

4. Take the binary representation of d to be such that $d = \sum_{i=0}^{\lceil \log_2(n+1) \rceil - 1} 2^i d_i$. Now set $c_{2^i} = d_i$ for $0 \leq i \leq \lceil \log_2(n+1) \rceil - 1$ [AF98].

4 Up to Two Bit Errors

When transferring data over a channel we want our code to be resilient against lots of noise, not just a single error. The previous Varshamov-Tenengolts code can only could correct up to one error, because if it were to see any two adjacent bits flipped the weight would be changed by 1 and what the original code was would be ambiguous. In order to catch and correct up to two bits errors, you can simply add another Varshamov-Tenengolts code to check that all the pairwise errors are distinct. For that to hold, if we have $\alpha_1 c_1, +\alpha_2 c_2 + ...\alpha_n c_n \equiv a' \pmod{M}$, then for all p, r, s, and q in n such that $p, q \neq r, s$, we must have that

$$\pm\alpha_p \pm \alpha_q \neq \pm\alpha_r \pm \alpha_s$$
$$\pm\alpha_p^2 \pm \alpha_q^2 \neq \pm\alpha_r^2 \pm \alpha_s^2$$

Now, rather than sending just the sum, we send the code with the sum and the sum of squares. That being you send $\alpha_1 c_1 + \alpha_2 c_2 + \ldots + \alpha_n c_n \equiv a' \pmod{A}$ and $\beta_1^2 c_1, +\beta_2^2 c_2 + \ldots + \beta^2 c_n \equiv b'$ $\pmod{B}$ where $A \geq 2n+1$ and $B = O(n^2)$. As for the size of this code, we know that there exists a', b' such that a code has size $\geq \frac{2^n}{AB} = \Omega(\frac{2^n}{n^3})$. We can get a tighter bound for the 2 bit example with $|C| \leq \frac{2^n}{1+n+\binom{n}{2}} = O(\frac{2^n}{n^2})$

This leaves us with an encoding scheme with $3\log_2(n) \pm O(1)$ bits, $\log_2(n) + O(1)$ from the first Varshamov-Tenengolts code and $2\log_2(n) + O(1)$ for the second Varshamov-Tenengolts code.

5 Varshamov Tenegolts (VT) Codes as Hamming codes

Currently, our codes are in the form of $1c_1 + 2c_2 + ...nc_n \equiv a \pmod{B}$ rather than an array of binary vectors as our definition stated. To do this transformation in a linear code we change each coefficient to a binary vector ($ic_i \rightarrow \vec{V_i} c_i$) so our code becomes something in this form.

$$\vec{V_1}c_1 + \vec{V_2}c_2 + ...\vec{V_n}c_n = \vec{a} \pmod{q}.$$

We can rewrite this in vector form as

$$\begin{bmatrix} \vec{V_1} & \vec{V_2} & \ldots & \vec{V_n} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \vec{0}$$

Example 3. The 2-bit error correction code above represented in this method is:

$$\begin{bmatrix} \vec{V_1} & \vec{V_2} & \ldots & \vec{V_n} \\ \vec{V_1}^2 & \vec{V_2}^2 & \ldots & \vec{V_n}^2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \vec{0}$$

Remark 1. when working within this new field not all properties when working with $\mathbb{R}$ are retained.

Fact 1. *For any two polynomials $f, g \in \mathbb{F}_2[x]$, we have the identity*

$$(f(x) + g(x))^2 = f(x)^2 + g(x)^2 \quad .$$

As such, $\sum \vec{V_i}^2 C_i = \sum \vec{V_i}^2 C_i^2 = (\sum \vec{V_i} C_i)^2$

Thus the two checks of our above code in over are redundant. We can resolve this by replacing the second check of sum squared with sum cubed so the resulting code looks like:

$$\begin{bmatrix} \vec{V_1} & \vec{V_2} & \dots & \vec{V_n} \\ \vec{V_1}^3 & \vec{V_2}^3 & \dots & \vec{V_n}^3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \vec{0}$$

References

[AF98] Khaled A. S. Abdel-Ghaffar and Hendrik C. Ferreira. Systematic encoding of the varshamov-tenengol'ts codes and the constantin-rao codes. *IEEE Trans. Inf. Theory*, 44(1):340–345, 1998. 4

[Lev66] Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966. 3

[VAM22] Guruswami V., Rudra A., and Sudan M. Essential coding theory. pages 35–37, 2022. 3