Teoría de Codificación y Homomorfismos

Álgebra aplicada

Domingo López Rodríguez

Índice de Contenidos

Introducción: El Problema de la Comunicación **3**

Bloque I: Fundamentos Teóricos **3**

Códigos Lineales: Homomorfismos de Grupos 3

Distancia de Hamming y Capacidad de Corrección 5

Matriz Generadora (G) 6

Matriz de Verificación de Paridad (H) 7

Decodificación por Clases Laterales 8

Decodificación por Síndromes 9

Bloque II: Implementación con R **11**

Funciones Básicas 11

Generación de la Matriz de Paridad 11

Codificación y Cálculo del Síndrome 11

Construcción de la Tabla de Clases Laterales 11

Decodificación por Síndrome 12

Resolución Completa del Ejemplo en Marcha 13

Introducción: El Problema de la Comunicación

Cuando transmitimos información digital —ya sea a través de un cable, ondas de radio, redes wifi o al almacenar datos en un disco duro— la integridad del mensaje se ve amenazada por el **ruido** y las imperfecciones del medio. Incluso los sistemas de comunicación más avanzados experimentan errores. La **Teoría de la Codificación** surge como respuesta a esta problemática, buscando métodos para detectar y corregir errores de manera eficiente.

Un ejemplo sencillo para ilustrar la idea básica es la **repetición**. Si queremos enviar un bit x, podríamos enviarlo triplicado: xxx. Si recibimos $x_1x_2x_3$ y asumimos que como mucho ha ocurrido un error, podríamos decodificar el mensaje original como la "mayoría" de los bits recibidos (por ejemplo, si recibimos 101, decodificaríamos como 1). Sin embargo, este método triplica la longitud del mensaje y sólo corrige un error. La teoría de la codificación busca generalizar y formalizar estas ideas, diseñando sistemas algebraicos mucho más eficientes.

> **i** Nota Histórica: Richard W. Hamming (1915-1998)
>
> Richard Hamming fue un matemático estadounidense que trabajó en los Laboratorios Bell. Su frustración con los errores de las tarjetas perforadas le llevó a desarrollar los códigos que hoy llevan su nombre. Su lema, *"El propósito del cómputo es el entendimiento, no los números"*, sigue siendo una guía para científicos y matemáticos.

Bloque I: Fundamentos Teóricos

Códigos Lineales: Homomorfismos de Grupos

En la comunicación digital, los mensajes se representan como secuencias de bits, es decir, elementos del cuerpo binario $\mathbb{Z}_2 = \{0, 1\}$. Consideramos palabras como vectores de longitud m en el espacio vectorial $\mathbb{Z}_2^m$.

Un **código lineal** se construye utilizando una función de codificación $\mathcal{C}$ que transforma un mensaje de longitud m en una palabra código de longitud $n > m$. Formalmente, es una función

inyectiva:

$$\mathcal{C} : \mathbb{Z}_2^m \longrightarrow \mathbb{Z}_2^n$$

El conjunto imagen $C = \mathrm{Im}(\mathcal{C})$ es el **código**, y sus elementos se denominan **palabras código** o **palabras clave**. En el caso de un código **lineal**, $\mathcal{C}$ es un **homomorfismo de grupos**, es decir:

$$\mathcal{C}(\mathbf{x} + \mathbf{y}) = \mathcal{C}(\mathbf{x}) + \mathcal{C}(\mathbf{y}) \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{Z}_2^m$$

Además, se tiene que $\mathcal{C}(\mathbf{0}) = \mathbf{0}$.

> **!** El Código como Subgrupo
>
> La propiedad de homomorfismo implica que el código $C = \mathrm{Im}(\mathcal{C})$ es un **subgrupo** (y subespacio vectorial) de $\mathbb{Z}_2^n$. Esta estructura de subgrupo es fundamental para todo el diseño y análisis posterior.

Ejemplo 0.1. Sea la función de codificación $\mathcal{C} : \mathbb{Z}_2^2 \to \mathbb{Z}_2^5$ definida por:

- $\mathcal{C}(00) = 00000$
- $\mathcal{C}(10) = 10110$
- $\mathcal{C}(01) = 01011$
- $\mathcal{C}(11) = 11101$

Verificación del homomorfismo: Comprobemos que $\mathcal{C}(10) + \mathcal{C}(01) = \mathcal{C}(10 + 01) = \mathcal{C}(11)$:

$$10110 + 01011 = 11101 = \mathcal{C}(11) \quad \checkmark$$

El código $C = \{00000, 10110, 01011, 11101\}$ tiene $|C| = 2^m = 2^2 = 4$ elementos, como corresponde a un subgrupo imagen de $\mathbb{Z}_2^2$.

Distancia de Hamming y Capacidad de Corrección

Para evaluar la calidad de un código, necesitamos medir su capacidad para detectar y corregir errores.

La **distancia de Hamming** $\delta(\mathbf{u}, \mathbf{v})$ entre dos palabras $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_2^n$ es el número de posiciones en las que difieren:

$$\delta(\mathbf{u}, \mathbf{v}) = |\mathbf{u} \oplus \mathbf{v}|$$

donde $|\mathbf{w}|$ denota el **peso de Hamming** de $\mathbf{w}$ (el número de componentes no nulas). La distancia de Hamming es una **métrica** en $\mathbb{Z}_2^n$.

La **distancia mínima** de un código C, denotada $d_{\min}(C)$, es:

$$d_{\min}(C) = \min\{\delta(\mathbf{c}_1, \mathbf{c}_2) \mid \mathbf{c}_1, \mathbf{c}_2 \in C, \mathbf{c}_1 \neq \mathbf{c}_2\}$$

> **Capacidad de Detección y Corrección**
>
> Un código C con distancia mínima $d_{\min}$ puede:
>
> - **Detectar** hasta $d_{\min} - 1$ errores: si se producen hasta $d_{\min} - 1$ errores, la palabra recibida no será una palabra código.
> - **Corregir** hasta $\lfloor \frac{d_{\min}-1}{2} \rfloor$ errores: existe una única palabra código a distancia mínima de la palabra recibida.

Teorema (Distancia mínima en códigos lineales): Para un código lineal C, la distancia mínima es igual al mínimo peso de una palabra código no nula:

$$d_{\min}(C) = \min\{|\mathbf{c}| \mid \mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}\}$$

Demostración:

1. Sea $d_{\min} = \min\{\delta(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}\}$.
2. Por definición, $\delta(\mathbf{x}, \mathbf{y}) = |\mathbf{x} \oplus \mathbf{y}| = |\mathbf{x} + \mathbf{y}|$ en $\mathbb{Z}_2$.
3. Como C es un subgrupo, $\mathbf{z} = \mathbf{x} + \mathbf{y} \in C$ y $\mathbf{z} \neq \mathbf{0}$ (pues $\mathbf{x} \neq \mathbf{y}$).

4. Por tanto, toda distancia entre pares es el peso de alguna palabra código no nula, y recíprocamente, todo peso de una palabra no nula es la distancia entre esa palabra y $\mathbf{0} \in C$. □

Esta propiedad simplifica significativamente el cálculo de $d_{\min}$: solo necesitamos examinar los pesos de las palabras código no nulas.

Ejemplo 0.2. Calculemos los pesos de las palabras no nulas de nuestro código:

- $|10110| = 3$
- $|01011| = 3$
- $|11101| = 4$

Conclusión: $d_{\min} = 3$. Por tanto:

- Podemos **detectar** hasta $3 - 1 = 2$ errores.
- Podemos **corregir** hasta $\lfloor (3-1)/2 \rfloor = 1$ error.

Matriz Generadora (G)

Los códigos lineales se describen mediante matrices. Una **matriz generadora** G de un código lineal de dimensión k y longitud n es una matriz $k \times n$ cuyas filas forman una base para C. La función de codificación se define como:

$$\mathcal{C}(\mathbf{m}) = \mathbf{m}G \pmod 2$$

Una matriz generadora **sistemática** tiene la forma $G = (I_k \mid P)$, donde I_k es la matriz identidad $k \times k$ y P es una matriz $k \times (n-k)$. En esta forma, las primeras k componentes de la palabra código son el mensaje original $\mathbf{m}$, y las $n-k$ restantes son los **símbolos de control** (redundancia).

Ejemplo 0.3. La matriz generadora de nuestro código $(2, 5)$ es:

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} = (I_2 \mid P)$$

donde $P = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ es la parte de control.

Verificación: Codifiquemos el mensaje $\mathbf{m} = (1, 1)$:

$$(1, 1) \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} = (1, 1, 1, 0, 1) \pmod 2$$

¡Coincide con $\mathcal{C}(11) = 11101$ de nuestra lista inicial!

Matriz de Verificación de Paridad (H)

Asociada a cada código lineal, existe la **Matriz de Verificación de Paridad** H de dimensiones $n \times (n - k)$. Si la matriz generadora es de la forma $G = (I_k \mid P)$, entonces H se construye como:

$$H = \begin{pmatrix} P \\ I_{n-k} \end{pmatrix}$$

Teorema: Si $G = (I_k \mid P)$ y $H = \begin{pmatrix} P \\ I_{n-k} \end{pmatrix}$, entonces $GH = 0$ (matriz nula $k \times (n - k)$).

Demostración:

1. $GH = (I_k \mid P) \begin{pmatrix} P \\ I_{n-k} \end{pmatrix} = I_k \cdot P + P \cdot I_{n-k} = P + P = 0 \pmod 2$. $\square$

La importancia de H radica en que una palabra $\mathbf{c} \in \mathbb{Z}_2^n$ es una palabra código si y solo si $\mathbf{c}H = \mathbf{0}$.

Ejemplo 0.4. Con $P = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$, la matriz de paridad es:

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

Verificación: Comprobemos que la palabra código $\mathcal{C}(10) = 10110$ cumple $\mathbf{c}H = \mathbf{0}$:

$$(1,0,1,1,0) \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (1+1, 1+1, 0) = (0,0,0) \pmod 2 \quad \checkmark$$

Decodificación por Clases Laterales

Recordemos que el código $\mathcal{W} = \mathrm{Im}(\mathcal{C})$ es un subgrupo de $\mathbb{Z}_2^n$. Cuando recibimos una palabra que **no** es código, esta pertenece a una **clase lateral** $\mathbf{e} \oplus \mathcal{W}$, donde $\mathbf{e}$ es un vector de error. La idea es que la palabra código más cercana a una recibida es la que aparece en la misma clase lateral.

Ejemplo 0.5. Para nuestro código $\mathcal{W} = \{00000, 01011, 10110, 11101\}$, la tabla completa de clases laterales es:

$\mathrm{Im}(\mathcal{C}) = \mathcal{W}$	00000	01011	10110	11101
$00001 \oplus \mathcal{W}$	00001	01010	10111	11100
$00010 \oplus \mathcal{W}$	00010	01001	10100	11111
$00100 \oplus \mathcal{W}$	00100	01111	10010	11001
$01000 \oplus \mathcal{W}$	01000	00011	11110	10101
$10000 \oplus \mathcal{W}$	10000	11011	00110	01101
$00101 \oplus \mathcal{W}$	00101	01110	10011	11000
$10001 \oplus \mathcal{W}$	10001	11010	00111	01100

El encabezado contiene las palabras código. Cada fila es una clase lateral. Para decodificar una palabra recibida, basta encontrarla en la tabla: la palabra código correcta es la del encabezado de su columna. Por ejemplo, si recibimos 11110, la palabra código más cercana es 10110, y el mensaje original sería 10.

Sin embargo, este método es **poco eficiente**, pues requiere generar toda la tabla de 2^n entradas. ¡Para eso existen los síndromes!

Decodificación por Síndromes

El **síndrome** de una palabra recibida $\mathbf{r} \in \mathbb{Z}_2^n$ se define como:

$$\mathrm{sind}(\mathbf{r}) = \mathbf{r}H$$

Teorema: Dos palabras $\mathbf{r}_1$ y $\mathbf{r}_2$ pertenecen a la misma clase lateral de $\mathcal{W}$ si y solo si $\mathrm{sind}(\mathbf{r}_1) = \mathrm{sind}(\mathbf{r}_2)$.

Esto establece una correspondencia biunívoca entre las 2^{n-k} clases laterales y los 2^{n-k} síndromes posibles, lo cual es mucho más eficiente que almacenar la tabla completa.

Algoritmo de Decodificación por Síndrome

1. Para cada síndrome **s** posible, elegir un **líder de clase lateral** $\mathbf{e_s}$ de peso mínimo. Este se interpreta como el error más probable.
2. Construir una **tabla de síndromes** que empareje cada **s** con su líder $\mathbf{e_s}$.

3. Al recibir $\mathbf{r}$, calcular $\mathbf{s} = \mathbf{r}H$.
4. Buscar en la tabla el líder $\mathbf{e_s}$ correspondiente.
5. Decodificar: $\mathbf{c} = \mathbf{r} + \mathbf{e_s}$.
6. Extraer el mensaje original $\mathbf{m}$ de las primeras k componentes de $\mathbf{c}$.

Ejemplo 0.6. La tabla de síndromes para nuestro código $(2, 5)$:

Síndrome s	Líder e	Justificación
$(0, 0, 0)$	00000	No hay error (síndrome nulo)
$(1, 1, 0)$	10000	Coincide con la Fila 1 de H
$(0, 1, 1)$	01000	Coincide con la Fila 2 de H
$(1, 0, 0)$	00100	Coincide con la Fila 3 de H
$(0, 1, 0)$	00010	Coincide con la Fila 4 de H
$(0, 0, 1)$	00001	Coincide con la Fila 5 de H

Dato clave: Cuando el error es de peso 1, el síndrome coincide directamente con la **fila de H correspondiente a la posición del error**. Esto hace la decodificación extremadamente eficiente.

Ejemplo 0.7. Ejercicio resuelto completo: Supongamos que enviamos el mensaje $\mathbf{m} = (1, 1)$.

1. **Codificación**: $\mathbf{c} = (1, 1)G = (1, 1, 1, 0, 1) = 11101$.
2. **Error en la transmisión**: Se introduce un error en el bit 4, recibimos $\mathbf{r} = 11111$.
3. **Cálculo del síndrome**: $\mathbf{s} = \mathbf{r}H = (1, 1, 1, 1, 1)H = (0, 1, 0) \pmod 2$.
4. **Búsqueda en la tabla**: El síndrome $(0, 1, 0)$ corresponde al líder $\mathbf{e} = 00010$ (error en bit 4).
5. **Corrección**: $\mathbf{c} = \mathbf{r} + \mathbf{e} = 11111 + 00010 = 11101$.
6. **Extracción del mensaje**: Las primeras $k = 2$ componentes de 11101 son $\mathbf{m} = (1, 1)$. ¡Mensaje recuperado!

Bloque II: Implementación con R

Implementaremos en R todas las operaciones de codificación y decodificación que hemos visto en la teoría, utilizando nuestro código $(2, 5)$ como ejemplo.

```
library(matlib)
library(dplyr)
```

Funciones Básicas

Generación de la Matriz de Paridad

Dada una matriz generadora en forma sistemática $G = (I_k \mid P)$, la función `generar_H` construye automáticamente la matriz de verificación de paridad $H = \begin{pmatrix} P \\ I_{n-k} \end{pmatrix}$:

```
generar_H <- function(G) {
  k <- nrow(G)
  n <- ncol(G)
  P <- G[, (k + 1):n]
  I_nk <- diag(1, n - k)
  H <- rbind(P, I_nk)
  return(H)
}
```

Codificación y Cálculo del Síndrome

```
codificar <- function(m, G) { (m %*% G) %% 2 }
calcular_sindrome <- function(r, H) { (r %*% H) %% 2 }
```

Construcción de la Tabla de Clases Laterales

Esta función genera automáticamente la tabla que asocia cada síndrome con su líder de clase lateral (vector de error de peso mínimo):

```r
construir_tabla_clases <- function(H) {
  n <- nrow(H)
  n_menos_k <- ncol(H)
  tabla <- list()

  # Síndrome nulo = sin error
  tabla[[paste(rep(0, n_menos_k), collapse = "")]] <- rep(0, n)

  # Vectores de error de peso 1
  for (i in 1:n) {
    e <- rep(0, n)
    e[i] <- 1
    s <- (e %*% H) %% 2
    s_str <- paste(s, collapse = "")
    if (!(s_str %in% names(tabla))) {
      tabla[[s_str]] <- e
    }
  }
  return(tabla)
}
```

Decodificación por Síndrome

```r
decodificar <- function(r, H, tabla) {
  s <- calcular_sindrome(r, H)
  s_str <- paste(s, collapse = "")
  e <- tabla[[s_str]]
  c_corregido <- (r + e) %% 2
  return(c_corregido)
}
```

```r
extraer_mensaje <- function(c_codigo, G) {
  k <- nrow(G)
  return(c_codigo[1:k])
}
```

Resolución Completa del Ejemplo en Marcha

Ejemplo 0.8. Reproducimos en R todo el proceso teórico visto anteriormente.

Paso 1: Definir las matrices

```r
G <- matrix(c(1,0,1,1,0,
              0,1,0,1,1), nrow = 2, byrow = TRUE)
H <- generar_H(G)
cat("Matriz Generadora G:\n"); print(G)
```

```
Matriz Generadora G:
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    1    1    0
[2,]    0    1    0    1    1
```

```r
cat("\nMatriz de Paridad H:\n"); print(H)
```

```
Matriz de Paridad H:
     [,1] [,2] [,3]
[1,]    1    1    0
[2,]    0    1    1
[3,]    1    0    0
[4,]    0    1    0
[5,]    0    0    1
```

Paso 2: Codificar un mensaje

```r
m <- c(1, 1)
c_ok <- codificar(m, G)
cat("Mensaje:", m, "\n")
```

```
Mensaje: 1 1
```

```r
cat("Palabra código:", c_ok, "\n")
```

```
Palabra código: 1 1 1 0 1
```

Paso 3: Simular un error y calcular el síndrome

```r
r_err <- c_ok
r_err[4] <- (r_err[4] + 1) %% 2  # Error en bit 4
cat("Palabra recibida con error:", r_err, "\n")
```

```
Palabra recibida con error: 1 1 1 1 1
```

```r
cat("Síndrome:", calcular_sindrome(r_err, H), "\n")
```

```
Síndrome: 0 1 0
```

Paso 4: Construir la tabla y decodificar

```r
tabla <- construir_tabla_clases(H)
c_corregido <- decodificar(r_err, H, tabla)
m_recuperado <- extraer_mensaje(c_corregido, G)
cat("Palabra corregida:", c_corregido, "\n")
```

```
Palabra corregida: 1 1 1 0 1
```

```r
cat("Mensaje recuperado:", m_recuperado, "\n")
```

```
Mensaje recuperado: 1 1
```

Verificación final: El síndrome $(0, 1, 0)$ señala la Fila 4 de H, indicando un error en el bit 4. Tras la corrección, recuperamos la palabra código 11101 y el mensaje original $(1, 1)$.