Modelos discretos sencillos

Algunos ejemplos cotidianos

Hay diversos métodos simples basados en congruencias que permiten verificar datos en tiempo real con un mínimo de fiabilidad. Veamos unos cuantos ejemplos.

La letra del DNI/NIF. La letra del DNI no es arbitraria, es un carácter de control que está determinado por el número. Con ello cuando introducimos el DNI completo en una aplicación informática puede comprobar fácilmente si nos hemos equivocado al teclear. La comprobación se basa en que el resto al dividir el número por 23 debe coincidir con el número asignado a la letra según la siguiente tabla:

T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Por ejemplo, un DNI con número $n = 12345678$ debe tener letra `Z` porque $n \equiv 14$ (mód 23).

Si llamamos $c_1c_2c_3c_4c_5c_6c_7c_8$ a las cifras del DNI, considerando 10^k módulo 23, se tiene que el número L que corresponde a la letra cumple

$$-9c_1 + 6c_2 - 4c_3 - 5c_4 + 11c_5 + 8c_6 + 10c_7 + c_8 \equiv L \pmod{23}.$$

Es fácil que con esta información podríamos recuperar un dígitos ilegible pero no dos.

El código de barras (EAN13). Los códigos de barras que aparecen en gran número de objetos cotidianos son una manera de codificar como líneas blancas y negras una lista de 13 dígitos que aparecen indicados debajo del código.

La manera en que se codifican estos dígitos es un poco compleja y está destinada a evitar confusiones al pasar el objeto por el lector en diferentes posiciones. Los dígitos tampoco son arbitrarios sino que el último está determinado por los anteriores según la fórmula:

$$c_{13} \equiv -\sum_{i=1}^{6} \left(c_{2i-1} + 3c_{2i}\right) \pmod{10}.$$

Por ejemplo si los dígitos son 8410055050011 entonces se tiene que la suma de los que están en el lugar impar excepto el último es 19, y de los que están en lugar par es 10. Se cumple $1 \equiv -19 - 3 \cdot 10 \pmod{10}$.

El ISBN-10. Los libros (desde 1970) llevan un código que los identifican de manera única, el *International Standard Book Number*. A partir de 2007 se usa el código de barras EAN13 pero los libros editados antes de este año llevan 10 dígitos, habitualmente separados por

guiones en grupos (porque cada grupo da cierta información). Numerando como antes los dígitos como $c_1c_2\cdots c_{10}$, se cumple

$$c_{10}\equiv\sum_{i=1}^{9} ic_i \pmod{11}.$$

Por tanto, de nuevo, el último dígito es de control. Como las congruencias son módulo 11, podría ser $c_{10}=10$. En ese caso se escribe una `X` (diez en romanos) en lugar de un dígito.

Código cuenta cliente. Las cuentas en bancos españoles se representan mediante 20 dígitos que conforman el *código cuenta cliente* o CCC. Los cuatro primeros representan la entidad, los cuatro siguientes, la oficina y los diez últimos el número de cuenta propiamente dicho. Los dos dígitos en las posiciones 9 y 10, bajo los cuales en las libretas pone `DC`, son *Dígitos de Control*, sólo para comprobar la información y minimizar errores por ejemplo cuando se hace una transferencia.

Digamos que nombramos los dígitos de la manera siguiente:

$$e_1e_2e_3e_4\ e_5e_6e_7e_8\ d_1d_2\ c_1c_2c_3c_4c_5c_6c_7c_8c_9c_{10}$$

es decir, los dígitos de control son d_1d_2. Entonces estos dígitos de control dependen del resto a través de las fórmulas:

$$d_1\equiv-\sum_{i=0}^{7}2^{i+2}e_{i+1} \pmod{11} \qquad \text{y} \qquad d_2\equiv-\sum_{i=0}^{9}2^{i}c_{i+1} \pmod{11}.$$

En realidad, en la práctica hay una pequeña chapuza, y es que cuando se obtiene $d_i\equiv 10 \pmod{11}$ se toma $d_i=1$, en vez de introducir un nuevo carácter como en el ISBN-10.

Descartando esta deficiencia en la implementación práctica, el código es capaz de detectar una trasposición entre dos dígitos consecutivos distintos (un error común cuando se teclea) porque $\alpha 2^n+\beta 2^{n+1}\equiv\beta 2^n+\alpha 2^{n+1} \pmod{11}$ implica $\alpha\equiv\beta \pmod{11}$.

Bases de la teoría de códigos lineales

En todos los ejemplos anteriores, tenemos ciertos datos a los que queremos dar fiabilidad y los representamos con algo de redundancia para reducir errores. Llamaremos a este proceso *codificación* y a cada traducción de un dato por esta codificación, un *código*.

La redundancia que se introducía en los ejemplos venía representada por relaciones lineales. Las relaciones lineales definen subespacios vectoriales dados por el núcleo de la aplicación que definen las ecuaciones. Esto motiva el contexto de los *códigos lineales* en que los códigos forman un subespacio vectorial $V=\ker(f)$ para alguna aplicación lineal f. Su matriz se suele denotar con H y se llama *matriz de paridad* o *matriz de comprobación*. La razón de este nombre, es que comprobar un código $\vec{c}\in V$ es comprobar si $f(\vec{c})=\vec{0}$ o equivalentemente $H\vec{c}=\vec{0}$.

Por ejemplo, el en caso del DNI las 8 cifras y la letra se pueden codificar como 9 clases módulo 23 que satisfacen la relación lineal $-9c_1+6c_2-4c_3-5c_4+11c_5+8c_6+10c_7+c_8-c_9=0$

en $\mathbb{Z}_{23}$. Así pues, podemos entender un DNI como un elemento de $\mathbb{Z}_{23}^9$ que está en el subespacio dado por la aplicación $f(\vec{x}) = H\vec{x}$ con H la matriz fila

$$H = \begin{pmatrix} -9 & 6 & -4 & -5 & 11 & 8 & 10 & 1 & -1 \end{pmatrix}.$$

En el caso del CCC podemos considerar que $f : \mathbb{Z}_{11}^{20} \longrightarrow \mathbb{Z}_{11}^2$, con $f(\vec{x}) = H\vec{x}$ y

$$H = \begin{pmatrix} 2^2 & 2^3 & \dots & 2^9 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 & 2^0 & 2^1 & \dots & 2^9 \end{pmatrix}.$$

Los códigos lineales son importantes en aplicaciones informáticas que requieren correcciones de errores a gran escala. En este contexto es natural trabajar en espacios vectoriales sobre $\mathbb{Z}_2$ (los unos y ceros de los *bits*) o, en general, sobre los cuerpos finitos $\mathbb{F}_{2^n}$, lo cual es bastante curioso porque en los cursos de matemáticas estos cuerpos parecían una gran abstracción, pues provenían de un teorema de existencia del cuerpo raíz y no sabíamos ni siquiera escribir sus elementos.

Sólo veremos aquí el caso de $\mathbb{Z}_2$ y consideraremos $f : \mathbb{Z}_2^n \longrightarrow \mathbb{Z}_2^{n-k}$ con f sobreyectiva. Es decir, cada código será una palabra de n bits que está en un subespacio vectorial de dimensión k (por la fórmula de la dimensión).

Si se comente un error en el i-ésimo bit de un código $\vec{c}$ es que se ha cambiado $\vec{c}$ por $\vec{c} + \vec{e_i}$ (donde $\vec{e_i}$ es la notación habitual para el i-ésimo vector de la base canónica). Para detectar este error, $\vec{c} + \vec{e_i}$ no debería estar en el espacio de códigos $\ker(f)$. De $f(\vec{c}) = \vec{0}$ y $f(\vec{c} + \vec{e_i}) \neq \vec{0}$, por la linealidad se obtiene que la condición para detectar un error en un solo bit es

$$f(\vec{e_i}) \neq \vec{0} \qquad \text{para } i = 1, 2, \dots, n.$$

Por otro lado, para corregir un error una vez que lo hemos detectado, debemos saber en qué bit está. Es decir, tenemos que ser capaces de distinguir entre $\vec{c} + \vec{e_1}$, $\vec{c} + \vec{e_2}$, $\vec{c} + \vec{e_3}$, etc. Para ello, al aplicar f y usando de nuevo la linealidad, se obtiene la condición

$$f(\vec{e_i}) \neq f(\vec{e_j}) \qquad \text{para } i, j = 1, 2, \dots, n \text{ con } i \neq j.$$

En términos matriciales la condición para detectar un error es que H no tenga columnas nulas, mientras que la condición para corregirlo una vez detectado es que tenga todas sus columnas distintas.

Veamos un ejemplo no trivial. Si tomamos todos las representaciones binarias de un número del 1 al 7 y las ponemos en columnas, obtenemos una matriz válida para corregir un error con $n = 7$ y $n - k = 3$:

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

Si vemos el núcleo de la aplicación lineal $f(\vec{x}) = H\vec{x}$, $f : \mathbb{Z}_2^8 \longrightarrow \mathbb{Z}_2^3$, se obtiene

$$\ker(f) = \left\langle \left\{ \vec{v}_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \vec{v}_2 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \vec{v}_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \vec{v}_4 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\} \right\rangle.$$

Los $\vec{v}_i$ son linealmente independientes, esto es, $\{\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4\}$ es una base. Este subespacio tiene 16 elementos y permite codificar cualquier palabra $b_1b_2b_3b_4$ de cuatro bits como

$$b_1b_2b_3b_4 \longmapsto \vec{c} = b_1\vec{v}_1 + b_2\vec{v}_2 + b_3\vec{v}_3 + b_4\vec{v}_4.$$

Si $\widehat{c}$ es un código con un error en el bit j-ésimo, lo detectaremos y lo podremos corregir simplemente viendo que $H\widehat{c}$ es la j-ésima columna de H, que coincide con j en binario. Por ejemplo

$$\texttt{0011} \xrightarrow{\text{codifica}} \texttt{1000011} \xrightarrow{\text{error}} \texttt{1010011}.$$

Aplicando H al vector $(1,0,1,0,0,1,1)^t$ se obtiene $(0,1,1)^t$ que es 3 en binario, entonces el error a corregir está en el tercer bit.

El diseño de códigos que corrijan varios errores no se refleja tan fácilmente en H y hay una serie de métodos de la teoría de códigos con este propósito. También como cabría esperar, hay ciertos límites para la redundancia que tiene el código y su capacidad para corregir errores.

Una de las aplicaciones cotidianas de los códigos lineales, en una versión bastante complicada, son los códigos correctores empleados en los CDs y DVDs. La tecnología actual no es tan precisa como para grabar estos soportes sin errores (a un precio competitivo), por otra parte, el uso causa pequeñas erosiones y arañazos que pueden impedir la lectura de datos, los cuales están grabados a escala micrométrica. Entonces es totalmente necesario un método para corregir errores que funcione en tiempo real. La base de este método son los *códigos de Reed-Solomon* que son bastante complejos. En [Cip93] se da una leve idea de su estrategia. Por supuesto, hay mucha bibliografía sobre el tema. Una introducción sencilla a la teoría de códigos es [Hil86].

Referencias

[Cip93] B. Cipra. The ubiquitous Reed-Solomon codes. *SIAM News* *http://www.eccpage.com/reed_solomon_codes.html*, 26(1), 1993.

[Hil86] R. Hill. *A first course in coding theory.* Oxford Applied Mathematics and Computing Science Series. The Clarendon Press Oxford University Press, New York, 1986.