Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Matemáticas

CÓDIGOS NEURALES COMBINATORIOS DESDE LA PERSPECTIVA DE LOS CÓDIGOS CORRECTORES DE ERRORES

Autora: Laura María Espejo Domínguez

Tutor: Diego Ruano Benito

Año: 2024/2025

Resumen

Este trabajo de fin de grado tiene dos objetivos principales. El primero es modelizar matemáticamente la respuesta de una población de neuronas ante un conjunto de estímulos, utilizando códigos binarios correctores de errores cuya transmisión se estudia a través de un canal binario asimétrico. Para ello, se introducen los códigos del campo receptivo, que asocian cada estímulo con un patrón de activación neuronal, es decir con el conjunto de neuronas que responden a él, identificando cada una de ellas como posiciones de un vector. El segundo objetivo es analizar en detalle el canal binario asimétrico, definiendo nuevos parámetros que permitan medir la discrepancia entre las palabras del código y explorando distintos métodos de decodificación. A diferencia de los canales simétricos, que son lo habitual en teoría de códigos, la asimetría de este canal permite ajustarse mejor a los datos experimentales y proporciona una representación más realista de la transmisión de información en el cerebro.

Abstract

This thesis has two main objectives. The first is to mathematically model how a population of neurons responds to a set of stimuli, using binary error-correcting codes transmitted through an asymmetric binary channel. To achive this, receptive field codes are introduced, establishing a relationship between each stimulus and a neuronal activation pattern—that is, the set of neurons that respond to it—where each neuron is represented as a position in a vector. The second objective is to conduct a detailed analysis of the asymmetric binary channel, defining new parameters to quantify the discrepancy between codewords and exploring different decoding methods. While symmetric channels are the standard in coding theory, incorporating asymmetry allows to create a model that aligns better with the experimental data, offering a more accurate representation of how information is transmitted in the brain.

Palabras clave

Teorema de codificación de Shannon, Codificación Neural, Códigos del Campo Receptivo, Códigos Correctores, Respuesta Neural, Estímulos, Canal Binario Asimétrico, Discrepancia, Discrepancia Simétrica, Decodificador de Discrepancia Mínima, Probabilidad de Decodificación no Exitosa.

Key words

Shannon's Coding Theorem, Neural Coding, Receptive Field Codes, Error-Correcting Codes, Neural Response, Stimuli, Binary Asymmetric Channel, Discrepancy, Symmetric Discrepancy, Minimum Discrepancy Decoder, Probability of Unsuccessful Decoding.

Índice general

Introducción

La codificación neural es un campo de estudio dentro de la neurociencia que consiste en caracterizar la relación que existe entre los estímulos y la respuesta neural que da nuestro cerebro a ellos. Un estímulo puede tener un origen externo a nosotros, que captamos través de los órganos sensitivos o provenir del interior de nuestro organismo, siendo en este caso, los interceptores los encargados de recibir esta información, pero de cualquier manera son siempre los estímulos y la información contenida en ellos lo que desencadena la activación neural. Las neuronas son células que tienen la capacidad especial de propagar señales eléctricas rápidamente mediante potenciales de acción, que son ondas que se transmiten a lo largo de la membrana celular modificando la distribución de la carga eléctrica a su paso, la información recibida en forma de estímulo se codifica en el cerebro en un patrón de potenciales de acción que conlleva en que se activen un conjunto de neuronas. El estudio de como una población de neuronas responde a un conjunto estímulos tiene fundamentalmente dos objetivos claros, uno es predecir las respuestas de la población de neuronas a nuevos estímulos y dos, el procedimiento inverso, encontrar el estímulo al que se ve sometido el cerebro a partir de una respuesta neural, un patrón de potenciales de acción o un conjunto de neuronas activas.

El objetivo de este trabajo de fin de grado es modelizar matemáticamente la respuesta de una población de neuronas a un conjunto de estímulos a los que se ve expuesta. Para alcanzar esta meta vamos a utilizar las herramientas que nos proporciona la teoría de códigos, en particular vamos a trabajar con los denominados códigos correctores que permiten recuperar la información corrompida, tras su paso a través de un canal *ruidoso*. Estos códigos se denominan códigos neurales del campo receptivo, también sería correcto emplear el término neuronal, que tiene el mismo significado. Pero es importante destacar, que este estudio no está relacionado, en principio, con redes neuronales [8] ni con inteligencia artificial. Mientras que las redes neuronales artificiales son modelos computacionales utilizados para entrenar algoritmos, este trabajo se centra en la modelización matemática del funcionamiento del cerebro desde una perspectiva teórica, precisamente por esta confusión posible, se considera que usar el término neural en este contexto es más adecuado. Además, cabe mencionar que el canal binario asimétrico que utilizaremos para modelizar la respuesta neural, aún es un área de investigación reciente en un estado inicial de desarrollo. En particular, el artículo de referencia [2] que se ha utilizado es del año 2022, y su impacto y aplicaciones en el estudio de los códigos neurales están aún por explorarse en profundidad.

Nos remontamos para desarrollar esta teoría al trabajo seminal de Claude Elwood Shannon publicado en el año 1948, *A Mathematical Theory of Comunication* [12] que fue la base para el posterior libro que escribió junto a Warren Weaver titulado *The Mathematical Theory of Comunication* que vio la luz un año más tarde, donde se introdujo un pequeño cambio en el nombre, que ya refleja la relevancia de su trabajo. Este texto da lugar a dos campos de investigación la teoría de la información y la teoría de códigos, mientras que la teoría de la información si ha sido ampliamente usada en neurobiología teórica, la representación de la respuesta neural a través de códigos es algo bastante novedoso cuando se publica en el año 2013 el artículo *Combinatorial Neural Codes*

from a Mathematical Coding Theory Perspective [4], por Carina Curto, Vladimir Itskov, Katherine Morrison, Zachary Roth y Judy L. Walker, esté será otra referencia fundamental de este trabajo.

Este trabajo consta de tres capítulos, en el **capítulo 1** se dará una introducción a la teoría de la información tratando de responder a las preguntas fundamentales de ¿como se modela matemáticamente la información?, y más aún, ¿cómo podemos medirla?, aquí entrarán en juego los conceptos clave de fuente de información, cantidad de información y entropía. Aplicaremos estos conocimientos a un canal con ruido teniendo como objetivo final llegar a demostrar del teorema de codificación de Shannon, piedra angular de la teoría de códigos. Junto a otras referencias, se han utilizado fundamentalmente el libro *Elements of information theory* [3] de los autores Thomas Cover y Joy Thomas y el texto *Codificación de Información* escrito por Carlos Munuera y Juan Tena [10].

Posteriormente en el **capítulo 2**, se estudiarán los códigos neurales que se presentan en el artículo científico ya mencionado [4]. Un código neural no es más que la aplicación entre los estímulos y la respuesta neural que provocan en la población de neuronas estudiada. Vamos a utilizar códigos binarios, donde en cada palabra del código, un dígito no nulo representa una neurona que genera un potencial de acción, respondiendo a un estímulo al que se ve sometida; mientras que un dígito nulo equivaldrá a una neurona que permanece en silencio, que no transmite el potencial de acción. Esta nueva visión de la representación neural de los estímulos presenta varios aspectos clave, por un lado, se hace una distinción entre las palabras del código, que serán simplemente un conjunto de vectores desprovisto de un significado intrínseco y los estímulos reales que recibe el cerebro; por otro lado, la aplicación de codificación que se presenta es determinista, es decir se considera que las diferentes representaciones neurales de un mismo estímulo son consecuencia de la transmisión de la respuesta a través de un canal con ruido, lo que provoca alteraciones en ella.

Finalmente, en el **capítulo 3**, vamos a estudiar el canal binario asimétrico utilizado para el modelo siguiendo el artículo *Parameters of Codes for the Binary Asymmetric Channel* [2], de los autores Giuseppe Cotardo y Alberto Ravagnani. Lo que caracteriza a este canal es que la probabilidad de que un dígito se transforme en otro a causa del ruido, depende del propio valor de este. Es decir, en un código binario existen dos probabilidades diferentes de que se produzca un error en un dígito, una si el dígito que se envía es el cero y otra si es el uno. Este canal es relevante, ya que no es habitual en teoría de códigos, siendo lo más común el uso de canales simétricos, donde la probabilidad de error en la transmisión de un dígito no depende del valor de este. Debido a la naturaleza del cerebro este canal asimétrico es más adecuado para la representación de la respuesta neural, ya que se ha comprobado de forma experimental que es más probable que una neurona no responda a un estímulo cuando estaba previsto que lo hiciera, a que ocurra lo contrario.

La relevancia de esta representación neural de los estímulos en neurociencia por códigos neurales que se envían a través de canales asimétricos, es patente en numerosos artículos científicos posteriores que utilizan esta construcción para diferentes estudios de índole biológica. Se proporcionan algunos ejemplos de su utilidad en la conclusión final del trabajo, **capítulo 4**.

Introduction

Neural coding is a field of study within neuroscience that aims to characterize the relationship between stimuli and the neural response our brain generates to them. A stimulus can originate externally, being perceived through sensory organs, or it can come from within our own body, in which case interceptors are responsible for receiving this information. However, regardless of its origin, it is always the stimuli and the information they carry that trigger neural activation. Neurons are specialized cells capable of rapidly propagating electrical signals through action potentials—waves that travel along the cell membrane, altering the distribution of electrical charge as they move. The information received as a stimulus is encoded in the brain as a pattern of action potentials, leading to the activation of a group of neurons. The study of how a population of neurons responds to a set of stimuli has two fundamental objectives: first, to predict the responses of a neuron population to new stimuli; and second, the inverse problem—determining the stimulus to which the brain was exposed based on a neural response, a pattern of action potentials, or a set of active neurons.

The objective of this undergraduate thesis is to mathematically model the response of a population of neurons to a given set of stimuli. To achieve this, we will employ the tools provided by coding theory, specifically error-correcting codes, which allow the recovery of corrupted information after passing through a *noisy* channel. These codes are referred to as receptive field neural codes (also known as neuronal codes, as both terms share the same meaning). However, it is important to emphasize that this study is not related to neural networks or artificial intelligence. While artificial neural networks are computational models used to train algorithms, this work focuses on the mathematical modeling of brain function from a theoretical perspective. Additionally, the binary asymmetric channel we will use to model the neural response is still a relatively new research area, in an early stage of development. In particular, the reference paper [2] used in this study was published in 2022, and its impact and applications in neural code research are still being explored in depth.

To develop this theory, we trace back to the seminal work of Claude Elwood Shannon, published in 1948, *A Mathematical Theory of Communication* [12], which later became the foundation for the book he co-authored with Warren Weaver, *The Mathematical Theory of Communication*, published a year later. The slight modification in the title already reflects the significance of their work. This text gave rise to two fields of research: information theory and coding theory. While information theory has been widely applied in theoretical neurobiology, the representation of neural responses through codes is a relatively new approach, first introduced in 2013 with the paper *Combinatorial Neural Codes from a Mathematical Coding Theory Perspective* [4], by Carina Curto, Vladimir Itskov, Katherine Morrison, Zachary Roth, and Judy L. Walker. This will be another fundamental reference for this work.

This thesis consists of three chapters. In **Chapter 1**, we introduce information theory, addressing fundamental questions such as: How is information mathematically modeled? And even further, how can it be measured? Here, key concepts such as the source of information, the amount of information, and entropy will come into play. These concepts will be applied to a noisy channel, with the ultimate goal of proving Shannon's coding theorem, a cornerstone of coding theory. The primary references used for this chapter include the book *Elements of Information Theory* [3] by Thomas Cover and Joy Thomas, as well as *Codificación de Información* by Carlos Munuera and Juan Tena [10].

Next, in **Chapter 2**, we will study neural codes as presented in the previously mentioned scientific paper [4]. A neural code is essentially the mapping between stimuli and the neural responses they trigger in a studied population of neurons. We will use binary codes, where each codeword represents a neural activity pattern: a nonzero digit indicates a neuron that fires an action potential in response to a stimulus, whereas a zero digit corresponds to a silent neuron that does not transmit an action potential. This new perspective on neural representation of stimuli highlights key aspects. On the one hand, it establishes a clear distinction between codewords—merely a set of vectors without intrinsic meaning—and the actual stimuli received by the brain. On the other hand, the encoding function used is deterministic, meaning that variations in the neural representations of the same stimulus are attributed to the transmission of the response through a noisy channel, which introduces alterations.

Finally, in **Chapter 3**, we will examine the binary asymmetric channel used in the model, following the paper *Parameters of Codes for the Binary Asymmetric Channel* [2], authored by Giuseppe Cotardo and Alberto Ravagnani. What characterizes this channel is that the probability of a digit flipping due to noise depends on its original value. In a binary code, this means there are two distinct probabilities of error—one if the transmitted digit is a zero and another if it is a one. This channel is particularly relevant because, in coding theory, the most commonly used models assume symmetric channels, where the probability of transmission error does not depend on the digit's value. Due to the nature of the brain, the asymmetric channel is more suitable for modeling neural responses, as experimental evidence has shown that it is more likely for a neuron to fail to respond to a stimulus when it was expected to fire than for the opposite to occur.

The significance of this neural representation of stimuli in neuroscience, based on neural codes transmitted through asymmetric channels, is evident in numerous subsequent scientific studies that use this framework for various biological investigations. Some examples of its utility are provided in the final conclusion of this work, **Chapter 4**.

Capítulo 1

Teoría de la información

Vamos a comenzar con algunas nociones de teoría de la información para proporcionar un marco teórico más amplio para los resultados que vamos a presentar a lo largo de este trabajo. Para la redacción de este capítulo se han tomado principalmente estos dos libros de texto como referencia *Codificación de Información* de Carlos Munuera y Juan Tena [10] y *Elements of information theory* de los autores Thomas Cover y Joy Thomas [3].

Empecemos por el principio, el objetivo de la teoría de la información es modelizar matemáticamente la cantidad de información que aporta un mensaje. La idea fundamental es que existe una relación entre la probabilidad de un mensaje y la cantidad de información que aporta. Pensemos por ejemplo, en una noticia que leemos en un periódico si lo que nos cuentan ya lo intuíamos, porque era muy probable que sucediera, podríamos considerar que no nos ha aportado gran cantidad de información esa noticia, mientras que sin embargo, si aquello que leemos es inesperado o era muy improbable que sucediera, podemos considerar que estamos recibiendo una mayor cantidad de información.

1.1. Introducción

Lo primero es concretar el modelo que vamos utilizar y para ello necesitamos que el esquema básico de transmisión de la información este claro: el emisor envía un mensaje previamente codificado a través de un canal, que puede ser por ejemplo espacial o temporal, y el receptor del mensaje deberá decodificarlo para obtener de vuelta la información que contenía. Codificar la información significa reescribirla de una forma diferente, incluso en otro alfabeto distinto, siguiendo unas determinadas reglas. Este proceso puede ser de gran utilidad cuando la información no esta escrita de una forma adecuada para su transmisión.

Definición 1.1. El *alfabeto fuente* $\mathcal{A} = \{a_1, a_2, ..., a_m\}$ es el conjunto de símbolos en los que esta escrita la información legible.

Definición 1.2. El *alfabeto código* $\mathcal{B} = \{b_1, b_2, ..., b_s\}$ es el conjunto de símbolos a los que se traduce la información cuando se somete al proceso de la codificación.

Ambos alfabetos pueden coincidir o no hacerlo.

Definición 1.3. El proceso de *codificar* el alfabeto fuente $\mathcal{A} = \{a_1, a_2, ..., a_m\}$ en el alfabeto código $\mathcal{B} = \{b_1, b_2, ..., b_s\}$ consiste en especificar una aplicación inyectiva $c : \mathcal{A} \longrightarrow \mathcal{P}(\mathcal{B})$. Haciendo corresponder a cada elemento del alfabeto fuente un conjunto de símbolos del alfabeto código.

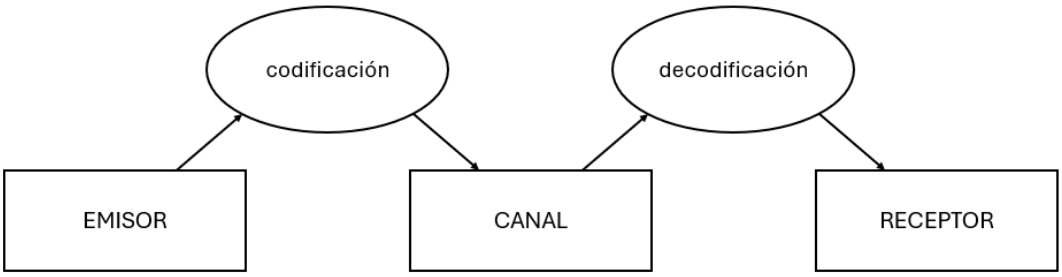

Figura 1.1: Esquema básico de transmisión de la información

Se dice que la imagen de un elemento del alfabeto fuente $c(a_i)$ es la *codificación* de dicho elemento a_i y el subconjunto $\mathcal{C} = Im(c) \subseteq \mathcal{P}(\mathcal{B})$ se denomina el *código* y sus elementos son las *palabras*.

También es procedente definir en general algunas características o propiedades de los códigos que serán de utilidad más adelante.

Definición 1.4. El *tamaño* de un código $\mathcal{C}$ es el número total de palabras de las que se condone, es por tanto el cardinal del conjunto, $|\mathcal{C}|$.

Definición 1.5. Si todas las palabras tienen una longitud fija n, se puede definir la *longitud* del código $\mathcal{C}$ como n también.

Definición 1.6. El *peso de Hamming* $\omega_H(c)$ de una palabra del código $c \in \mathcal{C}$ de longitud n es el número de posiciones distintas de cero en la palabra.

$$\omega_H(x) := |\{1 \leq i \leq n : x_i \neq 0\}|.$$

Definición 1.7. La *distancia de Hamming* entre dos palabras del código $c, c' \in \mathcal{C}$ será el número de posiciones en las que se diferencian una de otra, es decir, definimos la distancia entre c y c' como

$$d_H(c, c') = |\{i \mid x_i \neq y_i, 1 \leq i \leq n\}|.$$

1.2. Fuentes de Información

Para medir la cantidad de información en un mensaje, vamos a trabajar con un objeto abstracto al que vamos a llamar una fuente de información, y que ahora definiremos de forma teórica, pero que sencillamente su cometido es emitir símbolos de entre los de un conjunto finito que será el alfabeto fuente y cada uno de ellos con una cierta probabilidad.

Definición 1.8. Una *distribución de probabilidad* $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ sobre un conjunto de símbolos $\mathcal{A} = \{a_1, a_2 \ldots, a_m\}$, consiste en asignar a cada símbolo a_i una probabilidad $p_i = \text{prob}(a_i)$, de forma que se cumplan las dos condiciones siguientes:

1. $0 \leq p_i \leq 1$, para $i = 1, 2, .., m$,
2. $p_1 + p_2 + \cdots + p_m = 1$.

Definición 1.9. Una *fuente de información* $\mathcal{F} = (\mathcal{A}, \mathcal{P})$ está formada por un alfabeto fuente $\mathcal{A} = \{a_1, a_2 \ldots, a_m\}$ junto con una distribución de probabilidad $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ sobre el conjunto de símbolos que componen ese alfabeto fuente $\mathcal{A}$.

Las probabilidades de cada símbolo no son más que las probabilidades de emisión de dicho carácter por la fuente de información que se esta considerando. Este modelo supone además que la emisión de un símbolo es un evento independiente de todas las emisiones de símbolos anteriores. Es decir que para el cálculo de la probabilidad de emisión de un mensaje m compuesto por la concatenación de r símbolos $m = a_{i_1}a_{i_2}\cdots a_{i_r}$, será necesario multiplicar la probabilidad de emisión de cada uno de ellos.

$$\text{prob}(m) = \text{prob}(a_{i_1}a_{i_2}\cdots a_{i_r}) = p_{i_1}p_{i_2}\cdots p_{i_r}.$$

Supongamos ahora que una fuente de información emite m caracteres determinados $\{a_1, a_2, \ldots, a_m\}$, podríamos pensar que entonces el alfabeto fuente está unívocamente determinado y es el formado por los símbolos que emite la fuente $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$. Sin embargo, tenemos libertad para interpretar que la fuente no emite los símbolos por separado, sino que lo puede hacer de dos en dos o de k en k, entonces tendríamos que estudiar la probabilidad conjunta de las duplas o k-úplas de símbolos. Es decir que de forma más general, podemos considerar como alfabeto fuente $\mathcal{A}^k$, cuyos elementos son k-úplas de elementos de $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$.

Definición 1.10. Dada una fuente de información $\mathcal{F}$ asociada al alfabeto fuente $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ con una distribución de probabilidades $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$, llamaremos *extensión k-ésima de $\mathcal{F}$* a la fuente de información $\mathcal{F}^k$ que tiene como alfabeto fuente $\mathcal{A}^k$ y cuya distribución de probabilidades se obtiene mediante la regla:

$$\text{prob}(a_{i_1}, a_{i_2}, \ldots, a_{i_k}) = p_{i_1}p_{i_2}\ldots p_{i_k}.$$

Esta definición del cálculo de probabilidades es congruente con el hecho de que las emisiones de símbolos son eventos probabilísticamente independientes.

1.3. Medida de la información

Para cuantificar la cantidad de información, fijemos la fuente de información más simple posible $\mathcal{F}_0$ aquella que emite tan solo dos elementos con igual probabilidad, es decir que tenga como alfabeto fuente $\mathcal{A}_0 = \{a_1, a_2\}$ y distribución de probabilidad $\mathcal{P}_0 = \{\frac{1}{2}, \frac{1}{2}\}$. En este caso la información que aporta el símbolo a_1 es la misma que la que aporta el símbolo a_2, ya que sus probabilidades son iguales. Vamos a utilizar esta cantidad como unidad de medida de la información.

Definición 1.11. Llamaremos *bit* a la cantidad de información que contiene cada uno de los dos símbolos de una fuente de información que conste solo de dos símbolos equiprobables.

Para un fuente genérica $\mathcal{F}$, con un alfabeto $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ y distribución de probabilidad asociada al alfabeto $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$. Podemos denotar por $I(a_i)$ la cantidad de información correspondiente al símbolo a_i, pero esta debe depender exclusivamente de la probabilidad p_i asociada a este símbolo a_i. Debido a esta propiedad, es común escribir $I(p_i)$ en lugar de $I(a_i)$, enfatizando que la cantidad de información está determinada únicamente por la probabilidad del símbolo.

Ejemplo 1.12. Tendremos que en el caso de la fuente que hemos usado antes para definir la unidad de medida, $\mathcal{F}_0 = (\mathcal{A}_0, \mathcal{P}_0)$ la más sencilla posible con dos elementos equiprobables.

$$I(p_1) = I(p_2) = 1 \text{ bit}$$

Una propiedad que debe de tener esta función I es que debe ser decreciente, pues la información que aporta un suceso será mayor cuanto más improbable sea que suceda, es decir cuanto menor sea la probabilidad de suceder. Escrito de forma matemática, si $p_i < p_j$, entonces $I(p_i) > I(p_j)$.

Además, dado que la fuente $\mathcal{F}$ no tiene memoria, la emisión de un símbolo no depende de los anteriores, la cantidad de información contenida en dos símbolos es la suma de la información que contiene cada uno de ellos por separado, o lo que es lo mismo se debe verificar que para cada par de símbolos $a_i, a_j \in \mathcal{A}$ que $I(a_i, a_j) = I(a_i) + I(a_j)$, lo que quiere decir que el suceso a_i y el suceso a_j son sucesos independientes.

Proposición 1.13. *Si $f : (0, 1] \to \mathbb{R}$ es una función que verifica:*

1. *f es decreciente en $(0, 1]$*
2. *$f(xy) = f(x) + f(y)$ para cada $x, y \in (0, 1]$,*

entonces existe una constante $k < 0$ tal que $f(x) = k \ln(x)$.

Demostración. La función f que hemos definido es positiva, ya que $f(1) = k \cdot ln(1) = 0$ pues $\ln(1) = 0$ y por (1) es f decreciente. Esto tiene sentido pues si k es cualquier constante estrictamente negativa y $ln(t) < 0$ para todo t en el intervalo $(0, 1)$, luego tendremos una función f estrictamente positiva en $(0, 1)$.
Para estudiar el comportamiento de la función f en $(0, 1)$ vamos a fijar un punto $y \in (0, 1)$, y escribimos k como

$$k = \frac{f(y)}{\ln(y)}.$$

Como $y^0 = 1$ y tomando el límite cuando m tiende a infinito $y^m \to 0$, para cada punto arbitrario $x \in (0, 1)$ y cada $n \in \mathbb{N}$, existe un entero m (dependiente de x y de n) tal que nos permite acotar x^n superior e inferiormente con dos potencias de y de esta manera:

$$y^{m+1} \leq x^n < y^m.$$

Como f es decreciente, al aplicar f sobre estos puntos (todos ellos pertenecen al intervalo $(0, 1)$ donde está definida la función) obtenemos las desigualdades contrarias

$$f(y^m) < f(x^n) \leq f(y^{m+1}),$$

de donde, según la condición (2) del teorema, podemos escribir:

$$mf(y) < nf(x) \leq (m + 1)f(y).$$

Ahora, dividiendo los tres términos de esta desigualdad por $nf(y)$, se tiene:

$$\frac{m}{n} < \frac{f(x)}{f(y)} \leq \frac{m+1}{n}.$$

Razonando de forma análoga con la función $-\ln(x)$, que verifica las mismas propiedades (1) y (2) obtendremos la misma relación:

$$\frac{m}{n} < \frac{\ln(x)}{\ln(y)} \leq \frac{m+1}{n},$$

con lo que:

$$\frac{f(x)}{f(y)} - \frac{\ln(x)}{\ln(y)} < \frac{1}{n},$$

y como $\frac{1}{n}$ puede hacerse arbitrariamente pequeño, tendremos la igualdad entre ambas fracciones:

$$\frac{f(x)}{f(y)} = \frac{\ln(x)}{\ln(y)},$$

o lo que es lo mismo:

$$\frac{f(x)}{\ln(x)} = \frac{f(y)}{\ln(y)} = k.$$

Luego hemos llegado a lo que queríamos demostrar , el cociente $\frac{f(x)}{\ln(x)}$ toma un valor constante $k < 0$ para todo $x \in (0, 1)$, con lo que la función queda definida como $f(x) = k \ln(x)$. □

Como consecuencia inmediata de la proposición, se tiene que la función cantidad de información que aporta un símbolo tiene que ser de la forma $I(p_i) = k \ln(p_i)$, y solo falta determinar la constante k, que dependerá de la unidad de medida. Fijada la unidad como el bit, tenemos que:

$$1 = I\left(\frac{1}{2}\right) = k \ln\left(\frac{1}{2}\right) = -k \ln(2),$$

de donde deducimos que $k = -\frac{1}{\ln(2)}$.

Definición 1.14. La aplicación *cantidad de información medida en bits* se define como la aplicación $I : (0, 1] \longrightarrow \mathbb{R}$ tal que

$$I(p) = -\frac{1}{\ln(2)} \ln(p) = -\log_2 p,$$

y denominamos a $I(p)$ la *cantidad de información* que aporta un símbolo con probabilidad p.

La función I es continua, lo cual se deduce del hecho de que el logaritmo también lo es. Esto implica que pequeños cambios en la probabilidad de un símbolo generan únicamente pequeños cambios en su cantidad de información. Evaluando en la fórmula vemos que se cumple que $I(1) = 0$. Esto significa que la ocurrencia de un suceso seguro, uno con probabilidad igual a 1, no proporciona ninguna información nueva, como ya intuíamos.

Ejemplo 1.15. El código de paridad $\mathcal{C}$ se define como el conjunto de los vectores con coeficientes el cuerpo $\mathbb{F}_2$ que son imagen de la aplicación de codificación $c : \mathbb{F}_2^7 \longrightarrow \mathbb{F}_2^8$ la cuál que envía un vector $(x_1, x_2, \ldots, x_7) \in \mathbb{F}_2^7$ en el elemento:

$$(x_1, x_2, \ldots, x_7, y) \in \mathbb{F}_2^8 \quad \text{donde} \quad y = \sum_{i=1}^{7} x_i \text{ en } \mathbb{F}_2$$

Esto implica que el último bit y, llamado de paridad se establece de modo que la palabra codificada siempre tenga un número par de unos. Si el numero de unos, en los siete primeros dígitos, es par tomará el valor cero, mientras que si este número es impar tomará el valor uno.
También, podemos definir el código como el conjunto

$$\mathcal{C} = \{(x_1, x_2, \ldots, x_8) \quad | \quad \sum_{i=1}^{8} x_i = 0 \mod 2\} \subseteq \mathbb{F}_2^8.$$

Por ejemplo: $c(11000000) = (11000000)$, $\quad c(11111111) = (11111111)$.

Es claro que en este código de paridad, 7 bits de cada palabra del código aportan información, mientras que el octavo solo es un dígito de control, que no aporta nueva información. A estos dígitos también se les conoce como bits de redundancia.

1.4. Entropía

Antes de definir de forma rigurosa la entropía, la vamos a ilustrar este concepto con el siguiente ejemplo:

Ejemplo 1.16. Consideremos dos fuentes de información $\mathcal{F}_1$ y $\mathcal{F}_2$ con el mismo alfabeto fuente $\mathcal{A} = \{A, B, C, D\}$ formado por cuatro símbolos, pero con dos distribuciones de probabilidades diferentes $\mathcal{P}_1, \mathcal{P}_2$ respectivamente.
La primera fuente $\mathcal{F}_1$ produce símbolos equiprobables:

$$\text{prob}(A) = \text{prob}(B) = \text{prob}(C) = \text{prob}(D) = 0{,}25.$$

Mientras que la segunda fuente $\mathcal{F}_2$ los emite siguiendo esta distribución de probabilidades:

$$\text{prob}(A) = 0{,}5 \quad \text{prob}(B) = 0{,}125 \quad \text{prob}(C) = 0{,}125 \quad \text{prob}(D) = 0{,}25.$$

¿Cuántas preguntas esperaríamos hacer para adivinar el símbolo que va a salir en cada fuente? Bien, pues realizando las preguntas de la manera explicada en la Figura 1.2, para la primera fuente $\mathcal{F}_1$ tendremos siempre hacer dos para acertar el símbolo. Mientras que en el caso de la segunda fuente $\mathcal{F}_2$ si las formulamos de la manera explicada en la Figura 1.3, en algunos casos nos bastará con hacer una pregunta, en otros casos haremos dos, llegando en algunas ocasiones hasta hacer tres preguntas. Para esta segunda máquina sería interesante saber cuantas preguntas haríamos de media para poder establecer una comparación entre ambas.

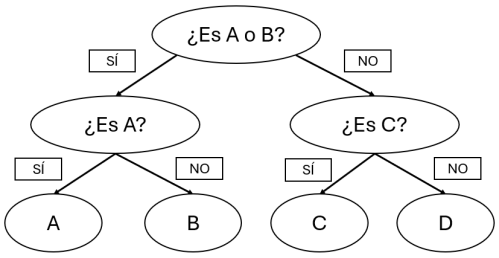

Figura 1.2: Esquema de preguntas para la primera máquina.

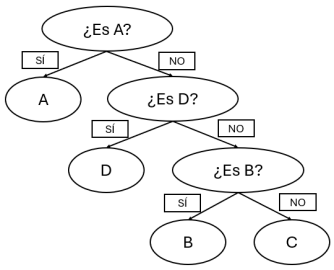

Figura 1.3: Esquema de preguntas para la segunda máquina.

Vamos a explicar esto con una analogía, pensemos que en vez de querer adivinar el siguiente símbolo, lo que queremos es construir las máquinas para recrear las fuentes de información. Imaginemos un experimento físico en el que dejamos caer una pelota desde una cierta altura sobre el medio de una cuña de madera triangular de forma que la pelota tiene igual probabilidad de caer hacia la izquierda o hacia la derecha de esta. Basándonos en hacia que lado cae la pelota podemos generar un símbolo o otro. Es decir en el caso de la primera máquina necesitamos debajo del primer triángulo añadir otra segunda fila con una cuña a cada lado de la primera, para poder generar cuatro resultados equiprobables. En el caso de la segunda máquina, en el primer bote la pelota si cae hacia un lado obtendremos el resultado A que ocurría el 50 % de las veces o si cae hacia el otro tendremos una nuevo triángulo de madera, que dará lugar a dos nuevos resultados o bien a una D que ocurre el 25 % de las veces o a un tercer bote en el que se decidirá entre la C y la D que ocurren el 12,5 %.
Para calcular el número promedio esperado de botes $\mathcal{B}$ que realizará la máquina para generar una letra, multiplicamos el número de botes para alcanzar una letra por su probabilidad:

$$\mathcal{B} = \text{prob}(A) \cdot 1 + \text{prob}(B) \cdot 3 + \text{prob}(C) \cdot 3 + \text{prob}(D) \cdot 2 = 1{,}75.$$

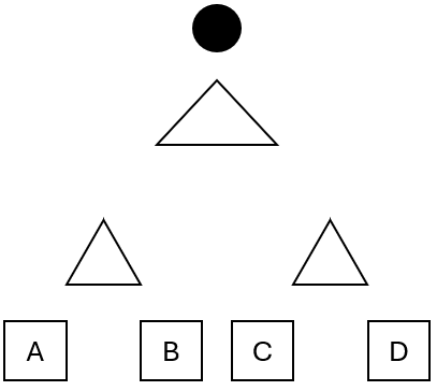

Figura 1.4: Esquema de botes para la generación de los símbolos de la primera máquina.

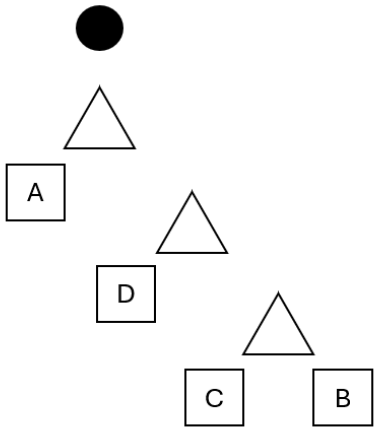

Figura 1.5: Esquema de botes para la generación de los símbolos de la segunda máquina.

Podemos observar en la Figura 1.4 y en la Figura 1.5 que existe una completa similitud con la forma en que teníamos que hacer las preguntas para adivinar el siguiente símbolo que saliera de las máquinas con el número de botes necesarios para generar un símbolo en cada máquina. Luego el número esperado de botes que tendrá que dar la pelota es igual al número esperado de preguntas promedio que tenemos que hacer para adivinar un símbolo. La primera máquina necesita 2 botes para generar un símbolo y hacen falta dos preguntas para adivinar uno, mientras que la segunda máquina necesita 1.75 botes en promedio para generar un elemento y por tanto solo harán falta en promedio 1.75 preguntas para adivinar un símbolo. Lo que significa que la máquina 2 esta produciendo menor cantidad de información pues hay menor incertidumbre en los mensajes que salen de ella.

Vamos a ver que esta medida de la información que produce una máquina o de la incertidumbre promedio en la generación de los símbolos que emite, la definiremos más adelante de forma rigurosa, pero será la entropía de la fuente. Se va a representar por la letra H y la unidad de medida va a ser el bit, que como hemos visto anteriormente es la incertidumbre que genera una fuente con dos resultados equiprobables o la información que produce. En nuestro caso un bit es equivalente a un bote de la pelota hacia un lado o hacia el otro de la cuña de madera; es decir, que en realidad lo que hemos estado calculando era la entropía.

Podemos definir la entropía entonces como:

$$H = \sum_{i=1}^{n} p_i \cdot \#\{\text{Botes hasta el símbolo } a_i.\}$$

El número de botes que tiene que realizar la pelota para llegar al símbolo a_i depende de lo abajo que estemos en el esquema de árbol que hemos dibujado. Si nos fijamos después de un bote solo podemos tener dos resultados diferentes mientras que después de dos botes podremos tener hasta cuatro diferentes, y así sucesivamente. Es decir que si denotamos:
$B = \{\text{Número de botes de la pelota hasta el símbolo } a_i\}$
$R = \{\text{Número de posibles resultados en la fila de } a_i\}$
se tiene que $2^B = R$ o lo que es lo mismo $B = \log_2(R)$. Para calcular el número de resultados en el nivel del árbol de a_i podemos hacer el inverso de la probabilidad del suceso a_i, $R = \frac{1}{p_i}$. Luego

juntando todos estos resultados, tenemos que

$$H = \sum_{i=1}^{n} p_i \cdot B = \sum_{i=1}^{n} p_i \cdot log_2 R = \sum_{i=1}^{n} p_i \cdot log_2 \frac{1}{p_i}.$$

La idea esencial es que la entropía disminuye cuando nos alejamos de una fuente con resultados equiprobables, y por tanto también disminuye el número de preguntas p que debemos hacer para adivinar los símbolos que proceden de la máquina, o dicho de otra manera la incertidumbre en el resultado disminuye.

Definición 1.17. La *entropía* de la fuente $\mathcal{F}$, denotada como $H(\mathcal{F})$, se define por la expresión:

$$H(\mathcal{F}) = \sum_{i=1}^{m} p_i \cdot I(p_i) = \sum_{i=1}^{m} p_i \cdot \log_2 \left(\frac{1}{p_i}\right) = -\sum_{i=1}^{m} p_i \cdot \log_2(p_i),$$

donde:

- m es el número de símbolos en el alfabeto $A = \{a_1, a_2, \ldots, a_m\}$.
- p_i es la probabilidad de ocurrencia del símbolo a_i.
- $I(p_i) = \log\left(\frac{1}{p_i}\right) = -\log(p_i)$ es la cantidad de información asociada a la ocurrencia del símbolo a_i con probabilidad p_i.

Una fuente en el contexto de la probabilidad no es más que una variable aleatoria que toma valores en un conjunto discreto con una ciertas probabilidades.
La entropía $H(\mathcal{F})$ se interpreta como la media ponderada de la cantidad de información que se obtiene al observar un símbolo del alfabeto. Como hemos visto en el ejemplo cuanto más improbable sea un símbolo, más información aporta su aparición. En el caso de que alguna de las probabilidades sea $p_i = 0$, no tiene sentido considerar el cociente $\frac{1}{p_i}$. Pero en este caso ese símbolo no será emitido nunca y la entropía de la fuente será la misma que la de otra fuente $\mathcal{F}'$ exactamente igual en todo lo demás pero sin ese símbolo de probabilidad nula.

La entropía de una fuente será máxima cuando todos los símbolos del alfabeto tienen la misma probabilidad, es decir, cuando la distribución es uniforme. En este caso, la incertidumbre es máxima porque todos los símbolos son igualmente probables. Al contrario, será mínima (igual a cero) cuando uno de los símbolos tiene probabilidad 1 y todos los demás tienen probabilidad 0. En este caso, no hay incertidumbre porque siempre se observará el mismo símbolo.

Ejemplo 1.18. Cálculo de la entropía.
Supongamos que tenemos un alfabeto fuente $\mathcal{A} = \{a_1, a_2, a_3\}$, vamos a ver que ocurre con la entropía dependiendo de la distribución de probabilidades que le asociemos al alfabeto.
La fuente $\mathcal{F}_1$ tendrá la siguiente distribución de probabilidades asociada a su alfabeto fuente $\mathcal{A}$, $\mathcal{P}_1 = \{p_1 = 0{,}6 \quad p_2 = 0{,}3 \quad p_3 = 0{,}1\}$. Luego, el cálculo de su entropía es como sigue:

$$H(\mathcal{F}) = -\left(0{,}6 \cdot \log_2(0{,}6) + 0{,}3 \cdot \log_2(0{,}3) + 0{,}1 \cdot \log_2(0{,}1)\right) = 1{,}2954.$$

Mientras que si tenemos otra fuente $\mathcal{F}_2$ con el mismo alfabeto $\mathcal{A}$ pero con distribución de probabilidades $\mathcal{P}_2 = \{p_1 = 0{,}3 \quad p_2 = 0{,}3 \quad p_3 = 0{,}4\}$, entonces el valor de la entropía será:

$$H(\mathcal{F}) = -\left(0{,}3 \cdot \log_2(0{,}3) + 0{,}3 \cdot \log_2(0{,}3) + 0{,}4 \cdot \log_2(0{,}4)\right) = 1{,}5709.$$

Por último, tenemos una tercera fuente $\mathcal{F}_3$ con el mismo alfabeto $\mathcal{A}$ pero con esta distribución $\mathcal{P}_3 = \{p_1 = 0{,}1 \quad p_2 = 0{,}1 \quad p_3 = 0{,}8\}$, la entropía se calculará como:

$$H(\mathcal{F}) = -\left(0{,}1 \cdot \log_2(0{,}1) + 0{,}1 \cdot \log_2(0{,}1) + 0{,}8 \cdot \log_2(0{,}8)\right) = 0{,}9219.$$

Podemos observar claramente que la entropía es mayor cuanto más cerca estemos del caso en el que los tres eventos son equiprobables, y tiende a cero cuando nos acercamos al caso en que solo un evento tiene probabilidad uno de suceder y los otros dos tienen probabilidad nula.

Es claro que el cero es una cota inferior para la entropía, pero también podemos establecer una cota superior para esta. Veamos primero un lema previo necesario.

Lema 1.19. *Sean $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ y $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$ dos distribuciones de probabilidades sobre el mismo alfabeto fuente $\mathcal{A} = \{a_1, a_2 \ldots, a_m\}$. Entonces*

$$\sum_{i=1}^{m} p_i \log_2\left(\frac{1}{p_i}\right) \leq \sum_{i=1}^{m} p_i \log_2\left(\frac{1}{q_i}\right),$$

se da la igualdad si y sólo si $p_i = q_i$ para todo i, es decir, si ambas son la misma .

Demostración. Sabemos que la función logaritmo neperiano es convexa, es decir, que su gráfica está por debajo de la de su tangente en cualquier punto. En particular, tomando el punto $x_0 = 1$,$y_0 = ln(x_0) = 0$, la recta tangente en el punto será $f(x) = x - 1$ y obtenemos que $\ln(x) \leq x - 1$ y teniéndose la igualdad $\ln(x) = x - 1$ si y sólo si $x = 1$. Por tanto

$$\log_2(x) = \frac{\ln(x)}{\ln(2)} \leq \frac{x-1}{\ln(2)},$$

con lo que

$$\sum_{i=1}^{m} p_i \log\left(\frac{1}{p_i}\right) - \sum_{i=1}^{m} p_i \log\left(\frac{1}{q_i}\right) = \sum_{i=1}^{m} p_i \log\left(\frac{q_i}{p_i}\right) \leq \frac{1}{\ln(2)} \sum_{i=1}^{m} p_i \left(\frac{q_i}{p_i} - 1\right) = 0$$

con igualdad si y sólo si el cociente $q_i/p_i = 1$ para todo i. □

Teorema 1.20. *Si $\mathcal{F}$ es una fuente de información asociada al alfabeto $\mathcal{A}$ con m símbolos y distribución de probabilidades $p_1, p_2, \ldots, p_m$, entonces*

$$0 \leq H(\mathcal{F}) \leq \log(m).$$

Además, $H(\mathcal{F}) = 0$ si y solo si existe un i tal que $p_i = 1$; y $H(\mathcal{F}) = \log(m)$ si y solo si $p_i = \frac{1}{m}$ para todo i.

Demostración. Es evidente que la entropía es no negativa, ya que se calcula como una suma de valores no negativos. En particular, $H(\mathcal{F}) = 0$ si y solo si cada sumando es nulo, es decir, $p_j = 0$ o $\log\left(\frac{1}{p_j}\right) = 0$. Como la primera posibilidad no puede darse para todo j, existirá un único i tal que $\log\left(\frac{1}{p_i}\right) = 0$, lo que implica que $p_i = 1$.

Para la segunda desigualdad, aplicando el Lema 1.19 con $q_i = \frac{1}{m}$ obtenemos:

$$H(\mathcal{F}) = \sum_{i=1}^{m} p_i \log\left(\frac{1}{p_i}\right) \leq \sum_{i=1}^{m} p_i \log(m) = \log(m).$$

Además, según el Lema 1.19, la igualdad se da únicamente cuando $p_i = q_i = \frac{1}{m}$ para todo i. □

Ejemplo 1.21. Sea $\mathcal{F}$ una fuente formada por un alfabeto $\mathcal{A} = \{a_1, a_2\}$ con distribución de probabilidades:

$$\begin{cases} a_1 & \text{con probabilidad } p, \\ a_2 & \text{con probabilidad } 1-p. \end{cases} \quad \text{con } 0 < p < 1, p \in \mathbb{R}$$

Entonces, la entropía de la fuente $H(\mathcal{F})$ está dada por:

$$H(\mathcal{F}) = -p \log p - (1-p) \log(1-p) \stackrel{\text{def}}{=} H(p).$$

En particular, $H(\mathcal{F}) = 1$ bit cuando $p = \frac{1}{2}$. La gráfica de la función $H(\mathcal{F}) = H(p)$ se muestra en la Figura 1.6, esto nos permite visualizar de forma clara algunas de las propiedades de esta función. La entropía es una función convexa y es igual a 0 cuando $p = 0$ o $p = 1$. Esto tiene sentido, ya que cuando $p = 0$ o $p = 1$, la variable no es aleatoria y no existe incertidumbre. De manera similar, la incertidumbre es máxima cuando $p = \frac{1}{2}$, lo que también corresponde al valor máximo de la entropía, cuando ambos sucesos son equiprobables.

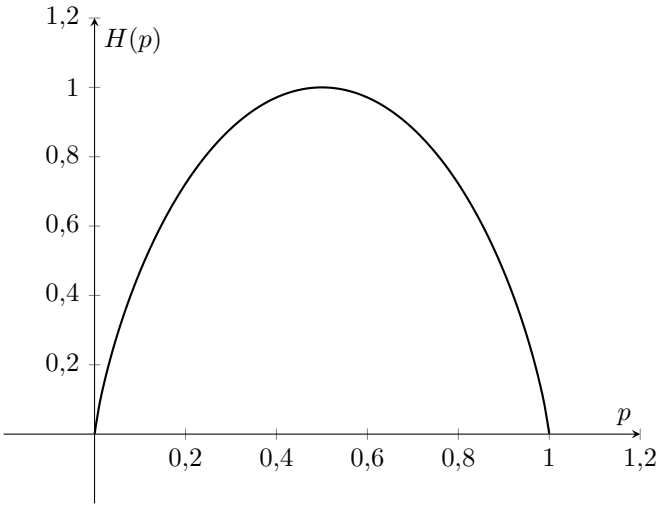

Figura 1.6: Distribución de la entropía según los valores que tome la probabilidad, para un alfabeto fuente compuesto por dos símbolos.

La entropía de una fuente extendida también puede calcularse y se hace en función de la entropía de la fuente original. Intuitivamente como cada elemento del alfabeto $\mathcal{A}^k$ está formado por k símbolos de $\mathcal{A}$, el promedio de información que contiene cada símbolo en el alfabeto extendido debe de ser k veces mayor.

Proposición 1.22. *Para cada fuente de información $\mathcal{F}$ y cada entero positivo k, se verifica que*

$$H(\mathcal{F}^k) = kH(\mathcal{F}).$$

Demostración. Vamos a demostrarlo por inducción sobre k.
Si $k = 1$, el resultado es evidente.
Supongamos, por hipótesis de inducción, que $H(\mathcal{F}^{k-1}) = (k-1)H(\mathcal{F})$.
Entonces, para k, tenemos:

$$H(\mathcal{F}^k) = \sum_{i_1,\ldots,i_k=1}^{m} p_{i_1} \cdots p_{i_k} \log\left(\frac{1}{p_{i_1} \cdots p_{i_k}}\right).$$

Descomponiendo el logaritmo:

$$H(\mathcal{F}^k) = \sum_{i_1,\ldots,i_k=1}^{m} p_{i_1}\cdots p_{i_k}\left(\log\left(\frac{1}{p_{i_1}\cdots p_{i_{k-1}}}\right)+\log\left(\frac{1}{p_{i_k}}\right)\right).$$

Utilizando a propiedad distributiva dividimos el sumatorio en dos:

$$= \sum_{i_1,\ldots,i_{k-1}=1}^{m}\sum_{i_k=1}^{m} p_{i_1}\cdots p_{i_k}\log\left(\frac{1}{p_{i_1}\cdots p_{i_{k-1}}}\right)+$$

$$+\sum_{i_k=1}^{m}\sum_{i_1,\ldots,i_{k-1}=1}^{m} p_{i_1}\cdots p_{i_k}\log\left(\frac{1}{p_{i_k}}\right).$$

Agrupamos todo lo que depende de la última variable k y lo que depende de las otras primeras $k-1$:

$$= \sum_{i_1,\ldots,i_{k-1}=1}^{m} p_{i_1}\cdots p_{i_{k-1}}\log\left(\frac{1}{p_{i_1}\cdots p_{i_{k-1}}}\right)\sum_{i_k=1}^{m} p_{i_k}+$$

$$+\sum_{i_k=1}^{m} p_{i_k}\log\left(\frac{1}{p_{i_k}}\right)\sum_{i_1,\ldots,i_{k-1}=1}^{m} p_{i_1}\cdots p_{i_{k-1}}.$$

Utilizando que estas sumas de probabilidades valen 1 y la hipótesis de inducción, podemos simplificar la expresión:

$$\sum_{i_k=1}^{m} p_{i_k}=1,\quad \sum_{i_1,\ldots,i_{k-1}=1}^{m} p_{i_1}\cdots p_{i_{k-1}}=1,$$

$$H(\mathcal{F}^k)=H(\mathcal{F}^{k-1})+H(\mathcal{F})=(k-1)H(\mathcal{F})+H(\mathcal{F})=kH(\mathcal{F}).$$

□

Para lo que resta de capítulo vamos a necesitar también los conceptos de entropía conjunta y la entropía condicionada que se pueden definir para dos variables aleatorias de la siguiente manera, y más adelante los aplicaremos a nuestro caso de fuentes de información.

Definición 1.23. La *entropía conjunta* de dos variables aleatorias X e Y, denotada como $H(X,Y)$, se define como:

$$H(X,Y) = -\sum_{x\in X}\sum_{y\in Y}\text{prob}(x,y)\log_2(\text{prob}(x,y))$$

donde $\text{prob}(x,y)$ es la probabilidad conjunta de los símbolos $x\in X$ y $y\in Y$, es decir que la variable aletoria X tome el valor x y la variable aleatoria Y tome el valor y simultáneamente.

Esta entropía conjunta representa la incertidumbre combinada sobre la observación de ambas variables aleatorias.

Definición 1.24. La *entropía condicional* de una variable a aleatoria X dada otra variable aleatoria Y, denotada como $H(X|Y)$, se define como:

$$H(X|Y) = -\sum_{x\in X}\sum_{y\in Y}\text{prob}(x,y)\log_2(\text{prob}(x|y))$$

donde $\text{prob}(x|y)$ es la probabilidad condicionada del suceso x sabiendo que se ha producido el suceso y.

La entropía condicional $H(X|Y)$ mide la incertidumbre que resta en una variable aleatoria X después de conocer el resultado de la variable Y. Si están fuertemente correlacionadas, el conocimiento de Y reduce significativamente la incertidumbre sobre X. En cambio, si son independientes, conocer Y no aporta información sobre X y la entropía condicional es máxima.
En resumen, la entropía conjunta $H(X,Y)$ refleja la incertidumbre combinada de ambas fuentes, mientras que la entropía condicional $H(X|Y)$ indica cuánta incertidumbre resta en una fuente conociendo lo que ha emitido otra.

1.5. Canal con ruido

Los mensajes que produce una fuente de información han de enviarse a través de algún canal a su destinatario. La realidad es que suponer que los mensajes se envían con exactitud a través del canal no es realista, ya que en los canales pueden producirse interferencias que modifican el mensaje original que se pretendía enviar. Este fenómeno es lo que se conoce como el ruido del canal. Los códigos diseñados para transmitir información a través de estos canales de denominan *códigos correctores de errores*, su finalidad es que aunque se produzcan perturbaciones en el mensaje original, ser capaces de detectarlas y corregirlas, es decir de recuperar la información perdida durante la transmisión del mensaje.

En un canal afectado por ruido, los mensajes pueden sufrir diversas alteraciones. Dependiendo de la naturaleza de estas alteraciones, podemos distinguir dos tipos. Los errores que son símbolos del alfabeto recibidos que difieren de los enviados. El receptor del mensaje no puede diferenciar un carácter transmitido correctamente de un error. Y un segundo tipo, que son los borrones, símbolos que no son legibles o que el receptor no puede interpretar. El receptor sí que los detecta pues no son otros caracteres que formen parte del alfabeto.
Aunque para simplificar, nos referiremos a ambos como errores, podemos interpretar los borrones como errores cuya posición es conocida.

Definición 1.25. Un *canal* es una terna $(\mathcal{A}, \mathcal{S}, \mathcal{P})$ donde:

- $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ es el alfabeto fuente, es decir el conjunto de símbolos que se van a enviar por el canal.
- $\mathcal{S} = \{s_1, s_2, \ldots, s_h\}$ es el conjunto de símbolos que salen del canal es decir los que conforman el alfabeto de salida.
- $\mathcal{P} = \{\ \text{prob}(s_i|a_j) \quad | \quad i = 1, \ldots, h \text{ y } j = 1, \ldots, m\}$ el conjunto de las probabilidades condicionadas de los símbolos del alfabeto salida a cada uno de los símbolos del alfabeto de entrada.

Aclaremos algunos aspectos de la definición.
El cardinal de $\mathcal{A}$ es m, es decir $|\mathcal{A}| = m$.
Concretamente $\text{prob}(s_i|a_j)$ representa la probabilidad de recibir el símbolo s_i cuando se sabe que se ha emitido el símbolo a_j.
El alfabeto de entrada es claro que esta formado por los elementos del alfabeto de una fuente de información, y para el alfabeto de salida existen dos posibilidades:

- Si no hay borrones, entonces $\mathcal{S} = \mathcal{A}$ y $s_i = a_i$ para $i = 1, \ldots, m$, en este caso el cardinal de ambos conjuntos coincide $|\mathcal{S}| = m$.
- Si hay borrones, tenemos que incluir otro carácter distinto de todos los del alfabeto fuente, $\mathcal{S} = \mathcal{A} \cup \{\perp\}$ y $s_i = a_i$ para $i = 1, \ldots, m$, y $s_{m+1} = \perp$ y entonces $|\mathcal{S}| = m + 1$.

Aunque $a_i = s_i$ para todo $i \in \{1, 2, .., m\}$ en ambos casos, vamos a mantener las dos notaciones a_i y s_i para aclarar cuándo nos referimos a un símbolo como entrada y cuándo como salida.

El modelo de canal que vamos a considerar es discreto y sin memoria: es decir que emite una señal por unidad de tiempo, además la transmisión de cada señal no afecta a las siguientes, y cada señal que se emite proporciona siempre una palabra que se recibe.

Supongamos que tenemos una fuente de información $\mathcal{F}$ con alfabeto fuente $\mathcal{A} = \{a_1, a_2, ..., a_m\}$ y distribución de probabilidades $\mathcal{P}_a = \{\text{prob}(a_1), \text{prob}(a_2), \ldots, \text{prob}(a_m)\}$. Vamos a estudiar la distribución de probabilidad del alfabeto de salida, $\mathcal{P}_s = \{\text{prob}(s_1), \text{prob}(s_2), \ldots, \text{prob}(s_m)\}$. Veremos que esta queda determinada completamente por la distribución de probabilidades del alfabeto de entrada $\mathcal{P}_a$ y el conjunto de probabilidades condicionadas que definen el canal $\mathcal{C}$, por la regla:

$$\text{prob}(s_i) = \sum_{j=1}^{m} \text{prob}(a_j) \cdot \text{prob}(s_i|a_j).$$

Es decir que, la probabilidad de que salga un símbolo s del canal es la suma de las probabilidades de que salga s condicionada a que haya sido emitido cada uno de los símbolos del alfabeto fuente, por la probabilidad de emisión de cada uno de ellos.

Ejemplo 1.26. Distribución de probabilidad del alfabeto de salida.
Sea una fuente $\mathcal{F}$ que emite dos símbolos $\{0, 1\}$ con distribución de probabilidad:

$$\mathcal{P}_a = \{\text{prob}(0) = 0{,}7,\ \text{prob}(1) = 0{,}3\}.$$

Sea $\{\mathcal{A}, \mathcal{S}, \mathcal{P}\}$ un canal con ruido que transmite los símbolos que emite la fuente de información $\mathcal{F}$. Tendrá como alfabeto de entrada a $\mathcal{A} = \{a_1, a_2\} = \{0, 1\}$, como alfabeto de salida, si suponemos que no se producen borrones, tendrá al conjunto $\mathcal{S} = \{s_1, s_2\} = \{0, 1\}$, y las siguientes probabilidades condicionadas:

$$\text{prob}(s_1|a_1) = \text{prob}(0|0) = 0{,}9, \quad \text{prob}(s_2|a_1) = \text{prob}(1|0) = 0{,}1$$

$$\text{prob}(s_1|a_2) = \text{prob}(0|1) = 0{,}2, \quad \text{prob}(s_2|a_2) = \text{prob}(1|1) = 0{,}8.$$

Esto significa que si se emite un "0", el receptor lo recibirá correctamente con un 90 % de probabilidad, pero con un 10 % de probabilidad recibirá un "1"debido al ruido. De manera similar, si se emite un "1", será recibido correctamente con una probabilidad del 80 %, y con un 20 % de probabilidad será recibido incorrectamente como un "0".

Vamos a calcular ahora la distribución de probabilidad $\mathcal{P}_s$ que inducen la distribución de probabilidades $\mathcal{P}_a$ y las probabilidades condicionadas del canal.

La probabilidad de recibir un símbolo s_1 en la salida será:

$$\text{prob}(s_1) = \sum_{j=1}^{2} \text{prob}(a_j) \cdot \text{prob}(s_1|a_j).$$

$$\text{prob}(s_1) = (0{,}7 \cdot 0{,}9) + (0{,}3 \cdot 0{,}2) = 0{,}63 + 0{,}06 = 0{,}69.$$

Por lo tanto, la probabilidad de recibir un "0" es del 69 %.
De manera similar, la probabilidad de recibir un símbolo s_2 se calcula como:

$$\text{prob}(s_2) = (0{,}7 \cdot 0{,}1) + (0{,}3 \cdot 0{,}8) = 0{,}07 + 0{,}24 = 0{,}31.$$

Por lo tanto, la probabilidad de recibir un "1" es del 31 %.

$$\mathcal{P}_s = \{\text{prob}(s_1) = 0{,}69,\ \text{prob}(s_2) = 0{,}31\}.$$

1.6. Capacidad de un canal

Cuando de un canal con ruido recibimos un elemento s_i, realmente no podemos saber si el símbolo que se envió fue ese mismo $a_i = s_i$. Es decir, existe una cierta pérdida de información que nos gustaría poder cuantificar para evitar usar canales cuya pérdida de información sea tan elevada que resulten inútiles para la transmisión de información a través de ellos. El objetivo de esta sección por tanto es medir esa pérdida de información.

Vamos a calcular lo primero la probabilidad conjunta de a_j y s_i, o lo que es lo mismo la probabilidad de que a_j sea el símbolo de entrada y s_i el símbolo de salida, y lo vamos a denotar por $\text{prob}(a_j \rightsquigarrow s_i)$, que visualmente es más claro que $\text{prob}(a_j, s_i)$. Utilizando las ley de probabilidad de Bayes:

$$\text{prob}(a_j \rightsquigarrow s_i) = \text{prob}(a_j) \cdot \text{prob}(s_i|a_j) = \text{prob}(s_i) \cdot \text{prob}(a_j|s_i).$$

Es decir que la probabilidad conjunta de ambos, es la probabilidad de uno de ellos por la probabilidad del otro condicionada a que ha salido el primero.

Ahora supongamos que se ha recibido un cierto símbolo s_i. Nos interesa saber entonces que símbolo a_j del alfabeto fuente es más probable que haya sido enviado. Esta probabilidad es fácil de calcular a partir de las relaciones anteriores:

$$\text{prob}(a_j|s_i) = \frac{\text{prob}(a_j) \cdot \text{prob}(s_i|a_j)}{\text{prob}(s_i)}.$$

Entonces observemos que hemos obtenido una nueva distribución de probabilidades en el alfabeto de entrada $\mathcal{A}$ para cada símbolo del alfabeto de salida s_i, en particular condicionada por haberlo recibido.

Nota 1.27. Se ha calculado en el apartado anterior la entropía de una fuente $H(\mathcal{F})$ pero está siempre tendrá asociado un cierto alfabeto $\mathcal{A}$, así que vamos a denotarla ahora por $H(\mathcal{A})$ directamente que es más conveniente, ya que también calcularemos la entropía del alfabeto de salida del canal.

Con esta nueva notación, escribimos que la entropía del alfabeto de entrada $\mathcal{A}$ con la distribución de probabilidades inducida por el elemento s_i es:

$$H(\mathcal{A}|s_i) = \sum_{j=1}^{m} \text{prob}(a_j|s_i) \log\left(\frac{1}{\text{prob}(a_j|s_i)}\right).$$

Pero esto tan solo es para un símbolo concreto del alfabeto de salida y nos va interesar una medida que releje la incertidumbre media sobre la entrada una vez conocida la salida. Necesitamos realizar para ello la media ponderada de estas cantidades $H(\mathcal{A}|s_i)$ sobre todos los símbolos de salida s_i y la denotaremos por $H(\mathcal{A}|\mathcal{S})$:

$$H(\mathcal{A}|\mathcal{S}) = \sum_{i=1}^{m} \text{prob}(s_i) H(A|s_i) = \sum_{i=1}^{m} \text{prob}(s_i) \sum_{j=1}^{m} \text{prob}(a_j|s_i) \log\left(\frac{1}{\text{prob}(a_j|s_i)}\right).$$

Otra forma equivalente de escribir $H(\mathcal{A}|\mathcal{S})$ es utilizando la definición de $\text{prob}(a_j \rightsquigarrow s_i)$:

$$H(\mathcal{A}|\mathcal{S}) = \sum_{i=1}^{m} \sum_{j=1}^{m} \text{prob}(a_j \rightsquigarrow s_i) \log\left(\frac{1}{\text{prob}(a_j|s_i)}\right).$$

Definición 1.28. Llamaremos *entropía de la entrada condicionada por la salida* a la cantidad:

$$H(A|S) = \sum_{i=1}^{m}\sum_{j=1}^{m} \text{prob}(a_j \rightsquigarrow s_i) \log\left(\frac{1}{\text{prob}(a_j|s_i)}\right),$$

medida en bits por símbolo.

La incertidumbre sobre la entrada al canal antes de conocer la salida es $H(\mathcal{A})$, es decir, la cantidad de información necesaria para describir la entrada sin conocer nada de la salida. Sin embargo, una vez que se conoce la salida, la incertidumbre sobre la entrada disminuye, y la nueva incertidumbre es $H(\mathcal{A}|\mathcal{S})$, que mide cuánta información falta para determinar la entrada sabiendo el símbolo que salió del canal. Por lo tanto, la diferencia $H(\mathcal{A}) - H(\mathcal{A}|\mathcal{S})$ mide cuánta incertidumbre hemos eliminado sobre la entrada al conocer la salida. En otras palabras, esta diferencia representa la información que obtenemos sobre la entrada al observar la salida.

Definición 1.29. La *información mutua* entre la entrada $\mathcal{A}$ y la salida $\mathcal{S}$ es:

$$I(\mathcal{A}|\mathcal{S}) = H(\mathcal{A}) - H(\mathcal{A}|\mathcal{S})$$

medida en bits por símbolo.

De forma similar, podemos medir la incertidumbre sobre la salida antes de conocer la entrada, $H(\mathcal{S})$, y la incertidumbre restante sobre la salida al conocer la entrada, $H(\mathcal{S}|\mathcal{A})$. La información mutua puede escribirse también como:

$$I(\mathcal{S}|\mathcal{A}) = H(\mathcal{S}) - H(\mathcal{S}|\mathcal{A}).$$

Hasta ahora, puede parecer que estas dos formas de expresar la información mutua son distintas, pero en realidad son equivalentes. Esto quiere decir que la información que obtenemos sobre la entrada al observar la salida es exactamente la misma que la que obtenemos sobre la salida al conocer la entrada.

Proposición 1.30. *La información mutua es una aplicación simétrica.*

$$I(\mathcal{A}|\mathcal{S}) = I(\mathcal{S}|\mathcal{A}).$$

Lo que justifica el término mutua de la definición.

Demostración. Vamos a desarrollar las expresiones usando las probabilidades conjuntas y condicionales. Para abreviar en la notación, estamos trabajando en bits luego los logaritmos con los que trabajaremos son en base dos aunque no se especifique y que lo que hemos denotado como $\text{prob}(a_j \rightsquigarrow s_i)$, no es más que la probabilidad conjunta de ambos símbolos y que en vez de denotar por $\text{prob}(a)$ denotaremos sencillamente por $P(a)$ como es también habitual.

$$I(\mathcal{A}|\mathcal{S}) = H(\mathcal{A}) - H(\mathcal{A}|\mathcal{S}) = \sum_{j=1}^{m} P(a_j)\log\frac{1}{P(a_j)} - \sum_{i=1}^{m+1}\sum_{j=1}^{m} P(a_j,s_i)\log\frac{1}{P(a_j|s_i)}$$

$$= \sum_{i=1}^{m+1}\sum_{j=1}^{m} P(a_j,s_i)\log\frac{1}{P(a_j)} - \sum_{i=1}^{m+1}\sum_{j=1}^{m} P(a_j,s_i)\log\frac{P(s_i)}{P(a_j,s_i)}$$

$$= \sum_{i=1}^{m+1}\sum_{j=1}^{m} P(a_j,s_i)\log\frac{P(a_j,s_i)}{P(a_j)P(s_i)}.$$

Análogamente se puede repetir el procedimiento intercambiando los papeles de $\mathcal{A}$ y de $\mathcal{S}$.

$$I(\mathcal{S}|\mathcal{A}) = H(\mathcal{S}) - H(\mathcal{S}|\mathcal{A}) = \sum_{i=1}^{m+1} P(s_i) \log \frac{1}{P(s_i)} - \sum_{j=1}^{m} \sum_{i=1}^{m+1} P(s_i, a_j) \log \frac{1}{P(s_i|a_j)}$$

$$= \sum_{j=1}^{m} \sum_{i=1}^{m+1} P(s_i, a_j) \log \frac{1}{P(s_i)} - \sum_{j=1}^{m} \sum_{i=1}^{m+1} P(s_i, a_j) \log \frac{P(a_j)}{P(s_i, a_j)}$$

$$= \sum_{j=1}^{m} \sum_{i=1}^{m+1} P(s_i, a_j) \log \frac{P(s_i, a_j)}{P(s_i)P(a_j)}.$$

Nótese que las dos expresiones a las que hemos llegado son idénticas. □

La información mutua no solo depende del canal, sino también de la distribución de probabilidad del alfabeto de entrada $\mathcal{P}_a = \{p_1 = \text{prob}(a_1), p_2 = \text{prob}(a_2), \ldots, p_m = \text{prob}(a_m)\}$.

Definición 1.31. La *capacidad del canal* C, es el valor máximo de $I(\mathcal{A}|\mathcal{S})$ sobre todas las posibles distribuciones de probabilidad en el alfabeto de entrada $\mathcal{A}$:

$$C = \text{máx}\{ \quad I(\mathcal{A}|\mathcal{S}) \quad | \quad p_1, p_2, \ldots, p_m \in [0,1] \text{ y } p_1 + p_2 + \cdots + p_m = 1\}$$

Este concepto representa la cantidad máxima de información que puede transmitir un canal por unidad de tiempo, y efectivamente se alcanza pues estamos estudiando los extremos de una función continua en un conjunto compacto de $\mathbb{R}^m$ y por el teorema de Weierstrass [6] existe el máximo y el mínimo absolutos y además se alcanzan en puntos del conjunto.

Ejemplo 1.32. Continuemos con el ejemplo 1.26
Queremos calcular ahora la información mutua del canal, que sabemos que se puede calcular utilizando esta fórmula:

$$I(\mathcal{S}|\mathcal{A}) = H(\mathcal{S}) - H(\mathcal{S}|\mathcal{A}).$$

Luego para ello necesitamos calcular la entropía de $\mathcal{S}$ y la entropía condicionada de el alfabeto de salida, conociendo la entrada del canal $H(\mathcal{S}|\mathcal{A})$.

$$\begin{aligned} H(\mathcal{S}) &= -\sum_{i=1}^{2} \text{prob}(s_i) \log_2 \text{prob}(s_i) \\ &= -\left(0{,}69 \log_2(0{,}69) + 0{,}31 \log_2(0{,}31)\right) \\ &\approx 0{,}87284 \,\text{bits}. \end{aligned}$$

$$H(\mathcal{S}|\mathcal{A}) = \sum_{j=1}^{2} \text{prob}(a_j) H(\mathcal{S}|\mathcal{A} = a_j)$$

Calculamos $H(\mathcal{S}|\mathcal{A} = a_1)$ y $H(\mathcal{S}|\mathcal{A} = a_2)$ para sustituir después en la expresión:

$$H(\mathcal{S}|\mathcal{A} = a_1) = -\left(0{,}9 \log_2(0{,}9) + 0{,}1 \log_2(0{,}1)\right) \approx 0{,}46900 \,\text{bits},$$

$$H(\mathcal{S}|\mathcal{A} = a_2) = -\left(0{,}2 \log_2(0{,}2) + 0{,}8 \log_2(0{,}8)\right) \approx 0{,}72193 \,\text{bits}.$$

$$H(\mathcal{S}|\mathcal{A}) = (0{,}7 \cdot 0{,}46900) + (0{,}3 \cdot 0{,}72193) \approx 0{,}54488 \,\text{bits}.$$

La información mutua tiene el valor:

$$I(A; \mathcal{S}) = H(\mathcal{S}) - H(\mathcal{S}|\mathcal{A}) \approx 0{,}87284 - 0{,}54488 \approx 0{,}32796 \,\text{bits}.$$

Si $\mathcal{C}$ es un código de longitud n sobre el alfabeto $\mathcal{A}$ con un total$q$ símbolos, entonces puede poseer a lo más q^n palabras. Por tanto, todo código en bloque con m palabras tiene longitud al menos $\lceil \log_q(m) \rceil$. De hecho, podemos considerar que de los n símbolos de cada palabra del código $\mathcal{C}$, $\log_q(m)$ contienen la información y el resto son únicamente de control. Esto motiva la siguiente definición:

Definición 1.33. Si $\mathcal{C}$ es un código de cuyas palabras tienen todas longitud n y con un número total de palabras $|\mathcal{C}| = m$ palabras sobre un alfabeto fuente $\mathcal{A}$ con $|\mathcal{A}| = q$ elementos, llamaremos *tasa de transmisión de información* de $\mathcal{C}$ al cociente

$$R(\mathcal{C}) = \frac{\log_q(m)}{n}.$$

1.7. Teorema de Shannon

Claude Shannon, en su trabajo seminal de 1948 titulado *"A Mathematical Theory of Communication"*, demostró que se puede alcanzar siempre la capacidad máxima de transmisión de información de un canal si los códigos que se emplean son suficientemente largos. En él proporcionó una demostración probabilística que se basa en la idea de usar códigos aleatorios pero que sin embargo no es constructiva, no detalla la forma de obtener dichos códigos. Para la demostración que se presenta en este trabajo se ha utilizado el libro de texto *Codes and Cryptography* [14].

Definición 1.34. Dado un código binario $\mathcal{C}$ de longitud n, llamaremos *probabilidad de error* del código, denotada como $\text{prob}_{err}(\mathcal{C})$, a la probabilidad media de que una palabra sea decodificada de forma errónea por el receptor cuando la información a sido codificada utilizando el código $\mathcal{C}$.

El significado de que una palabra sea decodificada de forma errónea es que al pasar por el canal ruidoso se han producido más errores de los que se permiten para poder decodificar de forma correcta y por tanto al decodificar la palabra recibida hemos obtenido una diferente a la original. Si suponemos que todas las palabras código son enviadas con la misma frecuencia, entonces la probabilidad de error será la media de la probabilidad de error de cada palabra del código:

$$\text{prob}_{err}(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \text{prob(error habiendo enviado } c).$$

Teorema 1.35 (**Teorema de codificación de un canal ruidoso de Shannon**). *Dado un canal binario simétrico con capacidad C, y una tasa de transmisión arbitraria pero menor que la capacidad del canal $0 < R < C$. Si se tiene que M_n es una sucesión de números naturales que cumplen que para todo n que, $1 \leq M_n \leq 2^{Rn}$ y $\varepsilon > 0$ es cualquier cantidad positiva, entonces en esas condiciones existe una sucesión de códigos $\mathcal{C}_n$ que tienen cada uno M_n palabras con una longitud de n dígitos y también existe un entero $n_0 \in \mathbb{N}$ tal que para todo $n \geq n_0$, se cumple que la $prob_{err}(\mathcal{C}_n) \leq \varepsilon$*

Para la prueba de este resultado vamos a necesitar dos desigualdades que vamos a exponer a continuación:

Proposición 1.36 (La desigualdad de la cola). *Para cualquier $\lambda \in \mathbb{R}$, con $0 \leq \lambda \leq \frac{1}{2}$,*

$$\sum_{k=0}^{\lfloor \lambda n \rfloor} \binom{n}{k} \leq 2^{nh(\lambda)},$$

donde h es la siguiente función $h(\lambda) = -[\lambda \log \lambda + (1-\lambda)\log(1-\lambda)]$.

Demostración. Para facilitar las cosas, y sin pérdida de generalidad, podemos asumir que el producto λn es un número entero. Entonces podemos escribir

$$1 = [\lambda + (1-\lambda)]^n = \sum_{k=0}^{\lambda n} \binom{n}{k} \lambda^k (1-\lambda)^{n-k}$$

$$\geq (1-\lambda)^n \sum_{k=0}^{\lambda n} \binom{n}{k} \left(\frac{\lambda}{1-\lambda}\right)^{\lambda n} = \lambda^{\lambda n}(1-\lambda)^{n(1-\lambda)} \sum_{k=0}^{\lambda n} \binom{n}{k}.$$

Por lo tanto, haciendo uso de la propiedad logarítmica $a = 2^{log_2(a)}$ y de la definición de la función h se obtiene la desigualdad del enunciado.

$$\sum_{k=0}^{\lambda n} \binom{n}{k} \leq \lambda^{-\lambda n}(1-\lambda)^{-n(1-\lambda)}$$

$$\sum_{k=0}^{\lambda n} \binom{n}{k} \leq 2^{log\lambda^{-n\lambda} + log(1-\lambda)^{-n(1-\lambda)}} = 2^{-n(\lambda log\lambda + (1-\lambda)log(1-\lambda))}.$$

□

Recordemos si tenemos una variable aleatoria discreta X, el valor esperado $\mu = \mathbb{E}[X]$ se define como la media ponderada de los valores que puede tomar por sus probabilidades, mientras que la varianza $\text{Var}[X]$ es el valor esperado de X menos su valor esperado μ todo ello al cuadrado [9].

$$\mathbb{E}[X] = \sum x_i P(x_i) \quad \text{Var}[X] = \mathbb{E}[(X-\mu)^2]$$

Donde x_i representan todos los posibles valores que puede tomar la variable aleatoria distreta X. Antes de introducir la desigualdad de Chebyshev que vamos a utilizar en la prueba del teorema, es necesario tambien introducir la desigualdad de Markov.

Proposición 1.37 (Desigualdad de Markov)**.** *Sea X una variable aleatoria no negativa y $a > 0$ un número real. Entonces, se cumple que:*

$$P(X \geq a) \leq \frac{\mathbb{E}[X]}{a}.$$

Demostración.

$$a \cdot P(X \geq a) = a \cdot \sum_{x:x \geq a} P(X = x).$$

Dado que $x \geq a$, podemos escribir:

$$a \cdot P(X \geq a) \leq \sum_{x:x \geq a} x \cdot P(X = x).$$

Al extender la suma al soporte completo de X, y pasar la constante a dividiendo al otro término obtenemos la desigualdad buscada:

$$a \cdot P(X \geq a) \leq \sum_{x:x \geq a} x \cdot P(X = x) \leq \sum_{x} x \cdot P(X = x) = \mathbb{E}[X].$$

□

Es importante notar que la condición $X \geq 0$ es utilizada en la tercera línea de la demostración para garantizar que la desigualdad se mantenga.

Proposición 1.38 (Desigualdad de Chevychev)**.** *Sea X una variable aleatoria, y sea $A > 0$. Entonces,*

$$P\left(|X - \mu| \geq A\right) \leq \frac{\mathrm{Var}[X]}{A^2}.$$

Demostración. Sea $\mu = \mathbb{E}[X]$ el valor esperado de X. Si definimos la variable aleatoria Y como $Y = (X - \mu)^2 = X^2 - 2\mu X + \mu^2$. Entonces,

$$\mathbb{E}[Y] = \mathbb{E}[X^2] - 2\mu\mathbb{E}[X] + \mu^2 = \mathbb{E}[X^2] - 2\mu^2 + \mu^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \mathrm{Var}[X].$$

Dado que $Y \geq 0$, podemos aplicar la desigualdad de Markov a esta variable. Tenemos

$$P\left(|X - \mu| \geq A\right) = P\left(Y \geq A^2\right) \leq \frac{\mathbb{E}[Y]}{A^2} = \frac{\mathrm{Var}[X]}{A^2}.$$

□

La demostración de este teorema se basa en la siguiente idea:
Si fijamos un entero n y trabajamos con códigos binarios en $\mathbb{F}_2^n$. Supongamos que estamos intentando encontrar un código con cardinal M. Elegiremos estas palabras del código c_i para todo $i = 1, 2, ..., M$ mediante el método siguiente. Seleccionando un vector al azar de entre los del espacio $\mathbb{F}_2^n$, de forma independiente para cada $i = 1, 2, ..., M$. Este proceso es denominado como *codificación aleatoria*. Después decodificaremos, fijando un número $r > 0$ y si denotamos por $B_r(y)$ a la bola de radio r alrededor de $y \in \mathbb{F}_2^n$, es decir,

$$B_r(y) = \{z \in \mathbb{F}_2^n \mid d(y, z) \leq r\}.$$

Entonces, si y es el vector recibido, decodificamos y como la palabra código c_j si c_j es la única palabra código dentro de la bola $B_r(y)$; en caso contrario, decodificamos y como alguna palabra código arbitraria.

Procedamos ahora sí con la demostración detallada del Teorema 1.35.

Demostración. Sea $c \in \mathcal{C}$ una palabra del código que se envía a través del canal ruidoso, sea Y el vector recibido una vez que la palabra atraviesa el canal. Denotemos por E el suceso "Ocurre un error en la decodificación", o lo que es lo mismo que Y es decodificado como una palabra código distinta de la enviada c, a esta otra palabra del código la denotaremos por c'. Ahora, un error en la decodificación solo puede ocurrir si se cumple una de estas dos condiciones:

(a) $d(c, Y) > r$

(b) $d(c, Y) \leq r$ y $d(c', Y) \leq r$ para alguna otra palabra código c'.

Es decir si hemos caído fuera de la bola de radio r alrededor del vector c, o si estamos dentro simultáneamente de dos bolas de radio r alrededor de dos palabras del código distintas.
Denotamos los sucesos descritos por (a) y (b) como A y B respectivamente, de modo que nuestro suceso inicial será la unión de ambos $E = A \cup B$, y por lo tanto su probabilidad será

$$P(E) = P(A \cup B) \leq P(A) + P(B).$$

Consideremos ahora el suceso B; este tendrá lugar si no se cometen más de r errores en la transmisión, y además otra de las palabras código distinta de c está a una distancia menor que r del vector recibido Y.
Si estos dos sucesos los denotamos respectivamente por B_1 y B_2 respectivamente, entonces, como $B = B_1 \cap B_2$, tenemos

$$P(B) \leq P(B_2).$$

Ahora consideremos el suceso B_2. Que la distancia entre palabra del código recibida c' y la otra palabra del código $y \in \mathcal{C}$ distinta de la enviada c sea menor que r, $d(y, c') < r$ es equivalente a que la palabra c' este en la bola de $\mathbb{F}_2^n$ centrada en la palabra y y de radio r, es decir en $B_r(y)$. Dado que las palabras código son elegidas al azar, y que el número de vectores de $\mathbb{F}_2^n$ que están en $B_r(y)$ es el mismo que el número de vectores que hay en la bola del mismo radio pero centrada en el vector cero $B_r(0)$. Que se calcula con los coeficientes binomiales $\binom{n}{k}$ que no son más que los vectores de $\mathbb{F}_2^n$ de peso k es decir que tienen k posiciones no nulas de las n disponibles, sumándolos desde que su peso vale 0 hasta que vale r.

$$|B_r(y)| = |B_r(0)| = \sum_{k=0}^{r} \binom{n}{k},$$

Por lo tanto, la probabilidad de que al menos una de las otras $M-1$ palabras código distintas de c esté a una distancia r del vector recibido y satisface la siguiente desigualdad. Tenemos que dividir el número de vectores en la bola $B_r(y)$ entre el número total de vectores en $\mathbb{F}_2^n$ y multiplicarlo por el número de palabras del código distintas a la recibida y tras la decodificación.

$$P(B_2) \leq \frac{M-1}{2^n} \sum_{k=0}^{r} \binom{n}{k}.$$

Por lo tanto, si para algún $\varepsilon > 0$, tomamos $r = \lfloor np + n\varepsilon \rfloor$ como el mayor entero que no supera $np + n\varepsilon$ donde p es la probabilidad de error del canal binario que estamos utilizando, obtenemos de las dos desigualdades anteriores y la desigualdad de la cola, siendo h la función de la entropía del canal, que la probabilidad de B tiene que estara acotada por .

$$P(B) \leq P(B_2) \leq \frac{M-1}{2^n} \sum_{k=0}^{\lfloor np+n\varepsilon \rfloor} \binom{n}{k} \leq \frac{M}{2^n} 2^{n(h(p+\varepsilon))}.$$

Ahora, pasamos al segundo tipo de error dado por el suceso A. Sea T el número de errores que se han producido el la palabra del código c al ser transmitida a través del canal ruidoso. Entonces podemos escribir la probabilidad del suceso A como la probabilidad de que T sea mayor que r.

$$P(A) = P(T > r)$$

La variable aleatoria T que estamos considerando seguirá una distribución binomial con parámetros n la longitud de una palabra y p la probabilidad de que un dígito sea modificado por el ruido del canal. Por lo tanto, por la desigualdad de Chebyshev, podemos acotar la probabilidad del suceso A.

$$P(A) = P(T > np + n\varepsilon) \leq P(|T - np| > n\varepsilon) \leq \frac{\mathrm{var}(T)}{n^2\varepsilon^2},$$

Dado que T es una variable aleatoria binomial, tenemos

$$\mathrm{var}(T) = npq,$$

donde q es el conjugado de p y, por lo tanto, la probabilidad total de error satisface la siguiente desigualdad que se obtiene de la suma de las dos cotas para los dos tipos de errores que se contemplaban

$$P(E) \leq M2^{-n(1-h(p+\varepsilon))} + \frac{pq}{n\varepsilon^2},$$

para n suficientemente grande. Como podemos expresar la capacidad del canal como 1 menos la información perdida a causa del ruido $C(p+\varepsilon) = 1 - h(p+\varepsilon)$, esto se reduce a

$$P(E) \leq \frac{pq}{n\varepsilon^2} + M2^{-nC(p+\varepsilon)}.$$

Dado que $\varepsilon > 0$, esta probabilidad de error puede hacerse arbitrariamente pequeña para n suficientemente grande, siempre que M el número de palabras del código, considerado como una función de n, crezca a un ritmo no mayor que $2^{nC(p)}$.

Hemos probado el teorema de codificación acotando la probabilidad de error media en lugar de la probabilidad de error máxima. Luego para finalizar la demostración es necesario mostrar la existencia de códigos $\mathcal{C}_n$ que tengan un número M_n de palabras tal que $M_n \leq 2^{nR}$, y tal que la probabilidad de error en la decodificación máxima sea $\leq \varepsilon$.

Si tomamos $\varepsilon' = \frac{\varepsilon}{2}$ y $M'_n = 2M_n$, es decir un código con el doble de palabras y notamos que, dado que $M_n \leq 2^{nR}$ y $R < C$, debe existir R' con $R < R' < C$, y N_0 tal que, para todo $n \geq N_0$ se cumple

$$M'_n \leq 2^{nR'},$$

de esta manera podemos construir una sucesión de códigos $\mathcal{C}'_n$ cada uno con M'_n palabras y probabilidad de error media en la decodificación $< \varepsilon'$ para todo $n \geq N_0$.
Si $x_1, \ldots, x_{M_n}$ son las palabras código de $\mathcal{C}'_n$, esto significa que

$$\frac{1}{M'_n} \sum_{i=1}^{M'_n} P(E \mid x_i) \leq \varepsilon'.$$

Por lo tanto, al menos la mitad de estas palabras código x_i deben satisfacer

$$P(E \mid x_i) \leq 2\varepsilon' = \varepsilon.$$

Sea $\mathcal{C}_n$ un subconjunto cualquiera de formado por $M_n = \frac{M'_n}{2}$ palabras código que satisfacen la ecuación anterior; entonces hemos obtenido nuestro código tal que su probabilidad de error máxima es $\leq \varepsilon$.

□

Destacar aquí, que este teorema esta demostrado para el caso en el que el canal sea simétrico, no para el caso de que el canal sea asimétrico, estudiaremos las propiedades de este último canal en el Capítulo 3 de este trabajo, ya que es el que vamos a utilizar en el modelo.

Capítulo 2

Teoría de códigos aplicada a códigos neurales

El objetivo de este capítulo es modelizar matemáticamente la respuesta neural que da una población de neuronas a un conjunto de estímulos mediante teoría de códigos correctores de errores. Para ello vamos construir los códigos del campo receptivo. Estos códigos simulan que neuronas responden a cada estímulo y el ruido al transmitirlos por el canal es lo que explicará las variaciones observadas en las poblaciones de neuronas al responder al mismo estímulo. Un código, como ya sabemos, no es más que la imagen de una aplicación lineal de codificación, y en el caso de los códigos del campo receptivo, el espacio de salida de está aplicación es el espacio de estímulos, el cual será descrito formalmente en este capítulo.

A lo largo de todo este capítulo vamos a considerar, $\mathbb{F}_2 = \{0, 1\}$ el cuerpo finito con dos elementos, $\mathbb{N} = \{0, 1, 2, ...\}$ el conjunto de números naturales, y $n \geq 2$ un número natural.

2.1. Códigos neurales y parámetros asociados

Vamos a definir formalmente los códigos neurales con sus diferentes parámetros.

Definición 2.1. Un *conjunto de neuronas enumeradas* N se define como un subconjunto de los números naturales con esta forma $N = \{1, 2, ..., n\} \subset \mathbb{N}$, donde el número i representa la neurona en la posición i. Este conjunto tiene cardinal $|N| = n$, y por lo tanto, el conjunto partes de N, $\mathcal{P}(N)$ que es el conjunto de todos los posibles subconjuntos de neuronas, tiene cardinal $|\mathcal{P}(N)| = 2^{|N|} = 2^n$.

Definición 2.2. La aplicación de *identificación* $I : \mathbb{F}_2^n \longrightarrow \mathcal{P}(N)$ es una aplicación que tiene como espacio de salida el espacio vectorial de dimensión n con coeficientes en el cuerpo de dos elementos $\mathbb{F}_2$, y que tiene como espacio de llegada el conjunto de todos los subconjuntos posibles de neuronas $\mathcal{P}(N)$. Esta aplicación envía un vector $x \in \mathbb{F}_2^n$ en $I(x) = \text{supp}(x) \stackrel{\text{def}}{=} \{i \in N \mid x_i = 1\} \in \mathcal{P}(N)$.

Es decir, cada subconjunto de neuronas se identifica mediante esta aplicación con un vector de longitud n donde la posición i toma el valor 1 o 0 respectivamente dependiendo de si la neurona $i \in N$ se encuentra o no en el subconjunto original.

Proposición 2.3. *La aplicación de identificación* $I : \mathbb{F}_2^n \longrightarrow \mathcal{P}(N)$ *así como la hemos definido es biyectiva.*

Demostración. Vamos a demostrar primero que esta aplicación es inyectiva. Sean $supp(x), supp(x') \in \mathcal{P}(N)$ imágenes de $x, x' \in \mathbb{F}_2^n$ tales que $supp(x) = supp(x')$, eso implica que si una neurona esta en

uno de los dos conjuntos estará en el otro también, luego la posición $x_i = x'_i$ será 1 si está o 0 sino, luego todas las posiciones de ambos vectores son iguales $x = x'$, la aplicación es inyectiva y como ambos conjuntos tienen el mismo cardinal entonces se tiene que la aplicación I es una biyección. □

En el cerebro cualquier respuesta neural, es consecuencia de la presencia de un estímulo, que puede ser de muchas clases, pero es siempre un estímulo el que desencadena un potencial de acción que activa o inhibe unas neuronas en concreto.

Definición 2.4. Vamos a denotar por $\Lambda \subset \mathbb{R}^d$ al *espacio de estímulos*, al subespacio vectorial que representa un conjunto continuo de estímulos.

Bastaría con pedir que el espacio de estímulos Λ fuera simplemente un espacio topológico para poder definir relaciones de proximidad y continuidad entre estímulos. Sin embargo, podemos considerar que Λ es un subconjunto de $\mathbb{R}^d$ ya que es particularmente útil para poder estudiar también su geometría.

Esta modelización del espacio de estímulos es coherente con el funcionamiento del cerebro, el cuál organiza los estímulos de forma que aquellos que son parecidos entre sí tienen representaciones neurales similares. Por ejemplo, una barra inclinada a 45 grados se percibe como más similar a otra inclinada a 50 grados que a una que se encuentra en una posición perfectamente vertical. En lugar de ver los estímulos como datos aislados, nuestro cerebro organiza la información sensorial de manera que percibamos relaciones y patrones lo que nos facilita la interpretación adecuada de nuestro entorno.

Sin embargo, es complicado definir lo que es un estímulo, o medirlo de alguna manera, pero lo que sí que podemos observar es cómo un organismo reacciona a él, y más en concreto cómo lo hace su cerebro. Dada una población de neuronas, el subconjunto de ellas que se activan en la presencia de un estímulo es algo tangible y que podemos observar. Es decir, vamos a identificar de ahora en adelante un estímulo con el subconjunto de neuronas de nuestra población que idealmente responderían a él.

Como sabemos un código se define como la imagen de una aplicación inyectiva que traduce el alfabeto fuente en el alfabeto código. En el caso de códigos neurales que vamos a definir a continuación; el alfabeto fuente será el conjunto de estímulos Λ mientras que el alfabeto de la información codificada será $\mathbb{F}_2^n$ que se puede interpretar según la aplicación de identificación de la Definición 2.2, como un conjunto de neuronas.

Definición 2.5. Un *código neural* $\mathcal{C}$ es la imagen de una aplicación de codificación $c : \Lambda \longrightarrow \mathbb{F}_2^n$, es decir, $\mathcal{C} = Im(c)$ donde cada estímulo se envía al vector que representa el conjunto de neuronas que se activan o que responden a él.

Podemos hablar de diversos parámetros asociados a un código neural, en particular, cuatro de ellos: el tamaño, la longitud, el peso de Hamming y la distancia de Hamming; ya los definimos para códigos en general en la introducción del Capítulo 1.1, pero vamos a concretarlo en esta sección para el caso específico de los códigos neurales que acabamos de definir.

Definición 2.6. El *tamaño* de un código neural $\mathcal{C}$ es simplemente el número total de palabras de código $|\mathcal{C}|$, que corresponderá a la cantidad de estímulos que estemos considerando en el espacio Λ, ya que cada uno de ellos conlleva una repuesta neural, representada como un vector en $\mathbb{F}_2^n$.

Definición 2.7. La *longitud* de un código neural $\mathcal{C}$ es la longitud de las palabras del código, (podemos definirla de esta manera, ya todas tienen la misma longitud por como hemos construido el código) y por tanto va a coincidir con el número de neuronas n, el cardinal del conjunto N.
Un código neural con n neuronas es un *código de bloque* de longitud n.

Definición 2.8. El *peso de Hamming* $\omega_H(x)$ de una palabra del código $x \in \mathcal{C}$ se define como el cardinal del conjunto imagen de $c \in \mathcal{C}$ por la aplicación de identificación $I(x) = supp(x)$, o alternativamente, el número de posiciones no nulas cuando se ve como un elemento de $\mathbb{F}_2^n$. Esto corresponde con el número de neuronas que se activan o que responden al estímulo.

$$\omega_H(x) = |\{\ i \in N \quad | \quad x_i = 1\ \}|$$

Definición 2.9. La *distancia* entre dos palabras del código neural $x, x' \in \mathcal{C}$ será el número de posiciones en las que se diferencian una de otra, es decir, definimos la distancia entre x y x' como $d(x, x') = |\{i \mid x_i \neq y_i, 1 \leq i \leq n\}|$. El número de neuronas que reaccionan de forma distinta para cada uno de los dos estímulos.

Definición 2.10. La *dispersión* s de un código neural es la media de los pesos relativos de todas las palabras del código, donde los pesos relativos se calculan como el peso de la palabra dividido entre la longitud de esta, que es siempre n.

$$s = \frac{1}{|\mathcal{C}|} \sum_{x \in \mathcal{C}} \frac{\omega_H(x)}{n}$$

Observando esta definición podemos ver que la dispersión es siempre positiva y toma valores en el rango $0 \leq s \leq 1$. La dispersión representa la idea de cuántas neuronas responden a cada estímulo. Si s es pequeño, próximo a 0, el código será disperso y en él la activación neural será más selectiva, solo un pequeño número de neuronas responde significativamente a un estímulo concreto. Esto ocurre en el cerebro, cuando no existe demasiado solapamiento entre los campos receptivos de las neuronas que estamos considerando; los campos receptivos son las regiones espaciales alrededor de cada neurona donde la presencia de un estímulo provoca que la neurona se active, pero más adelante veremos más detalladamente este concepto.

Definición 2.11. La *redundancia* ρ de un código neural $\mathcal{C}$ se define como

$$\rho = 1 - \frac{\log_2(|\mathcal{C}|)}{n}.$$

Esta cualidad del código está estrechamente relacionada con su tamaño, cuantifica cuantas neuronas a mayores de las necesarias se están utilizando para codificar un conjunto dado de estímulos, o la información que nos transmiten cada uno de ellos.
Observemos también, que $|\mathcal{C}| \leq 2^n$, suponiendo que el código esta formado por al menos una palabra $0 \leq \log_2(|\mathcal{C}|) \leq n$, en consecuencia la redundancia ρ de un código varía también entre 0 y 1, y cualquier par de códigos con el mismo tamaño y la misma longitud (misma cantidad de neuronas) tendrán automáticamente la misma redundancia, pues esta solo depende de esos dos parámetros.

Ejemplo 2.12. Redundancia y dispersión de códigos sencillos
Podemos considerar dos casos extremos y simples:

1. El primero es el código de repetición $\mathcal{C} = \{\emptyset, N\}$ que contiene solo dos palabras la palabra con todo ceros y la palabra con todo unos. En el contexto en el que estamos serían dos palabras correspondientes a un estimulo que no consigue desencadenar ninguna respuesta neural, y otro que no deja indiferente a ninguna neurona de la población. La redundancia de este código neural es $\rho = \frac{n-1}{n}$, lo que sugiere que todas las neuronas son redundantes excepto una, es decir con tan solo una se podría transmitir la misma cantidad de información.
   En este caso la dispersión de este código es sencillamente,

$$s = \frac{1}{2}\left(\frac{0}{n} + \frac{n}{n}\right) = \frac{1}{2}.$$

2. Por otro lado, tenemos el caso opuesto, el código $\mathcal{C} = \mathcal{P}(N)$, que incluye todos los subconjuntos posibles de N, la redundancia de este código es nula $\rho = 0$, lo que significa que todas las neuronas son esenciales para la codificación, si se suprimiera alguna no sería posible transmitir la misma cantidad de información.
   Para este ejemplo, si consideramos por ejemplo $n = 3$ para simplificar los cálculos, la dispersión será:
$$s = \frac{1}{2^3}\left(\frac{0}{3} + 3 \cdot \frac{1}{3} + 3 \cdot \frac{2}{3} + \frac{3}{3}\right) = \frac{1}{2^3}\left(0 + 1 + 2 + 1\right) = 1/2.$$

En ambos ejemplos, la dispersión es exactamente igual a un medio, pues las palabras con peso relativamente bajo y aquellas con peso relativamente alto se pueden encontrar en la misma proporción en el código.

Ejemplo 2.13. Sea $\mathcal{C}$ el código

$$\mathcal{C} = \{(0,0,0,1), (0,0,1,0), (0,1,0,0), (1,0,0,0)\} \subset \mathbb{F}_4^2.$$

Podemos calcular su dispersión muy fácilmente:

$$s = \frac{1}{4}\left(\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}\right) = 1/4.$$

En este caso, como todas las palabras del código tienen peso igual a 1, lo cual es muy bajo, la dispersión será menor que $1/2$ y por lo tanto, podríamos decir que es un código disperso.
En cambio, sea $\mathcal{C}$ el código

$$\mathcal{C} = \{(0,1,1,1), (1,1,1,0), (1,1,0,1), (1,1,1,1)\} \subset \mathbb{F}_4^2.$$

Su dispersión será:
$$s = \frac{1}{4}\left(\frac{3}{4} + \frac{3}{4} + \frac{3}{4} + \frac{4}{4}\right) = 13/16.$$

Este código es muy poco disperso, pues el peso de las palabras que lo componen es alto, ya que el peso máximo de las palabras de $\mathbb{F}_4^2$ es a lo sumo 4 y por lo tanto, al calcular su dispersión comprobamos que está es mayor que $1/2$.

2.2. Códigos del Campo Receptivo

En neurociencia, el campo receptivo de una neurona no es más que es la región del espacio dentro del cerebro en la cual la presencia de un estímulo puede provocar la respuesta de dicha neurona, es decir es la región del espacio donde esta es sensible a la presencia de estímulos. Mientras que la tasa de activación o de disparo de una neurona es la cantidad de potenciales de acción que genera por unidad de tiempo, es decir la cantidad de señales que la neurona emite para comunicarse con otras células nerviosas, es decir, mide la frecuencia con la que se comunica una neurona. La presencia de un estímulo en el campo receptivo de una neurona, provoca variaciones en la tasa de disparo aumentándola, o en algunos casos inhibiéndola, es decir que podemos tomar la tasa de disparo como un buen indicador del nivel de activación neural.

Para estudiar como afectan los diferentes estímulos a la población o conjunto de neuronas con el que estamos trabajando vamos a definir una aplicación para cada neurona, que contenga la información de como se comporta respecto a cada uno de los estímulos del espacio, y esta aplicación es lo que matemáticamente va a recibir el nombre de campo receptivo.

Definición 2.14. El *campo receptivo* asociado a la neurona $i \in N$ es una aplicación $f_i : \Lambda \to \mathbb{R}_{\geq 0}$ que va desde el espacio de estímulos Λ en el conjunto de los números reales $\mathbb{R}$. Sea $z \in \Lambda$ un estímulo, definimos $f_i(z)$ como la tasa de activación de la neurona i, como respuesta a dicho estímulo.

Sería interesante conocer el conjunto de todos los estímulos que provocan una reacción en la neurona i, podríamos pensar en obtenerlo considerando aquellos cuya imagen por el campo receptivo es no nula. Pero en la práctica, debido a diferentes limitaciones en los instrumentos de medida y capacidad de medición, vamos a necesitar definir un umbral, de tal manera que si la tasa de activación de una neurona para un estímulo dado supera el umbral, consideraremos que esta activa y sino lo supera, consideraremos que esta inactiva.

Definición 2.15. Dado un umbral $\theta > 0$, $\theta \in \mathbb{R}$, definimos el *soporte del campo receptivo bajo el umbral θ asociado a la neurona i* como el subconjunto del espacio de estímulos $\mathcal{S}_\theta \subset \Lambda$ donde la aplicación f_i toma valores superiores al umbral.

$$\mathcal{S}_\theta = \{z \in \Lambda \mid f_i(z) > \theta\}$$

Vamos a cometer un abuso de notación, ya que en lo que resta de este trabajo se designara como campo receptivo tanto a la aplicación en sí como a su soporte dando por hecho que se ha fijado un umbral, siempre y cuando no de lugar a confusión.

La interpretación biológica es que cuando un estímulo se encuentra en la intersección de varios campos receptivos, las neuronas correspondientes a dichos campos receptivos se activarán, mientras que aquellas para las que el estímulo no se encuentra en su campo receptivo permanecerán inactivas, como ya adelantabamos.

Lo que sabemos hasta ahora es lo siguiente, si tenemos $\sigma \subset N$ un subconjunto de neuronas de nuestra población, tal que es el conjunto de todas las que se activan ante un determinado estímulo $z \in \Lambda$. Mediante la aplicación de identificación lo podemos considerar también un elemento del espacio vectorial $x \in \mathbb{F}_2^n$.

$$x = (x_1, \ldots, x_n), \quad supp(x) = \sigma, \text{ donde } x_i = \begin{cases} 1 & \text{si } i \in \sigma, \\ 0 & \text{si } i \notin \sigma. \end{cases}$$

El objetivo es definir un código neural que tenga en cuenta los campos receptivos de las neuronas.

Definición 2.16. Dado un umbral $\theta > 0$, $\theta \in \mathbb{R}$, definimos la *aplicación de respuesta binaria* a la aplicación que transforma una respuesta neural a un estímulo (es decir un conjunto de neuronas que reaccionan a él) en una palabra del código un elemento de $\mathbb{F}_2^n$ de la siguiente manera:

$$\phi : \Lambda \to \mathbb{F}_2^n, \quad \text{donde } \begin{cases} \phi_i(z) = 1 & \text{si } f_i(z) \geq \theta \\ \phi_i(z) = 0 & \text{si } f_i(z) < \theta \end{cases} \text{ con } x \in \Lambda.$$

Es decir, la posición i de la palabra del código será 1 si la tasa de activación de neurona i es superior o igual a θ, en caso contrario, la posición i será 0 si la tasa de activación de la neurona i, no llega al umbral delimitado.

La aplicación de respuesta binaria ϕ no es inyectiva, lo que significa que diferentes estímulos pueden producir la misma palabra del código. En particular, esto ocurre en las regiones donde se producen solapamientos del soporte de los campos receptivos, es decir, el solapamiento de las regiones del espacio donde las neuronas que estamos considerando son sensibles a los estímulos.

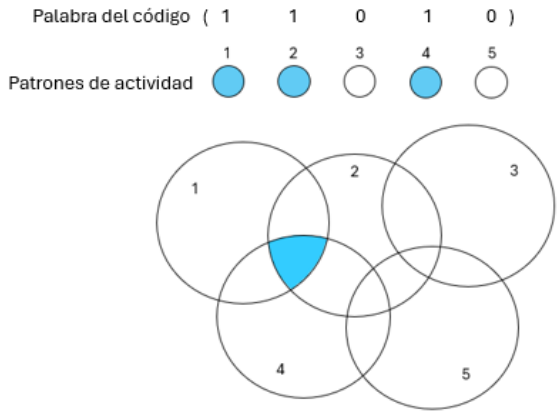

Figura 2.1: Esto es una representación de un código de campo receptivo bidimensional de longitud 5 es decir sobre un conjunto de 5 neuronas, cada una tiene asociada un campo receptivo que divide el espacio bidimensional de estímulos en regiones superpuestas o no, como se ilustra en la imagen. Todos los estímulos que estén dentro de la región sombreada activarán el mismo conjunto de neuronas y, por lo tanto, estarán asociados a la misma palabra del código.

Definición 2.17. Para un conjunto de neuronas dado N, un espacio de estímulos Λ junto con sus respectivos campos receptivos, definimos el *código del campo receptivo* $\mathcal{C}$ o simplemente *RF* abreviatura de su nombre en inglés *Receptive Field Code*, como la imagen de $\phi : \Lambda \to \mathbb{F}_2^n$ la aplicación de respuesta binaria que acabamos de definir.

El código del campo receptivo está formado por palabras que representan subconjuntos de neuronas de una población conocida, los subconjuntos formados por aquellas que se activan frente a cada uno los diferentes estímulos a los que se las expone, por lo tanto es claro que habrá tantas palabras en el código como estímulos estemos considerando en el espacio de estímulos. Considerando que activarse significa que su tasa de activación frente al estímulo supera el umbral establecido para definir el soporte de los campos receptivos y la aplicación de respuesta binaria.

Definición 2.18. La *dimensión* de un código del campo receptivo o RF es la dimensión del espacio de estímulos subyacente.

2.3. Códigos de Comparación

Para evaluar el rendimiento y la eficiencia de los códigos del campo receptivo (RF), una estrategia sería compararlos con códigos generados aleatoriamente que compartan las mismas características clave: tamaño, longitud y dispersión. Dado que la redundancia solo depende de estos parámetros, estos códigos aleatorios tendrían la misma redundancia que los códigos del campo receptivo correspondientes, lo que los convierte en una herramienta útil para su análisis. Vamos a describir en esta sección dos en concreto: códigos permutados y códigos de peso constante.

La elección de estos códigos se justifica por tres razones principales. Primero, según el teorema de codificación de canal de Shannon (1948), Teorema 1.35, se espera que los códigos aleatorios presenten un rendimiento cercano al óptimo, aunque esta demostración está limitada a canales binarios simétricos y no se extiende directamente al canal asimétrico considerado en este modelo.

Segundo, permiten ajustar sus parámetros para que coincidan con los códigos del campo receptivo, facilitando así la comparación. Y tercero, desde el punto de vista biológico, los códigos aleatorios son una opción plausible para el cerebro, ya que podrían implementarse mediante redes neuronales con conexiones aleatorias.

En este trabajo no se llevará a cabo un análisis detallado de la eficiencia de estos códigos. Su descripción aquí tiene el propósito de destacar su relevancia como una herramienta potencialmente útil para estudios futuros.

Definición 2.19. Un *código permutado* $\mathcal{C}'$ se define como

$$\mathcal{C}' = \{(c_{\pi_c(1)}, c_{\pi_c(2)}, ..., c_{\pi_c(n)})/c \in \mathcal{C}, \pi_c \in \mathcal{P}\}$$

donde

- $\mathcal{C}$ es un código del campo receptivo
- $\mathcal{P} = \{\pi_c/c \in \mathcal{C}\}$ es un conjunto de permutaciones que cumple que para cada par de ellas distintas $c, c' \in \mathcal{C}$, entonces

$$(c_{\pi_c(1)}, c_{\pi_c(2)}, ..., c_{\pi_c(n)}) \neq (c_{\pi'_c(1)}, c_{\pi'_c(2)}, ..., c_{\pi'_c(n)})$$

Es decir, un código permutado no es más que un código del campo receptivo donde hemos reordenados los elementos de todas las palabras de acuerdo con unas permutaciones fijas, exactamente se elije una permutación diferente para cada palabra.

El código permutado $\mathcal{C}'$ tiene el mismo número de palabras, es decir el mismo tamaño, las palabras tienen el mismo número de letras luego la misma longitud y distribución de peso, es decir para cada palabra su correspondiente palabra permutada tendrá el mismo peso y, por lo tanto, este código también tendrá la misma dispersión y redundancia que el código original $\mathcal{C}$.

Definición 2.20. Un *código de peso constante m* $\mathcal{C}_m$ es un conjunto de vectores con coeficientes en $\mathbb{F}_2$ que cumplen que todos sus elementos tienen el mismo peso igual a m, es decir cada palabra del código tiene m posiciones igual a 1.

Para generar un código de peso constante con parámetros similares a un código del campo receptivo $\mathcal{C}$. Haremos lo siguiente:

1. Calcularemos la media del peso de las palabras del código $\mathcal{C}$ y redondeamos ese valor para obtener un entero $w \in \mathbb{Z}$.
2. Elegimos aleatoriamente un conjunto de tamaño w de entre los índices de los elementos, es decir, del conjunto $\{1, 2, ..., n\}$, donde n es la longitud del código $\mathcal{C}$. De forma natural los elementos en este representan las posiciones de los unos en la palabra del código de peso constante, mientras que las demás posiciones se completan con ceros.
3. Repetimos este proceso hasta generar $|\mathcal{C}|$ palabras distintas todas con peso w.

El código resultante $\mathcal{C}_w$ tiene la misma longitud y el mismo tamaño, por como lo hemos construido y la misma redundancia que $\mathcal{C}$ pues esta solo depende de la longitud y tamaño del código, y por como lo hemos construido tiene aproximadamente el mismo peso y por tanto también aproximadamente la misma dispersión.

2.4. Discretización del espacio de estímulos

Es importante tener en cuenta, que para estudiar desde el marco de la teoría de códigos, los nuevos códigos del campo receptivo que acabamos definir, es necesario discretizar el espacio de estímulos. De esta manera podemos definir una aplicación de codificación inyectiva que tenga como espacio de partida el espacio de estímulos y como llegada el conjunto de palabras del código, lo que en la práctica significa, que utilizando la aplicación inversa a la codificación podemos obtener de vuelta el estímulo equivalente. Se pretende a través de esta aplicación identificar cada estímulo con un patrón cerebral, o lo que es lo mismo, con el conjunto de neuronas se activan o reaccionan ante dicho estímulo. Podemos utilizar con este fin, la respuesta ideal de la población de neuronas, a cada uno de los estímulos para definir esta aplicación, que no es más que una aplicación de respuesta binaria descrita en la Definición 2.16 .

Según hemos definido los códigos del campo receptivo, sabemos que el espacio de estímulos $\Lambda \subset \mathbb{R}^n$ es un subconjunto continuo de un espacio euclídeo, y que además, a través de la inversa de aplicación de respuesta binaria, un conjunto de n campos receptivos, asociados cada uno a una neurona, divide el espacio de estímulos en n regiones diferentes, que pueden solaparse. La región del espacio de estímulos para una neurona es el conjunto de estímulos, que la activan, matemáticamente que la imagen del estímulo por aplicación de campo receptivo supera el umbral establecido.
Recordemos que el código del campo receptivo se definía como la imagen de $\phi : \Lambda \to \mathbb{F}_2^n$ la aplicación de respuesta binaria, entonces para cada palabra del código $x \in \mathcal{C}$, existe una región de las anteriores, denotada como $\phi^{-1}(x)$, donde todos sus puntos en su interior se codifican en la misma palabra del código x. Esto implica que el código $\mathcal{C}$ no puede distinguir entre estímulos que están dentro de la misma región, es decir que dispone de resolución limitada. Para poder definir entonces una aplicación inyectiva vamos a tener que discretizar el espacio de estímulos asignando un representante para cada conjunto de estímulos que se mapeen a la misma palabra del código, para todos aquellos estímulos que activen las mismas neuronas.

Definición 2.21. El *centro de masa* de la región del espacio $\phi^{-1}(x)$, que son los puntos que se codifican en la palabra x, suponiendo que la densidad de masa en todos los puntos es la misma e igual a 1, se define como

$$\hat{z}(x) = \frac{\int_{\phi^{-1}(x)} z\, dz}{\int_{\phi^{-1}(x)} dz},$$

donde

- z es el vector de posición de los puntos en la región $\phi^{-1}(x)$
- La primera integral del numerador $\int_{\phi^{-1}(x)} z\, dz$ es el momento ponderado de la distribución de masas.
- La integral del denominador $\int_{\phi^{-1}(x)} dz$ calcula el total de la masa de la región $\phi^{-1}(x)$.

Definición 2.22. El *espacio de estímulos discretizado* se define como

$$\hat{\Lambda} = \{\hat{z}(x) \mid x \in \mathcal{C}\} \subset \Lambda,$$

Es el conjunto de todos los centros de masa de cada una de las regiones del espacio que se codifican en la misma palabra. De esta manera, tiene que existir una aplicación biyectiva entre los estímulos en espacio discretizado $\hat{\Lambda}$ y las palabras del código en $\mathcal{C}$, resultando en que ambos conjuntos tengan el mismo cadinal $|\hat{\Lambda}| = |\mathcal{C}|$.

Definición 2.23. Podemos definir ahora la *aplicación de codificación* como la restricción de la aplicación de respuesta binaria al espacio de estímulos discretizado.

$$\varphi = \phi|_{\hat{\Lambda}} : \hat{\Lambda} \to \mathcal{C}.$$

La aplicación de codificación φ es inyectiva, lo que significa que tiene una aplicación inversa bien definida. Por lo tanto, la estimación de respuesta ideal devuelta por el decodificador puede actuar como una aproximación del estímulo real.

Para los códigos de comparación, se utilizaría el mismo espacio de estímulos discretizado $\hat{\Lambda}$ que en el código del campo receptivo correspondiente. A cada estímulo se le asignaría aleatoriamente una palabra del código usando una aplicación de codificación aleatoria, $\phi : \hat{\Lambda} \to \mathcal{C}$. Esta aplicación se podría crear por ejemplo, ordenando tanto los estímulos en $\hat{\Lambda}$ como los códigos en $\mathcal{C}$, y luego aplicando una permutación aleatoria que asigne cada estímulo a cada palabra.

2.5. Modelización de la respuesta neural a los estímulos

Cuando el cerebro humano percibe un estímulo la reacción que se produce a nivel neural no es siempre la misma, sino que se pueden producir pequeñas modificaciones cada vez, un mismo estímulo puede provocar diversas reacciones neurales, o lo que es lo mismo el conjunto de neuronas que se activan en su presencia puede variar ligeramente. El objetivo de esta sección es modelar de forma matemática este fenómeno, con el fin de que aunque la respuesta neural no coincida exactamente con la respuesta ideal o respuesta modelo se pueda identificar cual es el estímulo que la ha desencadenado.

Recapitulando, si consideramos un espacio de estímulos discretizado $\hat{\Lambda}$ y la aplicación de codificación $\varphi : \hat{\Lambda} \longrightarrow \mathcal{C} \subseteq \mathbb{F}_2^n$ que es la aplicación inyectiva que manda cada estímulo en la palabra del código correspondiente, es decir, que cada estímulo provoca una reacción neural diferente en nuestra población. Podemos modelizar la reacción neural a un estímulo de la siguiente manera. Si tenemos un estímulo $z \in \hat{\Lambda}$ utilizando la aplicación de codificación obtenemos una palabra del código $\varphi(z) = x$ con $x \in \mathcal{C} \subseteq \mathbb{F}_2^n$, esta pasará a través de un canal con ruido donde cada dígito puede ser modificado con una cierta probabilidad. En este modelo es el ruido lo que explica, porque no siempre la respuesta neural a el mismo estímulo es siempre igual, luego si el ruido provoca una alteración en un dígito es que ha ocurrido uno de estos dos fenómenos:

- Un cambio de un dígito que pasa de ser un 1 a un 0, corresponde con una neurona que idealmente se tenía que activar en la respuesta ideal, pero que en la realidad no se activa.
- Un cambio de un dígito que pasa de ser un 0 a un 1, corresponde con una neurona que no se tenía que haber activado en la respuesta ideal, pero que en la realidad sí responde al estímulo.

Definición 2.24. La *palabra recibida* y es un vector perteneciente a $\mathbb{F}_2^n$ que se obtiene como resultado cuando una palabra del código recorre el canal ruidoso. La palabra recibida no tiene porque pertenecer al conjunto de las palabras del código, ya que ha visto sometida a esa probabilidad de sufrir modificaciones en sus dígitos.

Una vez que la palabra del código atraviesa el canal, y se obtiene la palabra recibida, esta tendrá que someterse a un decodificador que en la siguiente sección definiremos de forma rigurosa, pero de forma resumida su función es estimar cual era la palabra original del código enviada a través del canal, buscando la palabra del código que más se asemeja a la palabra recibida. Finalmente, si se desea encontrar también una aproximación del estímulo original basta aplicar la inversa de la aplicación de codificación, la cual existe por ser está biyectiva, a la palabra estimada del código que obtiene el decodificador. Pero como en el cerebro solo se pueden medir patrones de actividad es innecesario hacer el cambio y se puede considerar la respuesta neural como una representación del estímulo en sí mismo. En la Figura 2.2, podemos ver con detalle como funciona este modelo.

Según este modelo los patrones de actividad del cerebro que se obtienen al realizar mediciones corresponden a las palabras recibidas en lugar de a las palabras del código, aquí podemos ver un

poco el objetivo del decodificador será obtener de vuelta de esos patrones cerebrales la respuesta ideal, y así averiguar el estímulo al que se han visto expuestos.

Hay que tener en cuenta que el decodificador solo podrá funcionar si la palabra recibida no es otra palabra del código, ya que esta va a ser la forma que tiene de detectar si ha ocurrido un error. Esto quiere decir que si la palabra enviada se deforma tanto, que se convierte en otra palabra del código, no podremos hacer nada. Es decir, es el decodificador es eficiente cuando las palabras del código están lejos entre si, pues de esta forma aunque se modifiquen bastantes dígitos sigamos estando suficientemente cerca de la palabra original con respecto a las demás, para que los errores puedan corregirse enviando la palabra recibida a la la palabra del código más cercana. Esto es exactamente igual que lo que ocurre en teoría de códigos clásicos.

La función de un código neural es representar la información de manera que se pueda decodificar bien en un porcentaje muy alto de las ocasiones. La redundancia en la representación de la información de un código va a jugar un papel clave, ya que es lo que nos va permitir corregir errores.

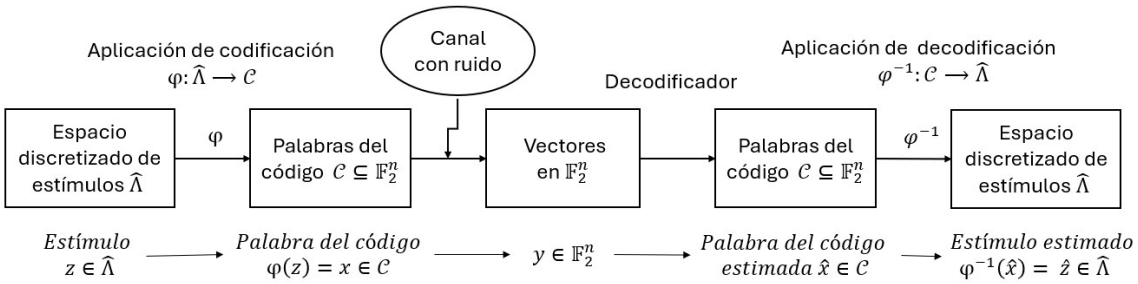

Figura 2.2: Modelo de codificación y decodificación de un estímulo a través de un canal ruidoso. Primero, la aplicación de codificación envía un estímulo a una palabra del código, que representa como respondería idealmente a ese estímulo, la población de neuronas estudiada. En este modelo se considera que la respuesta real de la población de neuronas se obtiene al atravesar la palabra inicial un canal ruidoso, y convertirse en otro vector y la palabra recibida, que ya no tiene porque ser una palabra del código, y esta es el patrón cerebral que observamos en las mediciones. Finalmente, se utilizará un decodificador para encontrar una palabra, que sí que pertenezca al código y que sea una estimación de lo que era la palabra y antes de pasar por el canal ruidoso, y por último se puede utilizar la inversa de la aplicación de codificación para recuperar el estímulo original al que se expuso a la pobración de neuronas.

Nota 2.25. Es conveniente hacer notar que se habla usualmente del proceso de decodificación no como la inversa de la aplicación de codificación, la cual existe por ser la aplicación de codificación inyectiva, este proceso es trivial ya que la aplicación de codificación es conocida. Sino como la estimación de la palabra del código original que se envío a través del canal a partir de la obtenida tras la trasmisión. Un decodificador usual para corregir los errores producidos en la transmisión es el que devuelve la palabra del código más cercana a la recibida, en el sentido que minimiza la distancia de Hamming entre ellas [10].

Para modelar este fenómeno también es fundamental describir el canal ruidoso involucrado. Se utiliza un canal binario asimétrico, la diferencia con el canal binario simétrico, que se utiliza en

la teoría de códigos clásica, radica en que existen probabilidades distintas para cada uno de los posibles cambios que se pueden producir.
Esto quiere decir que existe una probabilidad p de que un dígito 0 cambie al 1, es decir una probabilidad p de falso positivo y una probabilidad q de que el dígito 1 cambie al 0, es decir de que se produzca un falso negativo, ambas probabilidades se consideran siempre menores a $1/2$, pues que se produzcan errores ha de ser menos probable que, que la transmisión sea correcta. Dos casos especiales del canal binario asimétrico se producen cuando $p = q$ obteniéndose el canal binario simétrico y cuando $p = 0$ se obtiene el Z-canal, que es el canal que transmite todos los dígitos que son iguales a 0 de forma correcta y transforma los dígitos que son iguales a 1 en 0 con una probabilidad $q < 1/2$. Se puede observar este modelo de canal binario asimétrico en la Figura 2.3. Además, se considera que el ruido del canal actúa de forma independiente sobre cada uno de los símbolos que conformen la palabra del código que se esta transmitiendo, es decir sobre los los componentes individuales de un vector de $\mathbb{F}_2^n$.
En el cerebro, se ha observado que es más probable que una neurona que en principio se tenía que activar en la respuesta ideal no lo haga, a que una neurona que no tenía que activarse en la respuesta ideal se active. Para modelizar correctamente entonces este fenómeno, vamos suponer que $p \leq q$, esto nos dice que es más probable que un 1 se convierta en un 0 que un 0 se convierta en un 1, de acuerdo con la realidad observada experimentalmente, haciendo que el modelo se ajuste a ella. Estudiaremos más en profundidad este nuevo canal más adelante en el Capítulo 3 de este trabajo.

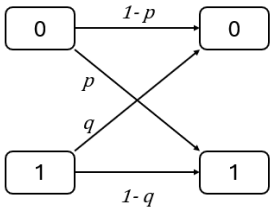

Figura 2.3: Representación gráfica del canal binario asimétrico

2.6. Decodificadores ML y MAP

La función de un decodificador es tomar la respuesta real de una población de neuronas y obtener de vuelta el estímulo al que han sido expuestas. Un decodificador toma la palabra recibida a través del canal $y \in \{0,1\}^n = \mathbb{F}_2^n$ y devuelve una palabra del código $\hat{x} \in \mathcal{C}$ que será una estimación de la palabra enviada $x \in \mathcal{C}$. En el siguiente capítulo, al describir con mayor profundidad en canal asimétrico veremos que existe otro decodificador equivalente al que introducimos ahora, el de máxima verosimilitud, pero cuya expresión de calculo es mucho más sencilla, solo que se necesita para ello un mayor conocimiento del canal. Introducimos aquí brevemente estas noción de decodificador, ya que es una pieza fundamental del modelo pero aún es necesario definir también de forma rigurosa la función probabilidad condicionada que estamos utilizando, conectaremos con esto en el Capítulo 3.

Para cada combinación de código y canal, el decodificador que es óptimo en el sentido de que minimiza errores, es el que devuelve la palabra del código $\hat{x}$ que tiene la mayor probabilidad de haber sido enviada, sabiendo que se ha recibido y.

Definición 2.26. El *decodificador de máximo a posteriori* o *MAP* por sus siglas en inglés se define como la palabra del código $x \in \mathcal{C}$ que maximiza la probabilidad condicionada de recibido y, haber

sido enviado x.

$$\mathcal{D}_{\mathcal{C}}^{\mathrm{MAP}}(y) = \arg\max_{x \in \mathcal{C}} P(\text{enviado} = x \mid \text{recibido} = y).$$

También es conocido en la literatura de neurociencia como decodificador de inferencia bayesiana o decodificador del observador ideal. Aunque este decodificador siempre es óptimo, es difícil de implementar en el contexto neural, ya que requiere conocer las todas las probabilidades de recibido y, cual es la probabilidad de haber sido enviado x para cada $x \in \mathcal{C}$, que son de cálculo complejo, por que es equivalente a conocer la distribución de probabilidad de los estímulos. Por lo tanto, es utópico pensar que se pueden utilizar en la práctica.

En su lugar, podemos pensar en este otro decodificador, para el cual veremos que las probabilidades involucradas en su definición son más sencillas de calcular.

Definición 2.27. El decodificador de máxima verosimilitud, también se suele ver escrito abreviado como (ML), con las siglas que provienen de su nombre en inglés, *maximum likelihood decoder*, y se define como la palabra del código $x \in \mathcal{C}$ que maximiza la probabilidad condicionada habiéndose enviado x de recibir el vector y.

$$\mathcal{D}_{\mathcal{C}}^{\mathrm{ML}}(y) = \arg\max_{x \in C} P(\text{recibido} = y \mid \text{enviado} = x).$$

Busca la palabra del código x que maximiza la probabilidad de haber recibido el mensaje y cuando se ha enviado dicha palabra del código x. Este es más sencillo de implementar, porque equivale a optimizar una función simple que se puede calcular directamente a partir de los parámetros del canal, o lo que es lo mismo de las probabilidades p y q de que ocurran errores en la transmisión, como veremos en detalle en el capítulo siguiente. Luego en la práctica será este decodificador el que se implemente.

Capítulo 3

Canal binario asimétrico

Hemos estado estudiando siguiendo el artículo [4] la teoría de códigos y su aplicación en la neurociencia para modelar las diferentes respuestas cerebrales a un mismo estímulo, mediante la discretización del espacio de estímulos, la división del espacio en regiones provocada por los campos receptivos neurales y modelando la representación neural de los estímulos como códigos binarios $\mathcal{C} \subseteq \mathbb{F}_n^2 = \{0,1\}^n$. En este nuevo enfoque, cada una de las n coordenadas de un vector representa una neurona, representamos que esta activa cuando la coordenada correspondiente a su posición toma el valor 1, y inactiva en caso de que tome el valor 0. En el contexto de los códigos de campos receptivos discretizados, se asume que el fallo en la activación de una neurona que debería responder a un estímulo es más probable que la activación por error de una neurona que no debería ser desencadenada por el estímulo, ya que es lo que se ha observado de forma experimental en neurociencia. Este comportamiento se modela mediante el canal binario asimétrico denominado también *BAC*, por sus siglas en inglés, en el cual un 0 se convierte en 1 con probabilidad p, y un 1 se convierte en 0 con otra probabilidad distinta mayor que la anterior $q > p$, por lo expuesto anteriormente, donde también se asume que ambas probabilidades son menores que $1/2$. Esto último es análogo en el caso simétrico, ya que siempre se asume que en el canal que se esta utilizando la probabilidad de que la transmisión de un dígito sea correcta es mayor, que la probabilidad de que el ruido lo modifique. Para la descripción de este canal en este trabajo se ha utilizado fundamentalmente el artículo [2].

Además, para las definiciones fundamentales y el decodificador que se incluye en este artículo, se presenta una propuesta de como se podrían implementar utilizando el sistema de software matemático libre *SageMath* [13], cuyo lenguaje base es Python. *SageMath* es una alternativa de código abierto a otros programas de pago más conocidos como *Magma*, *Maple*, *Mathematica* o *Matlab*. Algunos de fragmentos se presentan de forma paralela al texto, y a mayores el código completo más detallado se puede consultar en el Apéndice A junto con ejemplos. Todo el código que se presenta en el artículo son implementaciones propias de las funciones, que no estaban disponibles, y se han realizado con la única finalidad de ilustrar el contenido del trabajo.

El objetivo de este capítulo es describir las propiedades matemáticas del canal binario asimétrico y la estructura de los códigos binarios cuando el canal a través del cual son enviados es el asimétrico. Vamos a centrarnos en la teoría de códigos binarios para el canal asimétrico, enfocándonos en sus propiedades estructurales y parámetros. Cuando se estudia el canal binario simétrico como en [10] se utiliza la distancia de Hamming como métrica en el conjunto de palabras del código, de manera similar en este caso asimétrico, vamos a introducir dos nociones de discrepancia entre vectores binarios $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$, que vamos a denotar como $\delta(\mathbf{x}, \mathbf{y})$ y $\hat{\delta}(\mathbf{x}, \mathbf{y})$, respectivamente. Aunque estas funciones no son métricas en general, son relativamente simples y están relacionadas de manera

natural con las probabilidades que definen el canal binario asimétrico. Además, establecemos propiedades generales de δ y $\hat{\delta}$, mostrando su conexión con la distancia de Hamming usada en el caso simétrico. Lo más relevante será que la función δ no es simétrica, pero sí satisface la desigualdad triangular en general; y por el contrario, $\hat{\delta}$ es simétrica pero no cumple la desigualdad triangular, es decir cada una de ella cumple una propiedad de la distancia de Hamming pero no es posible encontrar un parámetro que cumpla ambas a la vez. Veremos que cada una de estas funciones de discrepancia define un parámetro fundamental de un código para el canal binario asimétrico, que llamaremos discrepancia mínima y discrepancia mínima simétrica respectivamente. Mostraremos también que estos parámetros proporcionan dos cotas superiores diferentes e incomparables para la probabilidad de fallo del decodificador de máxima verosimilitud, evidenciando así que ambas nociones de discrepancia son independientes la una de la otra y por lo tanto ambas relevantes.

3.1. Definiciones básicas y notación

Denotaremos, como hasta ahora, al cuerpo de dos elementos por $\mathbb{F}_2 = \{0, 1\}$ y al conjunto de los números naturales por $\mathbb{N} = \{0, 1, 2, \dots\}$. Además, $n \geq 2$ será un número natural que denotara la longitud de las palabras del código. Vamos a estudiar una familia de canales ruidosos indexada por un par de números reales (p, q) que pertenecen al intervalo $[0, 1/2)$.

Definición 3.1. Sea $0 \leq p \leq q < 1/2$ dos números reales. Definimos las siguientes probabilidades de transición del canal:

$$P_{p,q}(1 \mid 0) := p, \quad P_{p,q}(0 \mid 0) := 1 - p,$$
$$P_{p,q}(0 \mid 1) := q, \quad P_{p,q}(1 \mid 1) := 1 - q.$$

Por ejemplo la probabilidad de que se envíe un cero y cambie a un uno al atravesar el canal es p, es decir, la probabilidad condicionada de habiéndose enviado un cero llegue un uno.

Definición 3.2. El *canal binario asimétrico* asociado con (n, p, q) es el trío $K_n = (\mathbb{F}_2^n, \mathbb{F}_2^n, P_n^{p,q})$, donde $P_n^{p,q} : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{R}$ es la función definida por

$$P_n^{p,q}(y \mid x) = \prod_{i=1}^{n} P_{p,q}(y_i \mid x_i), \quad \text{para todo } x, y \in \mathbb{F}_2^n.$$

Donde esta última función calcula la probabilidad de recibir el vector $y \in \mathbb{F}_2^n$ habiéndose enviado la palabra del código $x \in \mathcal{C} \subset \mathbb{F}_2^n$, a través del canal asimétrico que tiene como probabilidades de transición p y q. Lo que es lo mismo la función $P_n^{p,q}$ expresa el producto de las probabilidades de transición cuando el canal se ha utilizado n veces, donde n es la longitud de los vectores x e y.

Es decir siguiendo la Definición 1.25, que presentamos en el primer capítulo, el alfabeto fuente son los elementos del código que naturalmente tienen coeficientes en $\mathbb{F}_2$ y longitud n, mientras que el alfabeto de salida son todos los vectores de $\mathbb{F}_2^n$, y finalmente el conjunto de las probabilidades se obtiene evaluando la función $P_n^{p,q}$ en cada una de las palabras de salida condicionadas a las de entrada, todas ellas se pueden calcular a partir de las probabilidades de transición del canal de la Definición 3.1.

Esta definición modela un canal discreto sin memoria donde el ruido actúa de forma independiente en cada componente de un vector binario. La suposición $p \leq q$ implica que es más probable que un 1 se convierta en 0, que un 0 pase a ser un 1 ya que el fallo de una neurona en su activación tiene probabilidad menor que el disparo de una neurona que no es el objetivo del estímulo. En la Figura 3.1, podemos ver con más claridad el funcionamiento de este canal y el efecto que tiene el

```
def transmit_unsafe(msg,p,q):

    for i in range(len(msg)):

        if msg[i]==0:

            if random() <= p:
                    msg[i] = msg[i] + 1

        else :
            if random() <= q:
                     msg[i] = msg[i] + 1
    return msg
```

```
>>> msg = vector(F,[0,1,1])
>>> transmit_unsafe(msg,0.1,0.4)
    (0, 0, 1)
>>> transmit_unsafe(msg,0.1,0.4)
    (0, 1, 0)
```

Figura 3.1: La función *transmit unsafe*, permite ver el efecto de un canal binario asimétrico con probabilidades de transición p y q sobre un vector con coeficientes en el cuerpo $\mathbb{F}_2$. También se incluyen dos ejecuciones, en las que primero se define el vector sobre el cuerpo de dos elementos y luego podemos ver como el ruido del canal ha modificado alguno de sus dígitos.

ruido sobre los vectores que lo atraviesan.

Para simplificar toda la discusión que se presenta en este capítulo es necesario introducir la siguiente notación.

Nota 3.3. Para dos vectores $x, y \in \mathbb{F}_2^n$, definimos:

$$d_{00}(y,x) := |\{i : y_i = x_i = 0\}|, \quad d_{01}(y,x) := |\{i : y_i = 0 \text{ y } x_i = 1\}|,$$

$$d_{11}(y,x) := |\{i : y_i = x_i = 1\}|, \quad d_{10}(y,x) := |\{i : y_i = 1 \text{ y } x_i = 0\}|.$$

Por ejemplo, d_{00} denota el número de dígitos que son cero en ambos vectores, mientras que d_{01} denota el número de posiciones que toman el valor 0 en y y el valor 1 en el vector x.

El siguiente resultado preliminar proporciona una expresión alternativa y útil para la función de probabilidades de transición $P_n^{p,q}$, que utilizaremos posteriormente en lugar de la definición.

Lema 3.4. *Para todo $x, y \in \mathbb{F}_2^n$, se tiene que*

$$P_n^{p,q}(y \mid x) = \left(\frac{q}{1-p}\right)^{d_{01}(y,x)} \left(\frac{p}{1-q}\right)^{d_{10}(y,x)} (1-q)^{\omega_H(y)}(1-p)^{n-\omega_H(y)},$$

donde se entiende que $0^0 := 1$.

Demostración. Usando la definición de $P_n^{p,q}(y \mid x)$ y los valores de $d_{00}(y,x)$ y $d_{11}(y,x)$, es directo obtener la expresión deseada. Por definición, la probabilidad condicional $P_n^{p,q}(y \mid x)$ se calcula como:

$$P_n^{p,q}(y \mid x) = \prod_{i=1}^{n} P_{p,q}(y_i \mid x_i),$$

donde $P_{p,q}(y_i \mid x_i)$ depende de los valores de x_i y y_i:

$$P_{p,q}(y_i \mid x_i) = \begin{cases} 1-p, & \text{si } x_i = y_i = 0, \\ 1-q, & \text{si } x_i = y_i = 1, \\ p, & \text{si } x_i = 0, y_i = 1, \\ q, & \text{si } x_i = 1, y_i = 0. \end{cases}$$

El número de posiciones en las que ocurre el primer caso será la cantidad que hemos denotado $d_{00}(y,x)$, análogamente el número de posiciones donde $x_i = 1$ y $y_i = 1$ es $d_{11}(y,x)$, luego también $d_{01}(y,x)$ es el número de posiciones donde $x_i = 0, y_i = 1$, y $d_{10}(y,x)$ donde $x_i = 1, y_i = 0$. Sustituyendo en la expresión del producto de probabilidades obtenemos:

$$P_n^{p,q}(y \mid x) = (1-p)^{d_{00}}(1-q)^{d_{11}}p^{d_{01}}q^{d_{10}}.$$

Teniendo en cuenta que el peso de hamming del vector y es el número de dígitos que tiene iguales a uno $\omega_H(x) := |\{1 \leq i \leq n : x_i = 1\}|$.

$$\omega_H(y) = d_{10}(y,x) + d_{11}(y,x), \quad d_{11}(y,x) = \omega_H(y) - d_{10}(y,x),$$

y por otro lado el número de ceros del vector y será n menos su peso de hamming luego tenemos

$$n - \omega_H(y) = d_{01}(y,x) + d_{00}(y,x), \quad \text{y} \quad d_{00}(y,x) = n - \omega_H(y) - d_{01}(y,x).$$

Sustituyendo finalmente estos valores se tiene la expresión buscada.

$$P_n^{p,q}(y \mid x) = \left(\frac{q}{1-p}\right)^{d_{01}(y,x)} \left(\frac{p}{1-q}\right)^{d_{10}(y,x)} (1-q)^{\omega_H(y)}(1-p)^{n-\omega_H(y)},$$

□

Esto es lo que adelantábamos en el capítulo anterior, hemos reescrito la función de probabilidad condicionada de recibir el vector y habiéndose enviado la palabra del código x como una función que depende solo de las probabilidades de transmisión del canal asimétrico.

3.2. Parámetros de discrepancia

En esta sección vamos a definir dos funciones cuyo objetivo es cuantificar como de diferentes son dos vectores binarios en $\mathbb{F}_2^n$. Para el resto de este capítulo los dos números reales p y q son fijos y satisfacen como hasta ahora que $0 \leq p \leq q < \frac{1}{2}$.

Definición 3.5. El *parámetro* $\gamma_{p,q}$ se define como

$$\gamma_{p,q} := \log_{\frac{q}{1-p}} \left(\frac{p}{1-q}\right) \in \mathbb{R} \cup \{+\infty\},$$

con la convención de que $\gamma_{p,q} = +\infty$ si $p = 0$, independientemente del valor de q.

Vamos a emplear el parámetro $\gamma_{p,q}$ para definir una función de discrepancia entre vectores binarios. Empezaremos con las siguientes propiedades sobre p, q y $\gamma_{p,q}$ que nos serán de gran utilidad después:

```
def prob_condicionada(y, x, p, q):

    d10 = sum(1 for i in range(len(y)) if y[i] == 1 and x[i] == 0)

    d01 = sum(1 for i in range(len(y)) if y[i] == 0 and x[i] == 1)

    P = (q / (1 - p)) ** d01 * (p / (1 - q)) ** d10
        * (1 - q) ** hamming_weight(y)
        * (1 - p) ** (len(y) - hamming_weight(y))

    return P
```

```
>>> x = [0,0,1,0,1,0]
>>> y = [1,1,1,0,1,0]
>>> prob_condicionada(y,x, 0.1,0.3)

    0.00396900000000000

>>> x = [0,1,1,0,1,0]
>>> y = [0,1,0,0,1,0]
>>> prob_condicionada(y,x, 0.1,0.3)

    0.107163000000000
```

Figura 3.2: La función *probabilidad condicionada* permite el cálculo de la probabilidad de recibir el vector y sabiendo que se ha enviado la palabra del código x a través del canal binario asimétrico de parámetros p y q. Se ha utilizado para hacer este cálculo la expresión de la función $P_n^{p,q}$ que se proporciona el Lema 3.4.
En el primer ejemplo que se presenta se han modificado dos ceros y se han convertido en unos al atravesar el canal, mientras que en el segundo ejemplo solo un uno se ha modificado y se ha convertido en un cero, debido a que la probabilidad de que ocurra este último cambio es más alta (la probabilidad de cambio de un uno a un cero, es q que es mayor que p por definición), la probabilidad condicionada en el segundo ejemplo es notablemente más alta. Es decir, los resultados obtenidos concuerdan con lo esperado teóricamente.

Lema 3.6. *Se cumplen estas dos propiedades:*

1. $0 \leq \frac{p}{1-q} \leq \frac{q}{1-p} < 1, \qquad \frac{p}{1-q} = \frac{q}{1-p} \iff p = q$
2. $\gamma_{p,q} \geq 1, \qquad \gamma_{p,q} = 1 \iff p = q$

Es decir, en ambos casos se tiene la igualdad cuando estamos en el caso simétrico $p = q$.

Demostración. Dado que $0 \leq p \leq q < \frac{1}{2}$, se tiene que $1 - p > \frac{1}{2}$ y $1 - q > \frac{1}{2}$, por lo tanto $0 \leq \frac{p}{1-q} < 1$ y $0 \leq \frac{q}{1-p} < 1$. Si $p = q$, entonces es claro que $\frac{p}{1-q} = \frac{q}{1-p}$. Por otro lado, si $p < q$, como $p + q < 1$, tenemos que:

$$(p + q)(q - p) < q - p, \quad \text{y operando } pq - p^2 + q^2 - pq < q - p$$

$$q^2 - p^2 < q - p, \quad p - p^2 < q - q^2, \quad p(1 - p) < q(1 - q),$$

lo cual implica que $\frac{p}{1-q} < \frac{q}{1-p}$ como queríamos.
Finalmente, como $\frac{p}{1-q} \leq \frac{q}{1-p} < 1$, se concluye que $\gamma_{p,q} \geq 1$. La igualdad ocurre únicamente si $p = q > 0$, como se deduce directamente de la definición de $\gamma_{p,q}$. □

Vamos con la definición del primero de los parámetro que vamos a utilizar.

Definición 3.7. La *discrepancia* entre $y, x \in \mathbb{F}_2^n$ se define como:

$$\delta_{p,q}(y, x) := \gamma_{p,q} d_{10}(y, x) + d_{01}(y, x),$$

con la convención de que $+\infty \cdot 0 = 0$.

Notemos que si $p = q$, entonces $\gamma_{p,q} = 1$ y la discrepancia $\delta_{p,q}$ coincide con la distancia de Hamming, denotada como $d_H(x, y)$, la cual habíamos definido como $d_H(x, y) = |\{i : x_i \neq y_i\}|$ para $x, y \in \mathbb{F}_2^n$. Sin embargo, si $p < q$, la medida de discrepancia $\delta_{p,q}$ así definida no es simétrica en general. Esta asimetría se puede observar mejor en los ejemplos de la Figura 3.3, donde se proporciona el código para el cálculo de la discrepancia.

Definición 3.8. La *discrepancia simétrica* entre dos vectores $y, x \in \mathbb{F}_2^n$ se define como:

$$\hat{\delta}_{p,q}(y, x) := \delta_{p,q}(y, x) - \omega_H(y)(\gamma - 1),$$

donde $\omega_H(y)$ es el peso de Hamming del vector y. La implementación mediante código en *SageMath* se muestra en la Figura 3.4.

Como veremos, las funciones $\delta_{p,q}$ y $\hat{\delta}_{p,q}$ tienen propiedades matemáticas muy diferentes. Primeramente vamos a comprobar que esta función así definida verdaderamente es simétrica, para ello probaremos primero este resultado preliminar.

Lema 3.9. *Sean $x, y \in \mathbb{F}_2^n$ se cumplen las siguientes propiedades:*

1. $d_{10}(y, x) = d_{10}(x, y) + \omega_H(y) - \omega_H(x)$,
2. $d_{01}(y, x) = d_{01}(x, y) + \omega_H(x) - \omega_H(y)$,
3. $\delta_{p,q}(y, x) = \delta_{p,q}(x, y) + (\omega_H(y) - \omega_H(x))(\gamma_{p,q} - 1)$,
4. $\delta_{p,q}(y, x) = d_H(y, x) + (\gamma_{p,q} - 1) d_{10}(y, x)$,
5. $\hat{\delta}_{p,q}(y, x) = d_H(y, x) - (\gamma_{p,q} - 1) d_{11}(y, x)$.

```
def discrepancia(y, x, p, q):

    d10 = sum(1 for i in range(len(y)) if y[i] == 1 and x[i] == 0)
    d01 = sum(1 for i in range(len(y)) if y[i] == 0 and x[i] == 1)

    gamma_value = ma.log(p / (1 - q), q / (1 - p))

    dis = gamma_value * d10 + d01
    return dis


>>> y = [1, 0, 0, 1]
>>> x = [0, 1, 0, 0]
>>> discrepancia(y, x, 0.05, 0.2)

    4.558833615477598

>>> y = [0, 0, 0, 1]
>>> x = [0, 1, 1, 1]
discrepancia(y, x, 0.05, 0.2)

    2.0
```

Figura 3.3: La función *discrepancia* calcula la discrepancia no simétrica entre dos vectores binarios en función de los parámetros del canal binario asimétrico p y q.
En los ejemplos que se presentan se puede ver que los vectores x e y tanto en el primer ejemplo como en el segundo difieren en el valor de dos de sus posiciones. Sin embargo, existe una diferencia, en el primero de los casos, son dos ceros lo que se han convertido en unos, mientras que en el segundo, son dos unos, lo que tras pasar por el canal se han convertido en ceros. Esto es significativo por la diferencia entre los valores p y q, entre las probabilidades respectivamente de que un cero pase a ser un uno, y de que un uno pase a ser un cero tras atravesar el canal.
La discrepancia en el primero de los ejemplos es más del doble que la que obtenemos en el segundo, además las perturbaciones que han ocurrido en el primer caso tienen una probabilidad de ocurrir de tan solo $p = 0{,}05$. Mientras que las que se han producido en el segundo caso, que tienen una probabilidad de $q = 0{,}2$. Luego, podemos observar que cuando se ha producido un cambio en la palabra que es muy poco probable que ocurriera, como en el ejemplo primero, la discrepancia del vector resultante y el original es mayor que si se produce una perturbación que sea mucho más probable, como la que ocurre en el segundo ejemplo. Esto esta de acuerdo con la relación que se establece en el Teorema 3.15 entre la probabilidad condicionada de enviando la palabra x se obtenga el vector y, es decir $P^n_{p,q}(y|x)$ y la discrepancia entre ambos y en este orden $\delta_{p,q}(y, x)$.

```
def discrepancia_simetrica(y,x,p,q):

    dis_sim = discrepancia(y,x,p,q) - hamming_weight(y)*(gamma(p,q)-1)

    return dis_sim

>>> p = 0.2
>>> q = 0.3
>>> x = [0,0,1,0]
>>> y = [0,1,0,0]
>>> discrepancia_simetrica(y,x,p,q)

    2.0

>>> discrepancia_simetrica(x,y,p,q)

    2.0
```

Figura 3.4: La función *discrepancia simétrica* calcula la discrepancia simétrica entre dos vectores binarios x e y que atraviesan un canal binario asimétrico con probabilidades de transición p y q. En el ejemplo que se muestra podemos comprobar que la función es verdaderamente simétrica, es decir no importa el orden en el que se coloquen los vectores y, x, como si era el caso en la discrepancia no simétrica.

Demostración. Para demostrar la primera y la segunda igualdad, observemos que se tienen las dos igualdades siguientes

$$d_{10}(y,x) + d_{11}(y,x) = \omega_H(y) \quad \text{y} \quad d_{01}(y,x) + d_{11}(y,x) = \omega_H(x).$$

Es decir podemos reescribir d_{10} y d_{01} como sigue:

$$d_{10}(y,x) = \omega_H(y) - d_{11}(y,x) = \omega_H(y) - d_{11}(x,y) = \omega_H(y) - \omega_H(x) + d_{10}(x,y).$$

$$d_{01}(y,x) = \omega_H(x) - d_{11}(y,x) = \omega_H(y) - d_{11}(x,y) = \omega_H(x) - \omega_H(y) + d_{01}(x,y).$$

La tercera igualdad es una consecuencia de la primera y la segunda. Si recordamos que la discrepancia se definía como $\delta_{p,q}(y,x) = \gamma_{p,q} d_{10}(y,x) + d_{01}(y,x)$, entonces

$$\delta_{p,q}(y,x) = \gamma_{p,q}\left(d_{10}(x,y) + \omega_H(y) - \omega_H(x)\right) + d_{01}(x,y) + \omega_H(x) - \omega_H(y) =$$

$$= \delta_{p,q}(x,y) + (\omega_H(y) - \omega_H(x))(\gamma_{p,q} - 1).$$

La cuarta igualdad se deduce del hecho de que $d_H(y,x) = d_{10}(y,x) + d_{01}(y,x)$, despejando $d_{01}(y,x)$ y sustituyendo su valor en la expresión de δ, se tiene

$$\delta_{p,q}(y,x) = \gamma_{p,q} d_{10}(y,x) + d_{01}(y,x) = \gamma_{p,q} d_{10}(y,x) + d_H(y,x) - d_{10}(y,x) =$$

$$= d_H(y,x) + (\gamma_{p,q} - 1) d_{10}(y,x).$$

Finalmente, combinando la cuarta igualdad con $d_{10}(y,x) + d_{11}(y,x) = \omega_H(y)$, podemos probar la última igualdad,

$$\hat{\delta}_{p,q}(y,x) = \delta_{p,q}(y,x) - \omega_H(y)(\gamma_{p,q} - 1) = d_H(y,x) - (\gamma_{p,q} - 1) d_{11}(y,x).$$

□

Proposición 3.10. *Para todo $x, y \in \mathbb{F}_2^n$, se cumple que $\hat{\delta}_{p,q}(y,x) = \hat{\delta}_{p,q}(x,y)$. Es decir que la discrepancia simétrica verdaderamente lo es.*

Demostración. Este resultado se deduce del lema anterior de la última igualdad, ya que la distancia de Hamming es simétrica $d_H(y,x) = d_H(x,y)$ y además para contar el número de unos que coinciden en posición en dos palabras no importa el orden tampoco $d_{11}(x,y) = d_{11}(y,x)$ para todo $x, y \in \mathbb{F}_2^n$. □

3.2.1. Decodificador de máxima verosimilitud y de discrepancia mínima

Definición 3.11. Para un código $\mathcal{C} \subseteq \mathbb{F}_2^n$, el *decodificador de máxima verosimilitud* se define como la función $D_{\mathcal{C}}^{ML} : \mathbb{F}_2^n \to \mathcal{C} \cup \{e\}$ tal que:

$$D_{\mathcal{C}}^{ML}(y) := \begin{cases} x, & \text{si } x \text{ es la única palabra de } \mathcal{C} \text{ que maximiza la función } P_{p,q}^n(y|x), \\ e, & \text{en cualquier otro caso,} \end{cases}$$

donde $e \notin \mathbb{F}_2^n$ denota un mensaje de error.

Este decodificador es el mismo del que hemos hablado en el capítulo anterior, en la Definición 2.27, es decir, encuentra la palabra del código x (en el caso de que sea única) que tiene la mayor probabilidad de entre todas las del código, de que habiéndose recibido el vector y, fuera exactamente esa la que se envió por el canal. Ahora ya podemos implementarlo, ya que hemos descrito la función probabilidad condicionada y que esta se puede escribir en función del los parámetros del canal.

Proposición 3.12. *El decodificador de máxima verosimilitud se puede calcular directamente a partir de los parámetros del canal. Si tenemos un canal binario asimétrico con parámetros p y q tenemos*

$$D_{\mathcal{C}}^{ML}(y) = \arg\max_{x \in C} \left((x \cdot y) \ln \left(\frac{(1-p)(1-q)}{pq} \right) - \omega_H(x) \ln \left(\frac{1-p}{q} \right) \right).$$

Demostración. Recordemos entonces que este decodificador está dado por

$$D_{\mathcal{C}}^{ML}(y) = \arg\max_{c \in \mathcal{C}} P(\text{recibido} = y \mid \text{enviado} = x),$$

donde $y \in \mathbb{F}_2^n$ es la palabra recibida, tenemos una población de n neuronas, y $\mathcal{C}$ es el código neural. Podemos reescribir esta probabilidad, usando el Lema 3.6 como

$$P(\text{recibido} = y \mid \text{enviado} = x) =$$

$$P_n^{p,q}(y \mid x) = \left(\frac{q}{1-p} \right)^{d_{01}(y,x)} \left(\frac{p}{1-q} \right)^{d_{10}(y,x)} (1-q)^{\omega_H(y)} (1-p)^{n-\omega_H(y)}.$$

Usando también, que el número de unos en el vector y, su peso de Hamming, serán los que coinciden originalmente con unos de x más los que eran ceros en x y ahora son unos. Tenemos las siguientes igualdades:

$$d_{10}(y,x) + d_{11}(y,x) = \omega_H(y) \quad \text{y} \quad d_{01}(y,x) + d_{00}(y,x) = n - \omega_H(y),$$

sustituyendo entonces en la expresión obtenemos

$$P_n^{p,q}(y \mid x) = (1-p)^{d_{00}(y,x)} p^{\omega_H(y) - d_{11}(y,x)} (1-q)^{d_{11}(y,x)} q^{n - \omega_H(y) - d_{00}(y,x)}.$$

Al maximizar sobre $x \in \mathcal{C}$, podemos ignorar los términos que no dependen de la palabra del código x, y obtenemos

$$D_{\mathcal{C}}^{ML}(y) = \arg\max_{x \in \mathcal{C}} \left((1-p)^{d_{00}(y,x)} p^{-d_{11}(y,x)} (1-q)^{d_{11}(y,x)} q^{-d_{00}(y,x)} \right) =$$

$$= \arg\max_{x \in \mathcal{C}} \left(\left(\frac{1-p}{q} \right)^{d_{00}(y,x)} \left(\frac{1-q}{p} \right)^{d_{11}(y,x)} \right).$$

Debido a la monotonía de la función logaritmo, sabemos que los extremos de la función original se alcanzarán en los mismos puntos que los extremos del logaritmo de la función original [6].

$$D_{\mathcal{C}}^{ML}(y) = \arg\max_{x\in\mathcal{C}} \left(\left(d_{00}(y,x) \ln \frac{1-p}{q} + d_{11}(y,x) \ln \frac{1-q}{p} \right) \right) \qquad (*)$$

Si ahora usamos el hecho de que en el cuerpo $\mathbb{F}_2$ se tiene que $ab = 1 \iff a = 1, b = 1$ podemos escribir $d_{11}(y,x) = x \cdot y$, y finalmente usando el principio de inclusión exclusión observamos que el número de ceros en las dos palabras a la vez, será el número de posiciones totales, menos las posiciones que son igual a 1 en cada una de las dos palabras más las posiciones que son igual a 1 en ambas

$$d_{00}(y,x) = (\mathbb{1} - x) \cdot (\mathbb{1} - y) = n - \omega_H(x) - \omega_H(y) + x \cdot y,$$

donde $\mathbb{1} \in \mathbb{F}_n^2$ representa el vector formado por todo unos.

$$D_{\mathcal{C}}^{ML}(y) = \arg\max_{x\in\mathcal{C}} \left(\left((n - \omega_H(x) - \omega_H(y) + x \cdot y) \ln \frac{1-p}{q} + (x \cdot y) \ln \frac{1-q}{p} \right) \right)$$

Nuevamente podemos ignorar todos los términos que no dependen de x, y sacando factor común al producto escalar $x \cdot y$, obtenemos la expresión buscada en función de los parámetros del canal

$$D_{\mathcal{C}}^{ML}(y) = \arg\max_{x\in\mathcal{C}} \left((x \cdot y) \ln \frac{(1-p)(1-q)}{pq} - \omega_H(x) \ln \frac{1-p}{q} \right).$$

□

Dado que asumimos $p, q < \frac{1}{2}$, el decodificador de máxima verosimilitud devuelve así una palabra de código x que maximiza el producto escalar $x \cdot y$ con la palabra recibida y, sujeto a un término de penalización proporcional a su peso $\omega_H(x)$. En otras palabras, este decodificador devuelve la palabra de código c que maximiza el número de dígitos iguales a 1 coincidentes con los de y mientras minimiza la introducción de números 1 adicionales. Esto es debido, a que la activación de una neurona, cuando el estímulo no pertenece a su campo receptivo, es más difícil que se produzca, que el fallo en la activación de una neurona, cuando el estímulo sí que pertenece a su campo receptivo, como ya se había mencionado.

Nota 3.13 (**Canal binario simétrico**)**.** Para $p = q < \frac{1}{2}$, tenemos el caso particular del canal binario simétrico y sustituyendo en la ecuación $(*)$ podemos encontrar la palabra del código estimada, maximizando la función

$$D_{\mathcal{C}}^{ML}(y) = \arg\max_{c\in C} (n - d_H(x,y)) = \arg\min_{x\in C} d_H(x,y),$$

donde $d_H(x,r) = d_{01} + d_{10} = |\{i \in [n] \mid x_i \neq y_i\}|$ es la distancia de Hamming entre dos palabras del código $\mathcal{C}$, el número de posiciones en la palabras en las que difieren. Este es el conocido resultado de que la decodificación de máxima verosimilitud es equivalente a la decodificación por el elemento más cercano, respecto a la distancia de Hamming [10].

```
def probabilidad2(x,y,p,q):

    d11 = sum(1 for i in range(len(y)) if y[i] == 1 and x[i] == 1)
    log1 = ma.log(((1-p)*(1-q))/(p*q))
    log2 = ma.log((1-p)/q)
    peso = peso_hamming(x)

    return  d11*log1 - peso*log2

def decodificadorML(C,y,p,q):

    s=1

    prob_max = max(probabilidad2(c,y,p,q) for c in C)
    mejores_c = [c for c in C if probabilidad2(c,y,p,q) == prob_max]

    if len(mejores_c)!=1:
        s=0

    return s, mejores_c[0]
```

Figura 3.5: La función *probabilidad2* calcula la expresión de la función a maximizar para el decodificador de máxima verosimilitud $D_{\mathcal{C}}^{ML}$. Esta expresión no es una probabilidad, pero maximizar esta función es equivalente como hemos visto, a maximizar la probabilidad condicionada de sabiendo que se ha recibido y cual es la probabilidad de que haya sido enviada la palabra del código x a través de un canal binario asimétrico de parámetros p y q, es decir, por como se ha construido sus extremos se encuentran en los mismos puntos.
La función *decodificadorML* implementa el decodificador de máxima verosimilitud para un código binario que se envía través de un canal asimétrico de parámetros p y q. Devolviendo la palabra del código x en el caso de ser única que cumple que maximiza la probabilidad de recibido el vector y haber sido enviada a través del canal. En el caso de no ser única esta palabra, es el parámetro s el que nos indica que la decodificación no se ha producido de forma correcta, tomando el valor 0, esto equivale al mensaje de error que se recibe en caso de no ser única la decodificación.

Ejemplo 3.14. Supongamos $p = 0{,}1$ y $q = 0{,}4$. Si tenemos el código formado por las siguientes palabras:

$$\mathcal{C} = \{(0,0,0),(0,1,0),(1,1,0),(1,1,1)\} \subseteq \mathbb{F}_2^4,$$

y sea $y = (0,0,1)$ el vector recibido.
Si usamos el decodificador de máxima verosimilitud, obtenemos la siguiente función a maximizar:

$$P_3^{p,q}(y \mid x) = \left(\frac{0,4}{0,9}\right)^{d_{01}(y,x)} \left(\frac{0,1}{0,6}\right)^{d_{10}(y,x)} (0,6)^1(0,9)^{3-1}$$

Calculando la probabilidad condicionada de que llegue y a que se haya enviado cada una de las cuatro palabras del código obtendremos que la única que maximiza esa probabilidad es $(1,1,1)$ y por tanto se tiene que $D_{\mathcal{C}}(y) = (1,1,1)$.

$$P_3^{p,q}(y \mid (0,0,0)) = \left(\frac{0,4}{0,9}\right)^0 \left(\frac{0,1}{0,6}\right)^1 (0,6)^1(0,9)^2 = 0,081$$

$$P_3^{p,q}(y \mid (0,1,0)) = \left(\frac{0,4}{0,9}\right)^1 \left(\frac{0,1}{0,6}\right)^1 (0,6)^1(0,9)^2 = 0,036$$

$$P_3^{p,q}(y \mid (1,1,0)) = \left(\frac{0,4}{0,9}\right)^2 \left(\frac{0,1}{0,6}\right)^1 (0,6)^1(0,9)^2 = 0,016$$

$$P_3^{p,q}(y \mid (1,1,1)) = \left(\frac{0,4}{0,9}\right)^2 \left(\frac{0,1}{0,6}\right)^0 (0,6)^1(0,9)^2 = 0,096$$

Podemos comprobarlo también con la función programada en *SageMath* para la probabilidad condicionada.

```
>>> p = 0.1
>>> q = 0.4
>>> C = [[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 1, 1]]
>>> y = [0,0,1]
>>> for i in C:
        print('P(',y,'|',i,') = ',prob_condicionada(y,i,p,q))

P( y | [0, 0, 0] ) =  0.0810000000000000
P( y | [0, 1, 0] ) =  0.0360000000000000
P( y | [1, 1, 0] ) =  0.0160000000000000
P( y | [1, 1, 1] ) =  0.0960000000000000
```

La relación entre la probabilidad condicionada y la discrepancia, que se describe en el siguiente teorema, nos va a permitir definir otro decodificador más sencillo pero equivalente. En el caso del canal binario simétrico, veíamos en la Nota 3.13 que el decodificador de máxima verosimilitud era equivalente a minimizar la distancia de Hamming, de manera análoga vamos a ver que en el caso asimétrico el decodificador de máxima verosimilitud será equivalente a otro que minimice la discrepancia no simétrica, la nueva noción que hemos definido en el canal binario asimétrico para medir diferencias entre vectores.

Teorema 3.15. *Sean $x, x', y \in \mathbb{F}_2^n$. Las siguientes afirmaciones son equivalentes:*

1. $\delta_{p,q}(y,x) < \delta_{p,q}(y,x')$.
2. $P_{p,q}^n(y|x) > P_{p,q}^n(y|x')$.

Demostración. Por el Lema 3.4, sabemos que probar que $P^n(y \mid x') < P^n(y \mid x)$ es equivalente a demostrar que

$$\left(\frac{q}{1-p}\right)^{d_{01}(y,x')}\left(\frac{p}{1-q}\right)^{d_{10}(y,x')} < \left(\frac{q}{1-p}\right)^{d_{01}(y,x)}\left(\frac{p}{1-q}\right)^{d_{10}(y,x)},$$

ya que los demás términos dependen de y y asumimos que $0^0 = 1$.
Supongamos primero que $p > 0$. Si en la desigualdad anterior primero tomamos logaritmos en base $\frac{q}{1-p}$ y luego tomamos potencias en base $\frac{q}{1-p}$ a ambos lados, tendríamos de nuevo la desigualdad anterior, para mayor claridad vamos a operar primero tomando logaritmos.

$$log_{\frac{q}{1-p}}\left[\left(\frac{q}{1-p}\right)^{d_{01}(y,x')}\left(\frac{p}{1-q}\right)^{d_{10}(y,x')}\right] =$$

$$= d_{01}(y,x') + d_{10}(y,x')log_{\frac{q}{1-p}}\left(\frac{p}{1-q}\right) = \delta_{p,q}(y,x').$$

Luego operando de la misma forma en el otro término y tomando potencias en ambos lados en la base $\frac{q}{1-p}$, llegamos a la siguiente desigualdad, que es equivalente a la primera.

$$\left(\frac{q}{1-p}\right)^{\delta_{p,q}(y,x')} < \left(\frac{q}{1-p}\right)^{\delta_{p,q}(y,x)}.$$

Dado que $\frac{q}{1-p} < 1$ por lo expuesto en el Lema 3.6, la desigualdad anterior se cumple si y solo si $\delta_{p,q}(y,x) < \delta_{p,q}(y,x')$, tal y como se deseaba.

Ahora suponemos que $p = 0$ y q es arbitrario (también podría ser $q = 0$). Entonces, la desigualdad inicial es equivalente a

$$q^{d_{01}(y,x')}0^{d_{10}(y,x')} < q^{d_{01}(y,x)}0^{d_{10}(y,x)}.$$

Esto sucede si y solo si se cumple una de las siguientes condiciones:

- $d_{10}(y,x') = d_{10}(y,x) = 0$ y $d_{01}(y,x') > d_{01}(y,x)$,
- $d_{10}(y,x) = 0$ y $d_{10}(y,x') > 0$.

En ambos casos por definición de $\delta_{p,q}$, esto es equivalente a $\delta_{p,q}(y,x) < \delta_{p,q}(y,x')$. □

El Teorema 3.15 establece entonces que podemos definir un decodificador equivalente al de máxima verosimilitud $D_{\mathcal{C}}$, uno que minimice la discrepancia entre la palabra recibida y las palabras del código, que vamos a denotar por tanto decodificador de discrepancia mínima y que tiene la ventaja de que la expresión de cálculo es más sencilla, tal y como adelantábamos antes.

Definición 3.16. El *decodificador de discrepancia mínima* $D^{\delta}_{\mathcal{C}}$ se define como:

$$D^{\delta}_{\mathcal{C}}(y) := \begin{cases} x, & \text{si } x \text{ es la única palabra de } \mathcal{C} \text{ que minimiza } \delta_{p,q}(y,x) \\ e, & \text{en cualquier otro caso} \end{cases}$$

donde $e \notin \mathbb{F}_2^n$ es un mensaje de error, como el que ya utilizamos en la definición del decodificador de máxima verosimilitud.

```
def decodificador(C, r, p,q):

    s=1
    min_dis = min(discrepancia(r, c, p, q) for c in C)
    mejores_c = [c for c in C if discrepancia(r,c,p,q) == min_dis]

    if len(mejores_c)!=1:
        s=0

    return s, mejores_c[0]
```

Figura 3.6: La función *decodificador* implementa la decodificación por discrepancia mínima para un código binario que se transmite a través del canal binario asimétrico de parámetros p y q. Esta función siempre devuelve la palabra del código más cercana a la palabra recibida r, pero el parámetro s es lo que nos indica si esta palabra es única o no lo es, tomando el valor 1 y 0 respectivamente. Cuando s toma el valor 0, indicándonos que la palabra no es única, es lo que correspondería al mensaje de error al que se hace referencia en la definición del decodificador. En el Ejemplo 3.18 y en el Ejemplo 3.19 se proporcionan dos ejemplos de uso.

Nota 3.17. Podemos implementar el decodificador de discrepancia mínima en *SageMath* como en la Figura 3.6, aunque en este punto es importante advertir que la implementación que presentamos para este canal asimétrico no es eficiente, es una búsqueda donde el número de operaciones escala de forma exponencial proporcionalmente a n, la dimensión del espacio vectorial en el que se encuentran las palabras del código. Sin embargo, ni siquiera existen, para códigos de un canal simétrico, algoritmos genéricos de decodificación eficientes, solo existen para algunos tipos de códigos específicos que presenten algunas características especiales como es el caso, por ejemplo, de los códigos Reed-Solomon.

Es natural también, tratar de comparar también este decodificador de discrepancia mínima $D^{\delta}_{\mathcal{C}}$ (o, equivalentemente, el decodificador de máxima verosimilitud $D_{\mathcal{C}}$) con el decodificador que minimiza la distancia de Hamming, denotado por $D^{H}_{\mathcal{C}}$, que se usa en el caso de los canales simétricos pero que también podemos aplicar en el caso asimétrico. En general, estos dos decodificadores son diferentes, veamos un ejemplo.

Ejemplo 3.18. Vamos a considerar el código del ejemplo anterior formado por cuatro palabras:

$$\mathcal{C} = \{(0,0,0),(0,1,0),(1,1,0),(1,1,1)\} \subseteq \mathbb{F}_2^3,$$

con los parámetros $p = 0,1$ y $q = 0,4$ de nuevo. Además, sabíamos que el vector que se había recibido tras pasar por el canal era $y = (0,0,1)$.

Podemos comprobar que $D^{H}_{\mathcal{C}}(y) = (0,0,0)$, es decir usando la distancia de Hamming como métrica la palabra del código $(0,0,0) \in \mathcal{C}$ es la más cercana al vector recibido y.

$$d_H(y,(0,0,0)) = 1 \quad d_H(y,(0,1,0)) = 2 \quad d_H(y,(1,1,0)) = 3 \quad d_H(y,(1,1,1)) = 2$$

Vamos a intentar decodificar ahora este vector $y = (0,0,1)$ con el decodificador de discrepancia mínima $D^{\delta}_{\mathcal{C}}$ y comprobemos dos cosas, uno que obtenemos el mismo vector que con el decodificador de máxima verosimilitud $D_{\mathcal{C}}$ y dos que es distinto al que proporciona el decodificador que minimiza la distancia de Hamming.
Recordemos que el parámetro que habíamos definido toma el valor $\gamma_{p,q} \approx 2,21$, y la discrepancia se definía como:

$$\delta_{p,q}(y,x) = \gamma_{p,q} \cdot d_{10}(y,x) + d_{01}(y,x).$$

Si el vector enviado hubiera sido $x = (0,0,0)$ obtendríamos el valor para la función discrepancia $\delta_{p,q}(y,x) = \gamma_{p,q} \approx 2{,}21$, pues $d_{10}(y,x) = 1$ y $d_{01}(y,x) = 0$. Mientras que si la palabra del código enviada hubiera sido $x = (1,1,1)$ la función a minimizar toma el valor $\delta_{p,q}(y,x) = 2$, pues $d_{10}(y,x) = 0$ y $d_{01}(y,x) = 2$. Repitiendo ese proceso con las otras dos palabras del código restantes vemos que la palabra que minimiza la función discrepancia entre ella misma y el vector recibido $y \in \mathbb{F}_2^n$ es $(1,1,1)$, es decir

$$D_{\mathcal{C}}^{\delta}(y) = (1,1,1).$$

Podemos comprobar la solución también usando la función *decodificador* programada en *SageMath*.

```
>>> p = 0.1
>>> q = 0.4
>>> C = [[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 1, 1]]
>>> y = [0,0,1]
>>> s,x = decodificador(C,y,p,q)
>>> if s==1:
    print('La decodificación es única y la palabra del código buscada es:',x)

    La decodificación es única y la palabra del código buscada es: [1, 1, 1]
```

En el código de la Figura 3.6, se puede observar que cuando el parámetro s toma el valor uno, significa que se ha producido un error, es decir, que no existe una única palabra del código que minimiza la discrepancia, sino que al calcular la discrepancia, obtenemos que con varias de ellas se obtiene el mismo valor mínimo, ilustremos esto con otro ejemplo.

Ejemplo 3.19. Consideramos el código $\mathcal{C} = \{x_1, x_2, x_3\} = \{(0,0,1),(0,1,0),(1,1,1)\}$. Una palabra de envía a través del canal binario asimétrico de parámetros $p = 0{,}1$ y $q = 0{,}3$ por tanto $\gamma_{p,q} \approx 1{,}77$ y se recibe el vector $r = (0,0,0)$, si ahora intentamos aplicarle el decodificador de discrepancia mínima para obtener de vuelta que palabra es más probable que se enviara a través del canal, obtendremos lo siguiente:

$$\begin{aligned}
\delta_{p,q}(r,x_1) &= d_{01}(r,x_1)\cdot\gamma_{p,q} + d_{01}(r,x_1) = 0\cdot 1{,}77 + 1 = 1\\
\delta_{p,q}(r,x_2) &= d_{01}(r,x_2)\cdot\gamma_{p,q} + d_{01}(r,x_2) = 0\cdot 1{,}77 + 1 = 1\\
\delta_{p,q}(r,x_3) &= d_{01}(r,x_3)\cdot\gamma_{p,q} + d_{01}(r,x_3) = 0\cdot 1{,}77 + 3 = 3
\end{aligned}$$

Tenemos entonces dos palabras que minimizan la discrepancia y por tanto el decodificador para el vector recibido r nos devolverá un error.

$$D_{\mathcal{C}}^{\delta}(r) = e$$

Si planteamos este mismo ejemplo en el ordenador, el parámetro s que nos indicaba si la decodificación era única toma el valor cero, y podemos obtener también si quisiéramos una de las palabras que minimiza la discrepancia, también modificando el código para que devuelva la lista entera en vez de solo un elemento podríamos obtener todas las palabras del código que minimizan la discrepancia con la palabra recibida r.

```
>>> p = 0.1
>>> q = 0.3
>>> C = [[0, 0, 1], [0, 1, 0], [1, 1, 1]]
>>> r = [0,0,0]
>>> s,x = decodificador(C,r,p,q)
>>> if s==0: print('La decodificación no es única')
```

`La decodificación no es única`

Nota 3.20. En adelante, nos vamos enfocar en la estructura de códigos dotados de la función de discrepancia $\delta_{p,q}$. Para este análisis, excluiremos el caso extremo del canal Z, el caso en el que $p = 0$ de nuestro estudio. Por lo tanto, asumiremos siempre:

$$0 < p \leq q < \frac{1}{2}.$$

3.2.2. Discrepancia mínima y discrepancia mínima simétrica

vamos centrarnos en analizar la estructura de un código $\mathcal{C} \subseteq \mathbb{F}_2^n$. Cada una de las dos medidas de la discrepancia entre palabras que hemos definido $\delta_{p,q}$ y $\hat{\delta}_{p,q}$, define un parámetro del código de la siguiente manera. Además, para simplificar la notación, omitiremos el subíndice p, q en los símbolos, escribiendo por ejemplo P^n en lugar de $P^n_{p,q}$, y δ en lugar de $\delta_{p,q}$.

Definición 3.21. Para un código $\mathcal{C}$, definimos:

$$\delta(\mathcal{C}) := \text{mín}\{\delta(x, x') : x, x' \in C, x \neq x'\},$$

$$\hat{\delta}(\mathcal{C}) := \text{mín}\{\hat{\delta}(x, x') : x, x' \in C, x \neq x'\}.$$

A estos parámetros los llamamos *discrepancia mínima* y *discrepancia simétrica mínima* de $\mathcal{C}$, respectivamente. En la Figura 3.7, se puede consultar la implementación de estas funciones que se ha realizado en *SageMath*.

Para cualquier código $\mathcal{C}$, tenemos que la discrepancia simétrica será menor que la discrepancia no simétrica $\hat{\delta}(\mathcal{C}) \leq \delta(\mathcal{C})$. De hecho, si $x, x' \in \mathcal{C}$ satisfacen $\delta(x, x') = \delta(\mathcal{C})$, entonces como γ es siempre mayor o igual que uno y el peso de Hamming de cualquier vector binario es al menos cero, se tiene:

$$\hat{\delta}(x, x') = \delta(x, x') - \omega_H(x)(\gamma - 1) \leq \delta(x, x') = \delta(\mathcal{C}).$$

La discrepancia simétrica siempre es positiva, no ocurre lo mismo con la discrepancia asimétrica que puede no serlo, como veremos en el siguiente ejemplo, que también proporciona una vía para el cálculo de las discrepancias mínima y discrepancia mínima simétrica de un código.

Ejemplo 3.22. Sea $p = 0{,}1$ y $q = 0{,}3$, de donde se obtiene el valor aproximado de $\gamma \approx 1{,}77$. Si se tiene el código $\mathcal{C} = \{x_1, x_2, x_3\} = \{(1, 0, 0), (0, 1, 1), (1, 1, 1)\}$, podemos calcular la discrepancia mínima como, calculando las discrepancias entre todas as palabras del código:

$$\begin{aligned}
\delta(x_1, x_2) &= \gamma d_{10}((1,0,0),(0,1,1)) + d_{01}((1,0,0),(0,1,1)) = \gamma \cdot 1 + 2 \approx 3{,}77 \\
\delta(x_2, x_1) &= \gamma d_{10}((0,1,1),(1,0,0)) + d_{01}((0,1,1),(1,0,0)) = \gamma \cdot 2 + 1 \approx 4{,}54 \\
\delta(x_1, x_3) &= \gamma d_{10}((1,0,0),(1,1,1)) + d_{01}((1,0,0),(1,1,1)) = \gamma \cdot 0 + 2 = 2 \\
\delta(x_3, x_1) &= \gamma d_{10}((1,1,1),(1,0,0)) + d_{01}((1,1,1),(1,0,0)) = \gamma \cdot 2 + 0 \approx 3{,}54 \\
\delta(x_2, x_3) &= \gamma d_{10}((1,1,1),(1,1,0)) + d_{01}((1,1,1),(1,1,0)) = \gamma \cdot 1 + 0 \approx 1{,}77 \\
\delta(x_3, x_2) &= \gamma d_{10}((1,1,0),(1,1,1)) + d_{01}((1,1,0),(1,1,1)) = \gamma \cdot 0 + 1 = 1
\end{aligned}$$

Es decir el valor de la discrepancia mínima del código $\mathcal{C}$ es $\delta(\mathcal{C}) = 1$. Calcular la discrepancia mínima simétrica es más sencillo una vez que se ha calculado la mínima, ya que se puede realizar directamente con las palabras que proporcionan la discrepancia mínima.

$$\hat{\delta}(\mathcal{C}) = \hat{\delta}(x_2, x_3) = \hat{\delta}(x_3, x_2) = \hat{\delta}((0,1,1),(1,1,1)) \approx -0{,}54.$$

Podemos comprobar este resultado con las funciones que hemos construido para calcular tanto la discrepancia mínima, como la discrepancia mínima simétrica.

```
def discrepancia_minima(C,p,q):

    min_dis_c = discrepancia(C[0], C[1], p,q)

    for i in range(len(C)):
        for j in range(len(C)):

            if j>i:
                d_izq = discrepancia(C[i], C[j],p,q)
                d_der = discrepancia(C[j], C[i],p,q)
                d = min(d_izq, d_der)

                if d < min_dis_c:
                    min_dis_c = d

    return min_dis_c

def discrepancia_minima_simetrica(C,p,q):

    min_dis_sim_c = discrepancia_simetrica(C[0], C[1], p,q)

    for i in range(len(C)):
        for j in range(len(C)):

            if j>i:
                d = discrepancia_simetrica(C[i], C[j],p,q)

                if d < min_dis_sim_c:
                    min_dis_sim_c = d

    return min_dis_sim_c
```

Figura 3.7: La función *discrepancia mínima* calcula la discrepancia no simétrica más pequeña que existe entre dos palabras del código que se atraviesa un canal asimétrico de con parámetros p y q. Se calcula para ello la discrepancia entre cada pareja de palabras del código, ordenadas de las dos maneras posibles.
Por otro lado la función *discrepancia mínima simétrica* calcula la mínima discrepancia simétrica que existe entre dos palabras de un código que se envía a través de un canal asimétrico con probabilidades de transición p y q. Para realizar este cálculo se utiliza la simetría de la función y que por tanto, solo hay que computar la discrepancia simétrica una vez para cada par de palabras, ya que no importa el orden en el que se coloquen.
En el Ejemplo 3.22, podemos ver como se pueden utilizar ambas funciones. Finalmente, es oportuno comentar que estas implementaciones son búsquedas, implican el cálculo de todas las posibilidades, y esta operación de búsqueda del mínimo es el mismo calculo que se realiza para la decodificación, luego nuevamente, recalcar que tampoco para códigos en general que atraviesan canales simétricos existen algoritmos eficientes.

```
>>> c = [(1,0,0), (0,1,1), (1,1,1)]
>>> discrepancia_minima(c,0.1,0.3)

    1

>>> c = [(1,0,0), (0,1,1), (1,1,1)]
>>> discrepancia_minima_simetrica(c,0.1,0.3)

    -0.5424874983228443
```

El siguiente resultado proporciona condiciones suficientes bajo las cuales un vector $y \in \mathbb{F}_2^n$ se decodifica como una palabra código $x \in C$, en términos de las discrepancias mínimas y simétricas de $\mathcal{C}$, este resultado será importante después para establecer cotas para la probabilidad de decodificación no exitosa.

3.2.3. Condición necesaria para la decodificación

Proposición 3.23. *Sea $\mathcal{C} \subseteq \mathbb{F}_2^n$ un código. Sean $x \in \mathcal{C}$ y $y \in \mathbb{F}_2^n$. Se tiene que $D_{\mathcal{C}}(y) = x$, siempre que se cumpla una de las siguientes condiciones:*

1. $\delta(y,x) < \frac{\delta(\mathcal{C}) + \left(\omega_H(y) - \omega_H(x)\right)(\gamma - 1)}{2}$,
2. $\delta(y,x) < \frac{\hat{\delta}(\mathcal{C}) + \omega_H(y)(\gamma-1)}{2}$.

Aunque para probarlo nos hace falta primero hacer uso de este lema.

Lema 3.24. *Para todos $x, y, z \in \mathbb{F}_2^n$, se cumple la siguiente desigualdad triangular para la discrepancia δ:*

$$\delta(z,x) \leq \delta(z,y) + \delta(y,x).$$

Demostración. Dado que la discrepancia es aditiva en los componentes del vector, basta con demostrar el resultado para $n = 1$. El análisis caso por caso se puede resumir en la siguiente tabla:

a	b	c	$\delta(c,a)$	$\delta(c,b)$	$\delta(b,a)$
0	0	0	0	0	0
1	0	0	1	0	1
0	1	0	0	1	γ
0	0	1	γ	γ	0
1	1	0	1	1	0
1	0	1	0	γ	1
0	1	1	γ	0	γ
1	1	1	0	0	0

□

Nota 3.25. Aunque $\hat{\delta}$ es simétrica, no satisface una desigualdad triangular natural. Más precisamente, en general no se cumple que:

$$\hat{\delta}(z,x) \leq \hat{\delta}(z,y) + \hat{\delta}(y,x).$$

Por ejemplo, supongamos que $\gamma > 1$ y consideremos $x = (0,0,0)$, $y = (1,0,0)$, y $z = (1,1,0)$. Entonces:

$$\hat{\delta}(z,x) = 2 > 3 - \gamma = \hat{\delta}(z,y) + \hat{\delta}(y,x).$$

La desigualdad más cercana a una desigualdad triangular que se puede obtener para la discrepancia simétrica $\hat{\delta}$ es:

$$\hat{\delta}(z,x) \leq \hat{\delta}(z,y) + \hat{\delta}(y,x) + \omega_H(y)(\gamma - 1),$$

la cual se cumple para todos $x, y, z \in \mathbb{F}_2^n$ y contiene $\omega_H(y)(\gamma - 1)$ como un término de corrección.

Procedamos ahora a demostrar la Proposición 3.23.

Demostración. Primero observamos que, para cualesquiera $x, x', y \in \mathbb{F}_2^n$, por la desigualdad triangular y el Lema 3.9 se tiene:

$$\delta(y, x') \geq \delta(x, x') - \delta(y, x) + (\omega_H(y) - \omega_H(x'))(\gamma - 1). \tag{I}$$

Fijemos $x \in \mathcal{C}$, $y \in \mathbb{F}_2^n$ y demostremos ambas afirmaciones por separado:

1. Para todo $x' \in \mathcal{C}$ con $x' \neq x$, la desigualdad en (I) y nuestra hipótesis sobre $\delta(y, x')$ implican:

$$\begin{aligned}
\delta(y, x') &\geq \delta(x, x') - \delta(y, x) + (\omega_H(y) - \omega_H(x'))(\gamma - 1) \\
&\geq \delta(\mathcal{C}) - \frac{\delta(\mathcal{C}) + (\omega_H(y) - \omega_H(x))(\gamma - 1)}{2} + (\omega_H(y) - \omega_H(x))(\gamma - 1) \\
&= \frac{\delta(\mathcal{C}) + (\omega_H(y) - \omega_H(x))(\gamma - 1)}{2} \\
&> \delta(y, x).
\end{aligned}$$

El resultado deseado se sigue del Teorema 3.15 y la definición del decodificador de máxima verosimilitud, pues x es la palabra del código que minimiza la discrepancia.

2. De manera análoga, para todo $x' \in \mathcal{C}$ con $x' \neq x$, la desigualdad en (I) y la hipótesis sobre $\delta(y, x')$ implican:

$$\begin{aligned}
\delta(y, x') &\geq \delta(x, x') - \delta(y, x) + (\omega_H(y) - \omega_H(x))(\gamma - 1) \\
&\geq \hat{\delta}(\mathcal{C}) - \frac{\hat{\delta}(\mathcal{C}) + \omega_H(y)(\gamma - 1)}{2} + \omega_H(y)(\gamma - 1) \\
&= \frac{\hat{\delta}(\mathcal{C}) + \omega_H(y)(\gamma - 1)}{2} \\
&> \delta(y, x).
\end{aligned}$$

Nuevamente, el enunciado se deduce a partir del Teorema 3.15. □

Nota 3.26. Sabemos que en el caso de la distancia de Hamming, si la distancia entre dos vectores sea menor que la mitad de la distancia mínima del código $d_H(y, x) < d_H(\mathcal{C})/2$ es suficiente para asegurar que el vector y se decodifica por la palabra del código x, $D_{\mathcal{C}}^H(y) = x$. Análogamente, podríamos pensar que si la misma condición $\delta(y, x) < \delta(\mathcal{C})/2$ se cumple para la discrepancia, entonces también $D_{\mathcal{C}}^{\delta}(y) = x$, donde $D_{\mathcal{C}}^{\delta}$ es el decodificador de discrepancia mínima. Sin embargo, no es cierto en general como podemos observar en el siguiente ejemplo.

Ejemplo 3.27. Consideremos un código sencillo formado únicamente por dos palabras

$$\mathcal{C} = \{(1, 0, 0), (0, 1, 1)\}$$

, y el canal binario asimétrico con parámetros $p = 0{,}1$ y $q = 0{,}4$. Luego, el valor de $\gamma \approx 2, 21$.

$$\begin{aligned}
\delta((0, 1, 1), (1, 0, 0)) &= \gamma d_{10}((0, 1, 1), (1, 0, 0)) + d_{01}((0, 1, 1), (1, 0, 0)) = 2 \cdot \gamma + 1 \approx 5, 41 \\
\delta((1, 0, 0), (0, 1, 1)) &= \gamma d_{10}((1, 0, 0), (0, 1, 1)) + d_{01}((1, 0, 0), (0, 1, 1)) = 1 \cdot \gamma + 2 \approx 4, 21
\end{aligned}$$

Es decir, se tiene que $\delta(\mathcal{C}) \approx 4{,}21$ y por tanto $\delta(\mathcal{C})/2 \approx 2{,}105$. Ahora consideremos el vector $y = (0, 0, 0)$ y calculemos las discrepancia de y con cada una de las palabras del código.

$$\begin{aligned}
\delta(y, (1, 0, 0)) &= \gamma d_{10}((0, 0, 0), (1, 0, 0)) + d_{01}((0, 0, 0), (1, 0, 0)) = 1 \\
\delta(y, (0, 1, 1)) &= \gamma d_{10}((0, 0, 0), (0, 1, 1)) + d_{01}((0, 0, 0), (0, 1, 1)) = 2
\end{aligned}$$

El conjunto de palabras código cuya discrepancia con y es estrictamente menor que $\delta(\mathcal{C})/2 \approx 2,1$ es todo el código $\mathcal{C}$. Sin embargo, la única palabra código que minimiza la discrepancia con el vector y es $(1,0,0)$. Por tanto, aunque para cualquier palabra $x \in \mathcal{C}$ se cumpla que $\delta(y,x) < \delta(\mathcal{C})/2$, como la única decodificación valida para y es $(1,0,0)$ y la condición no es suficiente en el caso asimétrico. Si comprobamos este resultado con la función *decodificador* programada en *SageMath*.

```
>>> p = 0.1
>>> q = 0.4
>>> C = [[1, 0, 0], [0, 1, 1]]
>>> y = [0,0,0]
>>> s,x = decodificador(C,y,p,q)
>>> if s==1:
    print('La decodificación es única y la palabra del código buscada es:',x)

    La decodificación es única y la palabra del código buscada es: [1,0,0]
```

El lema que se demuestra a continuación, permite establecer una relación entre la distancia de Hamming y la discrepancia no simétrica δ, se incluye aquí por completitud, pero este resultado no se va a necesitar en lo que resta del trabajo.

Lema 3.28. *La primera desigualdad de la Proposición 3.23*

$$\delta(y,x) < \frac{\delta(\mathcal{C}) + \big(\omega_H(y) - \omega_H(x)\big)(\gamma - 1)}{2},$$

es equivalente a la condición

$$d_H(y,x) < \frac{\delta(\mathcal{C})}{\gamma + 1}.$$

Demostración. Teniendo en cuenta que podemos escribir el peso de y como $\omega_H(y) = d_{10}(y,x) + d_{11}(y,x)$ y la definicion de la discrepancia no simétrica:

$$\begin{aligned}
\delta(y,x) &< \frac{\delta(\mathcal{C}) + \big(\omega_H(y) - \omega_H(x)\big)(\gamma - 1)}{2} \\
&< \frac{\delta(\mathcal{C}) + \big(d_{10}(y,x) + d_{11}(y,x) - (d_{01}(y,x) + d_{11}(y,x))\big)(\gamma - 1)}{2} \\
&< \frac{\delta(\mathcal{C}) + \big(d_{10}(y,x) - d_{01}(y,x)\big)(\gamma - 1)}{2} \\
&< \frac{\delta(\mathcal{C}) + \delta(y,x) - d_{10}(y,x) - \gamma d_{01}(y,x)}{2}.
\end{aligned}$$

Operando en esta última desigualdad obtenemos que:

$$\delta(y,x) < \delta(\mathcal{C}) - \big(\gamma d_{01}(y,x) + d_{10}(y,x)\big).$$

Nuevamente utilizando la definición de la discrepancia y teniendo en cuenta que la distancia de Hamming entre dos elementos la podemos escribir como $d_H(y,x) = d_{01}(y,x) + d_{10}(y,x)$.

$$\gamma d_{10}(y,x) + d_{01}(y,x) + \gamma d_{01}(y,x) + d_{10}(y,x) < \delta(\mathcal{C})$$

$$(\gamma + 1)(d_{01}(y,x) + d_{10}(y,x)) < \delta(\mathcal{C})$$

De donde se deduce que ambas desigualdades expuestas en el enunciado son equivalentes.

□

3.3. Cotas superiores para la probabilidad de error

Queremos ver como de efectivo es el decodificador de máxima verosimilitud que hemos definido previamente, es decir cuando es capaz de devolver el estímulo inicial que se envió a través del canal y cuando se produce un error en la decodificación o se envía otro estímulo distinto al original. Vamos a ver que podemos establecer dos cotas superiores para el error, una para cada de las medidas de discrepancia que hemos descrito previamente. Veremos que ambas cotas superiores son incomparables en general y que por lo tanto ambas medidas de discrepancia son independientes e importantes en la descripción de un código binario de un canal asimétrico.

Definición 3.29. Sea $\mathrm{S} := \{a + \gamma b \mid a, b \in \mathbb{N}\}$ el conjunto de *combinaciones lineales de* 1 *y* γ con coeficientes en los números naturales. Para un número natural fijo $h \in \mathbb{N}$, definimos $\mathrm{S}(h) := \{s \in \mathrm{S} \mid 0 \leq s < h\}$, como todos aquellos elementos de S estrictamente menores que h. Además, para $a, b \in \mathbb{R}$, definimos la función

$$\binom{a}{b}_{\mathbb{R}} := \begin{cases} \frac{a!}{b!(a-b)!} & \text{si } a, b \in \mathbb{N}, \\ 0 & \text{en otro caso.} \end{cases}$$

En la Figura 3.8 se puede observar como se calcularían tanto los elementos del conjunto S, como este calculo particular de los coeficientes binomiales.

Lema 3.30. *Sean* $x, x' \in \mathbb{F}_2^n$ *dos vectores con el mismo peso de Hamming,* $\omega_H(x) = \omega_H(x')$*. Sea* $i \in \{1, \ldots, n\}$ *y* $s \in \mathrm{S}$*. Entonces, tenemos la igualdad entre estos dos conjuntos*

$$\{y \in \mathbb{F}_2^n \mid \omega_H(y) = i,\ \delta(y, x) = s\} = \{y \in \mathbb{F}_2^n \mid \omega_H(y) = i,\ \delta(y, x') = s\}.$$

Además, si j *denota el peso de Hamming de* x*, entonces*

$$|\{y \in \mathbb{F}_2^n \mid \omega_H(y) = i,\ \delta(y, x) = s\}| = \binom{j}{\frac{i\gamma - s + j}{\gamma + 1}} \binom{n - j}{\frac{s - j + i}{\gamma + 1}}.$$

Demostración. Sea $x \in \mathbb{F}_2^n$ un vector y sea $j = \omega_H(x)$. Como la discrepancia se expresa como $\delta(y, x) = \gamma d_{10}(y, x) + d_{01}(y, x)$, denotando por $b = d_{10}(y, x)$, $a = d_{11}(y, x)$ y por tanto $d_{01} = j - a$; el número de vectores $y \in \mathbb{F}_2^n$ que cumplen $\omega_H(y) = i$ y $\delta(y, x) = s$ está dado por la suma en a y b del producto de las diferentes formas de elegir a posiciones de entre las j que son distintas de cero y elegir b entre las $n - j$ que son ceros por tanto.

$$\sum_{\substack{0 \leq a \leq j \\ 0 \leq b \leq n - j \\ a + b = i \\ j - a + \gamma b = s}} \binom{j}{a} \binom{n - j}{b}.$$

Este número no depende de x, sino solamente de su peso $\omega_H(x) = j$. Además, los valores de a y b quedan determinados únicamente por las restricciones impuestas en la suma, lo que reduce el problema a una combinación de coeficientes binomiales tal como se indica en el enunciado del lema. □

Nota 3.31. En lo que sigue, para $i, j \in \mathbb{N}$ y $s \in \mathbb{R}$, denotaremos por $\lambda(i, j, s)$ al número de vectores $y \in \mathbb{F}_2^n$ de peso i y tales que la discrepancia no simétrica con x tiene el valor s, lo que se denota como $\delta(y, x) = s$, donde $x \in \mathbb{F}_2^n$ es un vector cualquiera de peso j. Por el lema anterior, esta cantidad $\lambda(i, j, s)$ está bien definida y viene dada por

$$\lambda(i, j, s) = \binom{j}{\frac{i\gamma - s + i}{\gamma + 1}} \binom{n - j}{\frac{s - i + i}{\gamma + 1}}.$$

```
def conjunto_S(h,p,q):
    g = gamma(p,q)
    n = ceil(h) + 1
    S = []

    for a in range(n):
        for b in range(n):
            if a+g*b<h:
                S.append(a+g*b)
    return S

def binomialreal(n,m):
    if (N(n,13) in NonNegativeIntegers()) and (N(m,13) in NonNegativeIntegers()):
        s = binomial(round(n),round(m))

    else :
        s = 0
    return s
```

```
>>> p = 0.1
>>> q = 0.4
>>> print(gamma(p,q))
>>> conjunto_S(3.4, p,q)

    2.209511291351455
    [0.0, 2.209511291351455, 1.0, 3.209511291351455, 2.0, 3.0]


>>> binomialreal(5.0000000000000002, 1)

    5


>>> binomialreal(8, 0.3)

    0
```

Figura 3.8: La función *conjunto S* calcula los elementos del conjunto S(h) para un número $h \in \mathbb{R}$ arbitrario, y depende del valor de γ o lo que es lo mismo de las probabilidades de transición p y q. En el primero ejemplo, se presenta el conjunto de números que pertenecen a S(3,4) cuando las probabilidades de transmisión del canal son 0,1 y 0,4, o equivalentemente que $\gamma \approx 2{,}21$.
La función *binomial natural* es muy similar al cálculo de un número combinatorio, y de hecho, se utiliza dentro de ella la función *binomial* que viene ya implementada en *SageMath* y que calcula los coeficientes binomiales cuando sus coeficientes pertenecen a los números naturales, o aunque no pertenezcan a estos, estén muy próximos a alguno de sus elementos, en este caso, se calcula el coeficiente binomial con el número natural al que se aproximan. Esta adaptación va a ser necesaria debido a la precisión limitada de los cálculos númericos, ya que se guardan un número limitado de cifras decimales. En el segundo ejemplo escrito en la consola, podemos comprobar como funciona. Además, se contempla el caso en el que alguno de ellos no pertenezca a dicho conjunto, definiendo allí el coeficiente binomial como cero, esto se puede observar en el segundo ejemplo donde uno de los coeficientes no es un número natural.

Definición 3.32. Sea $\mathcal{C} \subseteq \mathbb{F}_2^n$ un código y $x \in \mathcal{C}$. La *probabilidad error en la decodificación de la palabra* x es decir de que el decodificador de máxima verosimilitud $D_{\mathcal{C}}$ devuelva un vector distinto al enviado $x' \neq x$ o un mensaje de error e está dada por

$$\sum_{y \in \mathbb{F}_2^n \setminus D_{\mathcal{C}}^{-1}(x)} P(y \mid x),$$

donde recordamos que $P(y \mid x)$ representa la probabilidad de que se reciba $y \in \mathbb{F}_2^n$, habiéndose enviado $x \in \mathcal{C}$ a través del canal.

Si suponemos que todos las palabras del código se transmiten con la misma probabilidad, como es habitual, podemos definir el siguiente concepto:

Definición 3.33. La *probabilidad de decodificación no exitosa* para un código $\mathcal{C} \subseteq \mathbb{F}_2^n$ es el promedio

$$\mathrm{PUD}(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{x \in \mathcal{C}} \sum_{y \in \mathbb{F}_2^n \setminus D_{\mathcal{C}}^{-1}(x)} P(y \mid x).$$

Las siglas PUD hacen referencia a su nombre en inglés "probability of unsuccesful decoding".

Nota 3.34. También existe la probabilidad de decodificación incorrecta que hace referencia a la probabilidad de se devuelva otra palabra distinta a la enviada pero no incluye la probabilidad de que se devuelva un mensaje de error al decodificar. En otras palabras, es la probabilidad de que la decodificación sea errónea y además, no se pueda detectar que ha ocurrido un error.

Definición 3.35. Para un código $\mathcal{C} \subseteq \mathbb{F}_2^n$ y un entero $i \in \{0, \ldots, n\}$, denotamos por $W_i^H(\mathcal{C})$ al número de palabras del código $x \in \mathcal{C}$ con peso de Hamming exactamente i, $\omega_H(x) = i$. El vector $(W_0^H(\mathcal{C}), \ldots, W_n^H(\mathcal{C}))$ es la *distribución de pesos* de $\mathcal{C}$.

Esto es exactamente igual que en el caso simétrico, donde escribíamos también el polinomio de pesos de la siguiente forma [11]

$$W^H(X) = \sum_{i=0}^{n} W_n^H(\mathcal{C}) X^n.$$

Teorema 3.36. *Sea $\mathcal{C} \subseteq \mathbb{F}_2^n$ un código. Entonces tenemos:*

$$PUD(\mathcal{C}) \leq 1 - \frac{1}{|\mathcal{C}|} \sum_{j=0}^{n} W_j^H(\mathcal{C}) \sum_{i=0}^{n} (1-q)^i (1-p)^{n-i} \sum_{s \in S\left(\frac{\delta(\mathcal{C})+(\gamma-1)(i-j)}{2}\right)} \left(\frac{q}{1-p}\right)^s \lambda(i,j,s),$$

y

$$PUD(\mathcal{C}) \leq 1 - \frac{1}{|\mathcal{C}|} \sum_{j=0}^{n} W_j^H(\mathcal{C}) \sum_{i=0}^{n} (1-q)^i (1-p)^{n-i} \sum_{s \in S\left(\frac{\delta(\mathcal{C})+(\gamma-1)i}{2}\right)} \left(\frac{q}{1-p}\right)^s \lambda(i,j,s).$$

Demostración. Comenzamos mostrando la primera cota de la proposición. Por el Lema 3.4 y las definiciones de γ y δ, tenemos:

$$\begin{aligned}
\mathrm{PUD}(\mathcal{C}) &= 1 - \frac{1}{|\mathcal{C}|} \sum_{x \in \mathcal{C}} \sum_{y \in \mathbb{F}_2^n \setminus D_{\mathcal{C}}^{-1}(x)} P^n(y \mid x) \\
&= 1 - \frac{1}{|\mathcal{C}|} \sum_{x \in \mathcal{C}} \sum_{y \in \mathbb{F}_2^n \setminus D_{\mathcal{C}}^{-1}(x)} \left(\frac{q}{1-p}\right)^{d_{01}(y,x)} \left(\frac{p}{1-q}\right)^{d_{10}(y,x)} (1-q)^{\omega_H(y)} (1-p)^{n-\omega_H(y)} \\
&= 1 - \frac{1}{|\mathcal{C}|} \sum_{x \in \mathcal{C}} \sum_{y \in \mathbb{F}_2^n \setminus D_{\mathcal{C}}^{-1}(x)} \left(\frac{q}{1-p}\right)^{\delta(y,x)} (1-q)^{\omega_H(y)} (1-p)^{n-\omega_H(y)}.
\end{aligned}$$

```
def PUD(C,p,q):

    n = len(C[0])
    elementos = espacio_vectorial(2,n)
    P = 0

    for c in C:

        for y in elementos:

            s, d = decodificador(C,y,p,q)

            if s==1:

                if comparar_listas(d,c) == 0:
                    P = P + prob_condicionada(y,c,p,q)

            else:
                P = P + prob_condicionada(y,c,p,q)

    P = P* (1/len(C))

    return P
```

```
>>> codigo= [[0, 0, 0],[0, 1, 1],[1, 1, 1]]
>>> PUD(codigo,0.1,0.3)

    0.244666666666667


>>> codigo= [[0, 0, 1],[0, 1, 0],[1, 1, 1]]
>>> PUD(codigo,0.1,0.3)

    0.339666666666667
```

Figura 3.9: La función *PUD* calcula la probabilidad de decodificación no exitosa o errónea para un código binario que se envía utilizando un canal binario asimétrico de parámetros p y q, como se ha comentado anteriormente esta probabilidad es la de que llegue a través del canal un vector que decodificaría a otra palabra del código diferente de la original que fue enviada.
Además en los ejemplos que se presentan, en el primero de ellos, todos los elementos del espacio vectorial $\mathbb{F}_2^3$ se pueden decodificar por el elemento más cercano en términos de discrepancia no simétrica. Mientras que en el segundo ejemplo el elemento $(0,0,0) \in \mathbb{F}_2^3$ tiene la misma discrepancia no simétrica con las dos primeras palabras del código y por lo tanto cualquier decodificación si se recibe este vector a través del canal producirá un error, y por tanto esto hará aumentar la probabilidad de decodificación no exitosa en el segundo ejemplo.

Donde teniendo en cuenta que $\gamma = log_{\frac{q}{1-p}}(\frac{p}{1-q})$ y que $\delta = \gamma d_{10} + d_{01}$, podemos reescribir este producto como simplemente un factor, obteniéndose así la última igualdad anterior.

$$\begin{aligned}\left(\frac{q}{1-p}\right)^{d_{01}(y,x)}\left(\frac{p}{1-q}\right)^{d_{10}(y,x)} &= \left(\frac{q}{1-p}\right)^{\delta-\gamma d_{10}(y,x)}\left(\frac{p}{1-q}\right)^{d_{10}(y,x)} \\ &= \left(\frac{q}{1-p}\right)^{\delta(y,x)}\left(\frac{q}{1-p}\right)^{-log_{\frac{q}{1-p}}(\frac{p}{1-q})d_{10}(y,x)}\left(\frac{p}{1-q}\right)^{d_{10}(y,x)} \\ &= \left(\frac{q}{1-p}\right)^{\delta(y,x)}.\end{aligned}$$

Por la Proposición 3.23, se tiene que el conjunto de puntos cuya discrepancia no simétrica con x verifica la primera desigualdad de la proposición, al que vamos a denotar A, tiene que estar contenido en el conjunto de puntos cuya decodificación es el vector x:

$$A := \{y \in \mathbb{F}_2^n \mid \delta(y,x) < \frac{\delta(\mathcal{C}) + (\gamma-1)(\omega_H(y) - \omega_H(x))}{2}\} \subseteq \{y \in \mathbb{F}_2^n \mid D_{\mathcal{C}}(y) = x\}, \text{ para todo } x \in \mathcal{C}.$$

Utilizando esto podemos acotar el valor de la probabilidad de decodificación errónea por:

$$\begin{aligned}\mathrm{PUD}(\mathcal{C}) &\leq 1 - \frac{1}{|\mathcal{C}|}\sum_{\substack{x\in\mathcal{C}\\ \omega_H(x)=j}}\sum_{y\in A}\left(\frac{q}{1-p}\right)^{\delta(y,x)}(1-q)^{\omega_H(y)}(1-p)^{n-\omega_H(y)} \\ &= 1 - \frac{1}{|\mathcal{C}|}\sum_{j=0}^{n}\sum_{\substack{x\in\mathcal{C}\\ \omega_H(x)=j}}\sum_{i=0}^{n}\sum_{s\in \mathrm{S}(\frac{\delta(\mathcal{C})+(\gamma-1)(i-j)}{2})}\sum_{\substack{y\in\mathbb{F}_2^n\\ D_{\mathcal{C}}(y)=x\\ \delta(y,x)=s}}\left(\frac{q}{1-p}\right)^{s}(1-q)^{i}(1-p)^{n-i}.\end{aligned}$$

Donde en la última igualdad se ha utilizado que δ es por definición de la forma $a\gamma + b$ con $a, b \in \mathbb{N}$ y como estamos sumando dentro del conjunto A se tiene que $\delta \in \mathrm{S}(\frac{\delta(\mathcal{C})+(\gamma-1)(\omega_H(y)-\omega_H(x))}{2})$. Finalmente, usando la distribución de pesos del código $\mathcal{C}$ y la notación que hemos introducido en esta sección, llegamos a la primera cota enunciada por el teorema:

$$\mathrm{PUD}(\mathcal{C}) \leq 1 - \frac{1}{|\mathcal{C}|}\sum_{j=0}^{n}W_j^H(\mathcal{C})\sum_{i=0}^{n}(1-q)^{i}(1-p)^{n-i}\sum_{s\in\mathrm{S}\left(\frac{\delta(\mathcal{C})+(\gamma-1)(i-j)}{2}\right)}\left(\frac{q}{1-p}\right)^{s}\lambda(i,j,s).$$

La demostración de la segunda cota es análoga, usando el hecho de que la segunda desigualdad de la Proposición 3.23 también implica que:

$$B := \{y \in \mathbb{F}_2^n \mid \delta(y,x) < \frac{\hat{\delta}(C) + \omega_H(y)(\gamma-1)}{2}\} \subseteq \{y \in \mathbb{F}_2^n \mid D_{\mathcal{C}}(y) = x)\}, \text{ para todo } x \in \mathcal{C}.$$

De manera similar, podemos acotar el valor de la probabilidad de decodificación errónea por:

$$\begin{aligned}\mathrm{PUD}(\mathcal{C}) &\leq 1 - \frac{1}{|\mathcal{C}|}\sum_{\substack{x\in\mathcal{C}\\ \omega_H(x)=j}}\sum_{y\in A}\left(\frac{q}{1-p}\right)^{\delta(y,x)}(1-q)^{\omega_H(y)}(1-p)^{n-\omega_H(y)} \\ &= 1 - \frac{1}{|\mathcal{C}|}\sum_{j=0}^{n}\sum_{\substack{x\in\mathcal{C}\\ \omega_H(x)=j}}\sum_{i=0}^{n}\sum_{s\in \mathrm{S}(\frac{\delta(\mathcal{C})+(\gamma-1)i}{2})}\sum_{\substack{y\in\mathbb{F}_2^n\\ D_{\mathcal{C}}(y)=x\\ \delta(y,x)=s}}\left(\frac{q}{1-p}\right)^{s}(1-q)^{i}(1-p)^{n-i}.\end{aligned}$$

Igual que antes en la última igualdad se ha utilizado que δ sabemos que por definición es de la forma $a\gamma + b$ con $a, b \in \mathbb{N}$, si queremos sumar para todos los valores de δ dentro del conjunto B se tiene que $\delta \in \mathrm{S}(\frac{\hat{\delta}(\mathcal{C})+(\gamma-1)\omega_H(y)}{2})$.
Finalmente, usando como antes la distribución de pesos del código $\mathcal{C}$ y la notación que hemos introducido en esta sección, llegamos a la segunda cota enunciada por el teorema:

$$\mathrm{PUD}(\mathcal{C}) \leq 1 - \frac{1}{|\mathcal{C}|}\sum_{j=0}^{n} W_j^H(\mathcal{C}) \sum_{i=0}^{n} (1-q)^i (1-p)^{n-i} \sum_{s \in \mathrm{S}\left(\frac{\hat{\delta}(\mathcal{C})+(\gamma-1)i}{2}\right)} \left(\frac{q}{1-p}\right)^s \lambda(i,j,s).$$

□

Ejemplo 3.37. Vamos a utilizar las funciones programadas en *SageMath* para ambas cotas, Figura 3.10, en este ejemplo, en el que vamos a ver como haciendo variar uno de los parámetros del canal binario asimétrico, en este caso variaremos q, como varían la probabilidad de decodificación no exitosa y las dos cotas que hemos deducido para ella. Como resultado en las gráficas veremos que ambas cotas no son comparables, es decir, que una no es una mejor que la otra, sino que depende de cada caso concreto, y que por tanto ambas cotas son independientes y tienen relevancia.

Para el código formado por tres palabras

$$\mathcal{C} = \{(0,0,0,0,0), (1,1,0,0,0), (1,1,1,1,1)\}$$

considerando el parámetro p fijo, con valor $p = 0{,}05$ y haciendo variar el otro parámetro q entre 0,1 y 0,5 podemos ver como varían las cotas y el valor de la probabilidad de decodificación no exitosa. La gráfica se puede ver en la Figura 3.11.

Para el código formado por las siguientes palabras

$$\mathcal{C} = \{(0,0,0,0), (0,0,1,0), (0,1,0,0), (1,0,0,0), (0,0,1,1), (1,1,1,1)\}$$

considerando el parámetro p fijo con valor $p = 0{,}05$ y haciendo variar el parámetro q entre 0,1 y 0,5 podemos ver como varían también las cotas y el valor de la probabilidad de decodificación no exitosa. La gráfica se puede observar en la Figura 3.12.

Finalmente, en este ejemplo se pretende ilustrar los cálculos que hemos realizado con las funciones programadas en *SageMath* sobre todo nos interesa, la función *Conjunto S*, ya que la definición de este conjunto puede resultar algo confusa.

Ejemplo 3.38. Sea $\mathcal{C} = \{(0,0,0), (0,1,1), (1,1,1)\} = \{x_1, x_2, x_3\} \in \mathbb{F}_2^3$ el código de longitud $n = 3$ definido sobre el cuerpo $\mathbb{F}_2$. Además, consideramos los siguientes parámetros para el canal binario asimétrico $p = 0,1$ y $q = 0,3$. Si quisiéramos calcular la primera cota para la probabilidad de decodificación no exitosa, necesitamos calcular los conjuntos S, para cada i y para cada j para luego poder sumar en ellos. Lo primero, el valor de gamma asociado al canal sera aproximadamente $\gamma \approx 1,77$ y podemos comprobarlo:

```
>>> C =[[0,0,0], [0,1,1], [1,1,1]
>>> gamma(0.1,0.3)

1.7712437491614221
```

```
def cota_1(C,p,q):
    n = len(C[0])
    suma = 0
    g = gamma(p,q)

    for j in range(n+1):
        w_j = polinomio_pesos(C)[j]

        for i in range(n+1):
            suma_i = ((1 - q)**i) * ((1 - p)**(n - i))
            d = discrepancia_minima(C,p,q)
            S = conjunto_S((d + (i-j) * (g - 1)) / 2, p, q)

            if len(S)!=0:

                for s in S:
                    b1 = binomialreal(j, (i*g - s + j) / (g + 1))
                    b2 = binomialreal(n - j, (s - j + i) / (g + 1))
                    suma = suma + w_j * suma_i *((q / (1 - p))**s) * b1 * b2

    suma =  1 - (1 / len(C))*suma
    return suma

def cota_2(C,p,q):
    n = len(C[0])
    suma = 0
    g = gamma(p,q)

    for j in range(n+1):
        w_j = polinomio_pesos(C)[j]

        for i in range(n+1):
            suma_i = ((1 - q)**i) * ((1 - p)**(n - i))
            d = discrepancia_minima_simetrica(C, p, q)
            S = conjunto_S((d + i* (g - 1)) / 2, p, q)

            if len(S)!=0:
                for s in S:
                    b1 = binomialreal(j, (i*g - s + j) / (g + 1))
                    b2 = binomialreal(n - j, (s - j + i) / (g + 1))
                    suma = suma + w_j * suma_i * ((q / (1 - p))**s) * b1 * b2

    suma =  1 - (1 / len(C))*suma
    return suma
```

Figura 3.10: Las funciones *cota 1* y *cota 2* calculan las cotas para la probabilidad de decodificación no exitosa que proporciona el Teorema 3.36. Un aspecto importante es que el conjunto S del último sumatorio puede ser vacío, para algún valor de i y j, ese término entonces no aporta nada a la suma final.

```
>>> q = srange(0.05, 0.1, 0.55)
>>> C = [[0, 0, 0, 0, 0], [1, 1, 0, 0, 0], [1, 1, 1, 1, 1]]
>>> for i in range(len(q)):
        pud.append(PUD(C,0.05, q[i]))
        cota1.append(cota_1(C,0.05, q[i]))
        cota2.append(cota_2(C,0.05, q[i]))
```

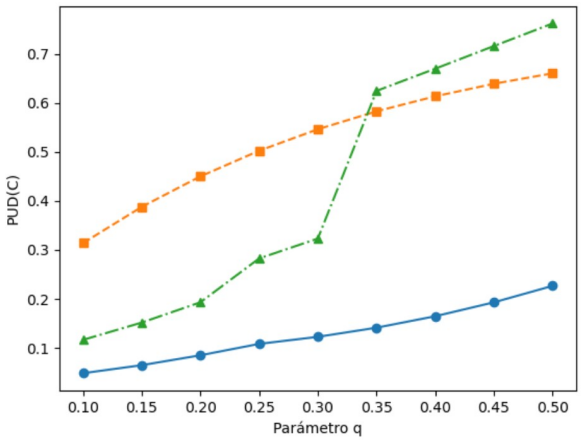

Figura 3.11: En azul y puntos representados por círculos se pueden observar los valores aproximados que toma la probabilidad de decodificación no exitosa.
En naranja y puntos representados por cuadrados se pueden observar los valores que toma la primera cota para la probabilidad de decodificación no exitosa.
En verde y con puntos representados por triángulos se pueden observar los valores que toma la segunda cota para la probabilidad de decodificación no exitosa.

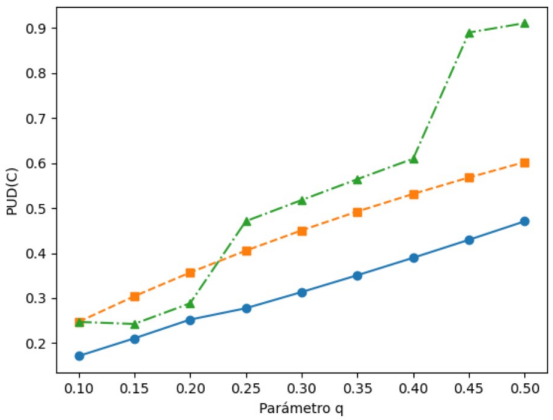

Figura 3.12: En azul y puntos representados por círculos se pueden observar los valores aproximados que toma la probabilidad de decodificación no exitosa.
En naranja y puntos representados por cuadrados se pueden observar los valores que toma la primera cota para la probabilidad de decodificación no exitosa.
En verde y con puntos representados por triángulos se pueden observar los valores que toma la segunda cota para la probabilidad de decodificación no exitosa.

Además, es necesario calcular la discrepancia mínima del código es igual a 1, $\delta(\mathcal{C}) = 1$.

$$\begin{aligned} \delta(x_1, x_2) &= 2 & \delta(x_2, x_1) &= 2 \cdot \gamma \\ \delta(x_1, x_3) &= 3 & \delta(x_3, x_1) &= 3 \cdot \gamma \\ \delta(x_2, x_3) &= 1 & \delta(x_3, x_2) &= 1 \cdot \gamma \end{aligned}$$

```
>>> discrepancia_minima(C, 0.1,0.3)

1
```

Calculemos ahora los conjuntos S necesarios para calcular la primera cota para la probabilidad de decodificación no exitosa. Como su valor depende de los valores que vayan adquiriendo i y j, vamos a hacer una tabla, como i, j varían entre 0 y n, en este ejemplo pertenecerán al conjunto $i, j \in \{0, 1, 2, 3\}$. Recordemos que el conjunto es $\mathrm{S}(h) = \{a\gamma + b < h \mid a, b \in \mathbb{N}\}$ y los conjuntos que intervienen en el cálculo son de la forma:

$$\mathrm{S}\left(\frac{\delta(\mathcal{C}) + (\gamma - 1)(i - j)}{2}\right).$$

$j \backslash i$	$i = 0$	$i = 1$	$i = 2$	$i = 3$
$j = 0$	$\mathrm{S}(\frac{1}{2}) = \{0\}$	$\mathrm{S}(\frac{\gamma}{2}) = \{0\}$	$\mathrm{S}(\frac{2\gamma-1}{2}) = \{0, 1\}$	$\mathrm{S}(\frac{3\gamma-2}{2}) = \{0, 1\}$
$j = 1$	$\mathrm{S}(\frac{2-\gamma}{2}) = \{0\}$	$\mathrm{S}(\frac{1}{2}) = \{0\}$	$\mathrm{S}(\frac{\gamma}{2}) = \{0\}$	$\mathrm{S}(\frac{2\gamma-1}{2}) = \{0, 1\}$
$j = 2$	$\mathrm{S}(\frac{3-2\gamma}{2}) = \{\emptyset\}$	$\mathrm{S}(\frac{2-\gamma}{2}) = \{0\}$	$\mathrm{S}(\frac{1}{2}) = \{0\}$	$\mathrm{S}(\frac{\gamma}{2}) = \{0\}$
$j = 3$	$\mathrm{S}(\frac{4-3\gamma}{2}) = \{\emptyset\}$	$\mathrm{S}(\frac{3-2\gamma}{2}) = \{\emptyset\}$	$\mathrm{S}(\frac{2-\gamma}{2}) = \{0\}$	$\mathrm{S}(\frac{1}{2}) = \{0\}$

Se puede observar entonces que el conjunto S puede ser vacío, pues por ejemplo en el caso de $i = 0$ y $j = 2$, se tiene que h es negativo $\frac{3-2\gamma}{2} \approx -0{,}27$ y por tanto el conjunto S(-0,27) que por definición está formado únicamente por números no negativos, no puede contener ningún elemento. Esto implica sumar en el conjunto vacío, lo que corresponde con que ese término en el sumatorio quede multiplicado por cero.

Capítulo 4

Conclusión

Toda esta teoría de representación de la respuesta neural a los estímulos desarrollada en la referencia central de este trabajo [4] en el año 2013 se utiliza en la actualidad para diversos trabajos de índole biológica que siguen buscando modelizar y entender el funcionamiento de nuestro cerebro un poco mejor. En este mismo artículo, se estudia además como introduciendo una pequeña tolerancia al error, los códigos del campo receptivo alcanzan el desempeño de los códigos aleatorios de comparación, Sección 2.3, y tienen la ventaja de que representan mejor las relaciones entre los estímulos. También, existe otra perspectiva de índole algebraica sobre los códigos neurales, que se puede observar en el artículo *The neural ring: an algebraic tool for analyzing the intrinsic structure of neural codes* [5], publicado a la par en el año 2013, y que comparte algunos autores con nuestra referencia principal. Este artículo también ha sido de ayuda para comprender el funcionamiento del modelo que aquí se explica.

Un ejemplo de la investigación que surge a raíz de este modelo, lo podemos encontrar en el artículo siguiente publicado un par de años después en 2015, *A Thesaurus for a Neural Population Code* [7], dónde se investiga como se puede organizar un código neural en diferentes clases de equivalencia, donde el contenido de cada una de ellas son todas las representaciones equivalentes de un mismo estímulo, es decir que ante el mismo estímulo las respuestas que da una población de neuronas a él aunque no sean iguales, suceso que ha sido observado de forma experimental, tienen que pertenecer a la misma clase y que dos respuestas de la misma clase por tanto han de ser intercambiables entre sí.

Otros ejemplos más recientes del uso de esta teoría de códigos neurales en la investigación biológica sobre el cerebro los proporcionan estos dos artículos. En el artículo *Optimal Combinatorial Neural Codes With Matched Metric δ_r: Characterization and Constructions* [15] publicado en abril de 2023 se demuestra que los códigos neurales alcanzan las cotas superiores para la discrepancia asimétrica si el código alcanza las correspondientes cotas superiores para la usual distancia de Hamming y por tanto los códigos neurales óptimos provienen de los códigos óptimos usuales. En este otro artículo también bastante reciente *Optimal combinatorial neural codes via symmetric designs* [16] publicado en el verano 2024, se estudian los códigos neurales que se transmiten usando el canal asimétrico que son óptimos en el sentido de que igualan una cota propuesta que mejora la clásica de Plotkin [10], lo cual sería imposible en códigos que se transmiten a través del canal simétrico clásico.

Finalmente, en este otro artículo *A Simple Self-Decoding Model for Neural Coding* [1] publicado también en agosto de 2024, se presenta un modelo novedoso para la decodificación de los códigos neurales, lo que equivale a recuperar a parir del patrón neural respuesta el estímulo que lo genera,

en él se utiliza la representación de los estímulos en forma de código neural ya que permite reflejar la geometría del espacio de estímulos considerado.

Bibliografía

[1] Thach V. Bui. A simple self-decoding model for neural coding. In *2024 IEEE International Symposium on Information Theory (ISIT)*, pages 268–273, 2024.

[2] Giuseppe Cotardo and Alberto Ravagnani. Parameters of codes for the binary asymmetric channel. *IEEE Trans. Inf. Theory*, 68(5):2941–2950, 2022.

[3] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing. John Wiley & Sons, Hoboken, New Jersey, 2nd edition, 2006.

[4] Carina Curto, Vladimir Itskov, Katherine Morrison, Zachary Roth, and Judy L. Walker. Combinatorial neural codes from a mathematical coding theory perspective. *Neural Computation*, 25(7):1891–1925, 2013.

[5] Carina Curto, Vladimir Itskov, Alan Veliz-Cuba, and Nora Youngs. The neural ring: an algebraic tool for analyzing the intrinsic structure of neural codes. *Bulletin of mathematical biology*, 75:1571–1611, 2013.

[6] Félix Galindo, Javier Sanz, and Luis A. Tristán. *Guía práctica de cálculo infinitesimal en varias variables*. Ediciones Paraninfo, S.A., Madrid, 2005.

[7] Elad Ganmor, Ronen Segev, and Elad Schneidman. A thesaurus for a neural population code. *Elife*, 4:e06134, 2015.

[8] Fernando Izaurieta and Carlos Saavedra. Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*, 2000.

[9] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[10] Carlos Munuera and Juan G. Tena. *Codificación de la información*, volume 25 of *Manuales y textos universitarios, Ciencias*. Secretariado de Publicaciones e Intercambio Científico, Universidad de Valladolid, Valladolid, 1997.

[11] Ruud Pellikaan, Xin-Wen Wu, Stanislav Bulygin, and Relinde Jurrius. *Codes, Cryptology and Curves with Computer Algebra*. Cambridge University Press, 2017.

[12] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[13] The Sage Developers. *SageMath, the Sage Mathematics Software System*, 2025. DOI 10.5281/zenodo.8042260.

[14] Dominic Welsh. *Codes and Cryptography*. Oxford University Press, Oxford, UK, 1988.

[15] Aixian Zhang, Xiaoyan Jing, and Keqin Feng. Optimal combinatorial neural codes with matched metric δ r: Characterization and constructions. *IEEE Transactions on Information Theory*, 69(8):5440–5448, 2023.

[16] Xingyu Zheng, Shukai Wang, and Cuiling Fan. Optimal combinatorial neural codes via symmetric designs. *Designs, Codes and Cryptography*, pages 1–12, 2024.

Apéndice A

Código en SageMath para la implementación del canal binario asimétrico

Todas las simulaciones y gráficos que se incluyen en este trabajo han sido realizadas usando el sistema software matemático de *SageMath* [13] y son implementaciones propias realizadas específicamente para ilustrar este trabajo. En este apéndice, se recogen las funciones principales que se han ido mostrando según se definían en el desarrollo del trabajo con mayor detalle, además, de todas las funciones auxiliares que han sido necesarias para programar las principales. Para facilitar la consulta se incluyen también aquí, ejemplos de uso de todas las funciones, aunque algunos ya hayan sido expuestos con anterioridad en el trabajo.

```
# Configuración inicial

import numpy as np
import matplotlib.pyplot as plt
import math as ma
from itertools import product

F = GF(2)                          # Es el cuerpo finito de dos elementos
```

La función *transmit unsafe* simula la transmisión de una palabra del código a través de un canal binario asimétrico con probabilidades $0 < p \leq q < 1/2$. Para el correcto funcionamiento de esta función es necesario que el vector se defina con coeficientes en el cuerpo de dos elementos F.

```
def transmit_unsafe(msg,p,q):

    for i in range(len(msg)):

        # La función random proporciona un número aleatorio entre 0 y 1
        if msg[i]==0:
            if random() <= p:
                    msg[i] = msg[i] + 1
```

```
        else :
            if random() <= q:
                    msg[i] = msg[i] + 1
    return msg
```

```
>>> msg = vector(F,[0,1,1,0])
>>> transmit_unsafe(msg,0.1,0.4)
    (0, 1, 1, 1)
>>> transmit_unsafe(msg,0.1,0.4)
    (0, 0, 1, 0)
```

La función *gamma* calcula el parámetro γ en función de las probabilidades de transmisión del canal binario asimétrico p y q.

```
def gamma(p, q):
    g = ma.log(p / (1-q), q / (1 - p))
    return g
```

```
>>> gamma(0.2,0.4)

    1.584962500721156
```

```
>>> gamma(0.1,0.3)

    1.7712437491614221
```

La función *hamming weight* calcula el peso de Hamming e un vector de longitud arbitraria pero con coeficientes en $\mathbb{F}_2$, porque asume que las entradas no nulas son unos. La suma en $\mathbb{Z}$ de las entradas de un vector con coeficiente en $\mathbb{F}_2$ es justamente su peso de Hamming, y esto es justo lo que se utiliza para calcularlo. Esta función será necesaria para calcular la discrepancia simétrica, la probabilidad condicionada de recibir un vector habiéndose enviado una palabra del código concreta y para calcular el polinomio de pesos de un código.

```
def hamming_weight(x):
    s=0
    # Sumamos todas las componentes del vector (son o ceros o unos)
    for i in range(len(x)):
        s = s + x[i]

    return s
```

```
>>> x = [0,0,1,1,0,1,0]
>>> hamming_weight(x)

    3
```

```
>>> y = [0,1,0,0,1,1,1]
>>> hamming_weight(y)

    4
```

La función *polinomio de pesos* devuelve un vector con los coeficientes del polinomio de pesos de un código, donde el coeficiente de x^i o el elemento del vector que esta en la posición i representa el número de palabras del código que tienen peso exactamente i.

```
def polinomio_pesos(C):
    s = [0]*(len(C[0])+1)

    for c in C:                    # Si el peso de una palabra del código es i,
                                   # sumamos 1 a la componente i-ésima del vector
        i = hamming_weight2(c)
        s[i] = s[i] + 1

    return s
```

```
>>> C = [[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 1, 1]]
>>> polinomio_pesos(C)

    [1, 1, 1, 1]
```

```
>>> C = [[0, 0, 0, 1], [0, 1, 0, 0], [1, 1, 0, 0], [1, 1, 1, 0]]
>>> polinomio_pesos(C)

    [0, 2, 1, 1, 0]
```

La función *probabilidad 2* es una función equivalente, en el sentido de que sus extremos se alcanzan en los mismos puntos a la probabilidad condicionada de sabiendo que se ha recibido el vector binario y se haya enviado a través del canal la palabra del código x. Esta definida para dos vectores de longitud arbitraria pero igual.

```
def probabilidad2(x,y,p,q):

    # Comprobación de que ambos vectores deben tener la misma longitud
    if len(y) != len(x):
        raise ValueError("Los vectores y y x deben tener la misma longitud.")

    # Cálculo del número de posiciones que son unos a la vez en ambos vectores
    d11 = sum(1 for i in range(len(y)) if y[i] == 1 and x[i] == 1)

    # Cálculo de los logaritmos involucrados en la función
    log1 = ma.log(((1-p)*(1-q))/(p*q))
    log2 = ma.log((1-p)/q)

    # Cálculo del peso de Hamming de la palabra del código
    peso = peso_hamming(x)

    return  d11*log1 - peso*log2
```

```
>>> y = [1, 0, 0, 1]
>>> x = [0, 1, 0, 0]
>>> probabilidad2(x , y, 0.05, 0.2)

    -1.5581446180465497
```

```
>>> y = [0, 0, 0, 1]
>>> x = [0, 1, 1, 1]
>>> probabilidad2(y, x, 0.05, 0.2)

    -0.34370051385331823
```

La función siguiente es *decodificadorML*, está calcula, dado un código binario de bloque de longitud n arbitraria y un vector de la misma longitud que las palabras del código con coeficientes también en el cuerpo de dos elementos $\mathbb{F}_2$ y que puede o no pertenecer al código, la única palabra del código que maximiza la probabilidad condicionada de habiendo recibido ese vector y que haya sido enviada la palabra del código x. En el caso de que no exista un única palabra el parámetro s toma el valor cero, indicándonos que se ha producido un error, precisamente porque varias palabras del código proporcionado maximizan esta probabilidad con el vector recibido y.

```
def decodificadorML(C,y,p,q):
    s=1

    # Cálculo de la probabilidad condicionada máxima
    prob_max = max(probabilidad2(c,y,p,q) for c in C)

    # Busco si existen varias palabras en el código, que proporcionen
    # para el vector y, ese valor para la probabilidad condicionada
    mejores_c = [c for c in C if probabilidad2(c,y,p,q) == prob_max]

    # Si existe más de una palabra del código que máximiza la probabilidad
    # se cambia el valor del parámero s a 0
    if len(mejores_c)!=1:
        s=0
    return s, mejores_c[0]


>>> p = 0.1
>>> q = 0.4
>>> C = [[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 1, 1]]
>>> y = [0,0,1]
>>> s,x = decodificadorML(C,y,p,q)
>>> if s==1:
        print('La decodificación es única y la palabra buscada es:',x)

    La decodificación es única y la palabra buscada es: [1, 1, 1]

>>> p = 0.1
>>> q = 0.3
>>> C = [[0, 0, 1], [0, 1, 0], [1, 1, 1]]
>>> y = [0,0,0]
>>> s,x = decodificadorML(C,y,p,q)
>>> if s==0:
        print('La decodificación no es única')

    La decodificación no es única
```

La función *discrepancia* es la definición de la medida de discrepancia no simétrica entre dos vectores con coeficientes $\mathbb{F}_2$ de longitud arbitraria pero igual, que hemos presentado en el último capitulo, se define en función de las probabilidades de transmisión del canal.

```
def discrepancia(y, x, p, q):
    #Se comprueba que la longitud de ambos vectores es igual
    if len(y) != len(x):
        raise ValueError("Los vectores y y x deben tener la misma longitud.")
```

```
    # Cálculo del número de unos en el vector y que son ceros en el vector x
    d10 = sum(1 for i in range(len(y)) if y[i] == 1 and x[i] == 0)
    # Cálculo del número de ceros en el vector y que son unos en el vector x
    d01 = sum(1 for i in range(len(y)) if y[i] == 0 and x[i] == 1)

    # Cálculo del parámetro gamma
    gamma_value = gamma(p,q)

    return gamma_value * d10 + d01
```

```
>>> y = [1, 0, 0, 1]
>>> x = [0, 1, 0, 0]
>>> discrepancia(y, x, 0.05, 0.2)
```

```
4.558833615477598
```

```
>>> y = [0, 0, 0, 1]
>>> x = [0, 1, 1, 1]
discrepancia(y, x, 0.05, 0.2)
```

```
2.0
```

La función *discrepancia simétrica* es la segunda noción de discrepancia que se presenta en el último capítulo entre vectores con coeficientes en $\mathbb{F}_2$ que tienen la misma longitud y que atraviesan un canal asimétrico de parámetros p y q. Se calcula a partir de la discrepancia no simétrica.

```
def discrepancia_simetrica(y,x,p,q):

    #Se comprueba que la longitud de ambos vectores es igual
    if len(y) != len(x):
        raise ValueError("Los vectores y y x deben tener la misma longitud.")

    dis_sim = discrepancia(y,x,p,q) - hamming_weight(y)*(gamma(p,q)-1)
    return dis_sim
```

```
>>> p = 0.2
>>> q = 0.3
>>> x = [0,0,1,0]
>>> y = [0,1,0,0]
>>> discrepancia_simetrica(y,x,p,q)
```

```
    2.0
```

```
>>> p = 0.1
>>> q = 0.4
>>> x = [0,0,1,0,0]
>>> y = [0,1,0,0,1]
>>> discrepancia_simetrica(x,y,p,q)
```

```
   3.0
```

La función *discrepancia mínima* encuentra la discrepancia mínima no simétrica que existe entre dos palabras de un código de bloque binario, es decir, se asume que todas las palabras tienen la misma longitud. Además, las probabilidades de transición del canal binario asimétrico son p y q.

```
def discrepancia_minima(C,p,q):
    min_dis_c = discrepancia(C[0], C[1], p,q)

    # Se recorren todas las palabras del código en orden
    for i in range(len(C)):

        # Se compara cada una de ellas con cada una de las siguientes
        for j in range(len(C)):

            if j>i:
                # Debido a la su asimetría hay que calcular las dos
                d_izq = discrepancia(C[i], C[j],p,q)
                d_der = discrepancia(C[j], C[i],p,q)
                d = min(d_izq, d_der)

                if d < min_dis_c:
                    min_dis_c = d
    return min_dis_c
```

```
>>> C =[[0,0,0], [0,1,1], [1,1,1]]
>>> discrepancia_minima(C, 0.1,0.3)
```

```
    1.0
```

```
>>> C =[[0,0,0,0], [0,1,1,1], [1,1,1,0]]
>>> discrepancia_minima(C, 0.1,0.25)
```

```
    2.572995379644637
```

La función *discrepancia mínima simétrica* encuentra la discrepancia mínima, pero esta vez la simétrica, que existe entre dos palabras de un código de bloque binario, es decir, se asume que todas las palabras tienen la misma longitud. Además, las probabilidades de transición del canal binario asimétrico son p y q.

```
def discrepancia_minima_simetrica(C,p,q):
    min_dis_sim_c = discrepancia_simetrica(C[0], C[1], p,q)

     # Se recorren todas las palabras del código en orden
    for i in range(len(C)):

        # Se compara cada una de ellas con cada una de las siguientes
        for j in range(len(C)):

            if j>i:
                # Por su simétria, da igual calcularla para cualquier orden
                # de las dos palabras
                d = discrepancia_simetrica(C[i], C[j],p,q)

                if d < min_dis_sim_c:
                    min_dis_sim_c = d
    return min_dis_sim_c
```

```
>>> C = [(1,0,0), (0,1,1), (1,1,1)]
>>> discrepancia_minima_simetrica(C,0.1,0.3)
```

```
-0.5424874983228443
```

```
>>> C =[[0,0,0,0], [0,1,1,1], [1,1,1,0]]
>>> discrepancia_minima_simetrica(C,0.1,0.25)
```

```
0.8540092407107256
```

La función con el nombre de *decodificador*, calcula, dado un código binario de bloque de longitud n arbitraria y un vector de la misma longitud que las palabras del código con coeficientes también en el cuerpo de dos elementos $\mathbb{F}_2$ y que puede o no pertenecer al código, la única palabra del código que minimiza la discrepancia no simétrica con este vector. En el caso de que no exista un única palabra el parámetro s toma el valor cero indicándonos que se ha producido un error, precisamente porque varias palabras del código proporcionado minimizan la discrepancia con el vector recibido.

```
def decodificador(C, r, p,q):
    s=1
    # Cálculo de la discrepancia de cada palabra del código con la recibida
    min_dis = min(discrepancia(r, c, p, q) for c in C)

    # Búsqueda de las palabras en el código, que proporcionen para el vector r
    # ese valor de discrepancia mínima
    mejores_c = [c for c in C if discrepancia(r,c,p,q) == min_dis]

    # Si existe más de una palabra del código que máximiza la probabilidad
    # se cambia el valor del parámero s a 0
    if len(mejores_c)!=1:
        s=0
    return s, mejores_c[0]
```

```
>>> p = 0.1
>>> q = 0.4
>>> C = [[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 1, 1]]
>>> y = [0,0,1]
>>> s,x = decodificador(C,y,p,q)
>>> if s==1:
        print('La decodificación es única y la palabra buscada es:',x)
```

```
La decodificación es única y la palabra buscada es: [1, 1, 1]
```

```
>>> p = 0.1
>>> q = 0.3
>>> C = [[0, 0, 1], [0, 1, 0], [1, 1, 1]]
>>> r = [0,0,0]
>>> s,x = decodificador(C,r,p,q)
>>> if s==0:
        print('La decodificación no es única')
```

```
La decodificación no es única
```

La siguiente función *probabilidad condicionada* que se define es la que calcula la probabilidad condicionada de recibir el vector y habiéndose enviado por el canal asimétrico de parámetros p y q, la palabra del código x. Ambos vectores binarios deben tener la misma longitud, se supone que el ruido afecta a los dígitos existentes no se pierden ni se producen nuevos dígitos.

```
def prob_condicionada(y,x,p,q):

    # Número de unos en el vector y que son ceros en la palabra x
    d10 = sum(1 for i in range(len(y)) if y[i] == 1 and x[i] == 0)

    #Número de ceros en el vector y que son unos en la palabra x
    d01 = sum(1 for i in range(len(y)) if y[i] == 0 and x[i] == 1)

    Pr = (q/(1-p))**(d01) * (p/(1-q))**(d10) * (1-q)**(hamming_weight(y)) *
         * (1-p)**(len(y)-hamming_weight(y))
    return Pr
```

```
>>> x = [0,0,1,0,1,0]
>>> y = [1,1,1,0,1,0]
>>> prob_condicionada(y,x, 0.1,0.3)

    0.00396900000000000

>>> x = [0,1,1,0,1,0]
>>> y = [0,1,0,0,1,0]
>>> prob_condicionada(y,x, 0.1,0.3)

    0.107163000000000
```

A continuación, vamos a definir dos funciones auxiliares. La primera con el nombre de *espacio vectorial* crea una lista con todos los vectores del espacio vectorial sobre el cuerpo de p elementos, aunque en nuestro caso solo sería necesario el cuerpo de dos elementos, y de dimensión n. La segunda función *comparar listas* devuelve un valor numérico tras comprobar si dos listas son iguales elemento a elemento, si todos los elementos son iguales y por tanto lo son las listas devuelve un 1, y si no, devuelve un 0, además se asegura que la longitud de ambas listas que se quieren comparar coincide.

```
def espacio_vectorial(p, n):
    return [list(v) for v in product(range(p), repeat=n)]
```

```
def comparar_listas(l,m):

    #Se comprueba que la longitud de ambas listas es igual
    if len(l) != len(m):
        raise ValueError("Los vectores y y x deben tener la misma longitud.")

    else :
        n = len(l)
        s=0
```

```
        for i in range(n):
             if l[i]==m[i]:
                s = s+1
        if s == n:          # Si todos los elementos son iguales
            return 1
        else :
            return 0
```

La siguiente función es *PUD*, que calcula la probabilidad de decodificación errónea de un código binario en función de las probabilidades de transmisión del canal p y q.

```
def PUD(C,p,q):
    n = len(C[0])

    # Se crea una lista con todos los vectores del espacio vectorial
    elementos = espacio_vectorial(2,n)
    P = 0

    #Se recorren todas las palabras del código
    for c in C:

        # Para cada palabra se recorren todos los posibles vectores
        for y in elementos:
            s, d = decodificador(C,y,p,q)

            # Si el vector y no decodifica por la palabra del código c,
            # se suma la probabilidad condicionada de obtener y enviado x.
            if s==1:
                if comparar_listas(d,c) == 0:
                    P = P + prob_condicionada(y,c,p,q)
            else:
                P = P + prob_condicionada(y,c,p,q)

    P = P* (1/len(C))
    return P
```

```
>>> codigo= [[0, 0, 0],[0, 1, 1],[1, 1, 1]]
>>> PUD(codigo,0.1,0.3)

    0.244666666666667

>>> codigo= [[0, 0, 1],[0, 1, 0],[1, 1, 1]]
>>> PUD(codigo,0.1,0.3)

    0.339666666666667
```

La función *conjunto S* calcula todos los elementos que pertenecen al conjunto

$$\mathrm{S}(h) = \{s < h \quad | \quad s = a\gamma + b \quad \text{y} \quad a, b \in \mathbb{N}\}$$

dado un número $h \in \mathbb{R}$ arbitrario. Como los elementos del conjunto son combinaciones lineales de 1 y γ, su valor depende de los parámetros p y q.

```
def conjunto_S(h,p,q):
    # Cálculo del parámetro gamma
    g = gamma(p,q)

    # Elegimos un número natural un poco más grande que el número
    n = ceil(h)
    S = []

    #Recorremos todos los números naturales más pequeños que él
    for a in range(n):
        for b in range(n):

            # Si la combinación lineal obtenida es menor que el número inicial,
            # la añadimos al conjunto
            if a+g*b<h:
                S.append(a+g*b)
    return S
```

```
>>> conjunto_S(3.4, 0.1, 0.4)

    [0.0, 2.209511291351455, 1.0, 3.209511291351455, 2.0, 3.0]

>>> conjunto_S(1.89, 0.2, 0.4)

    [0.0, 1.584962500721156, 1.0]
```

La función *binomial real* es una adaptación de la función binomial que solo esta definida cuando los coeficientes del binomio son naturales. Cuando los coeficientes binomiales no son naturales, simplemente se define el valor del número combinatorio como cero. Además, se presenta computacionalmente otro problema, por la definición del número λ, en la Nota 3.31, los coeficientes del número combinatorio pueden ser números reales con infinitos decimales, pero la precisión de *SageMath* es limitada, los números reales que almacena tienen 16 cifras decimales. Esto provoca que al hacer las operaciones que se piden al calcular las cotas para la probabilidad de decodificación no exitosa, exista la probabilidad de que no obtenga un número natural cuando el resultado de la operación sí lo es, precisamente por operar con un número de cifras finito.

```
def binomialreal(n,m):

    # Comprobamos si n y m son aproximadamente números naturales
    if (N(n,13) in NonNegativeIntegers()) and (N(m,13) in NonNegativeIntegers()):

        # Cálculo del número combinatorio con los números naturales a
        # los que se aproximan
        s = binomial(round(n),round(m))

    else :
        s = 0
    return s
```

```
>>> binomialreal(5.0000000000000002, 1)

    5


>>> binomialreal(8, 0.3)

    0
```

La función *cota 1* calcula la primera cota para la probabilidad de decodificación no exitosa y análogamente la función *cota 2* calcula la segunda cota para la decodificación no exitosa. Como ya se comento anteriormente, es necesario tener cuidado con el conjunto S, ya que este puede ser vacío y en este caso no debemos sumar nada. La otra dificultad que presentaba es el calculo de los coeficientes binomiales, pero que ya que solvento en la función *binomial real*.

```
def cota_1(C,p,q):
    n = len(C[0])
    suma = 0
    g = gamma(p,q)

    # Cálculo del número de palabras del código que tienen peso j
    for j in range(n+1):
        w_j = polinomio_pesos(C)[j]


        for i in range(n+1):
            suma_i = ((1 - q)**i)*((1 - p)**(n - i))
            d = discrepancia_minima(C,p,q)      # Discrepancia mínima del código

            # Conjunto de valores del conjunto S, para esa combinación de i y j
            S = conjunto_S((d + (i-j)*(g - 1))/2, p, q)

            # Comprobamos si el conjunto S contiene algún elemento,
            # si esta vacio no sumamos nada
            if len(S)!=0:

                for s in S:
                    b1 = binomialreal(j, (i*g - s + j) / (g + 1))
                    b2 = binomialreal(n - j, (s - j + i) / (g + 1))
                    suma = suma + w * suma_i * ((q/(1 - p))**s) * b1 * b2

    suma =  1 - (1 / len(C))*suma
    return suma
```

```
def cota_2(C,p,q):
    n = len(C[0])
    suma = 0
    g = gamma(p,q)

    # Cálculo del número de palabras del código que tienen peso j
    for j in range(n+1):
        w_j = polinomio_pesos(C)[j]

        for i in range(n+1):
            suma_i = ((1 - q)**i) * ((1 - p)**(n - i))
            d = discrepancia_minima_simetrica(C, p, q)

            # Cálculo de los elementos de conjunto S para este valor de i
            S = conjunto_S( (d + i* (g - 1)) / 2, p, q)

            # Comprobamos si el conjunto S tiene o no elementos
            if len(S)!=0:
                for s in S:
                    b1 = binomialreal(j, (i*g - s + j) / (g + 1))
                    b2 = binomialreal(n - j, (s - j + i) / (g + 1))
                    suma = suma + w_j * suma_i * ((q / (1 - p))**s) * b1* b2

    suma =  1 - (1 / len(C))*suma
    return suma
```

```
>>> C = [[0, 0, 1],[0, 1, 0],[1, 1, 1]]
>>> print(cota_1(C,0.1,0.3))
>>> print(cota_2(C,0.1,0.3))

    0.507666666666667
    0.360666666666667

>>> C = [[0, 0, 0],[0, 1, 1],[1, 1, 1]]
>>> print(cota_1(C,0.1,0.3))
>>> print(cota_2(C,0.1,0.3))

    0.495666666666667
    0.738666666666667
```