Teoría de la información y la compresión de cadenas

Autor
Esteban Hernández Ramírez

Trabajo presentado como requisito para optar por el título de
profesional en Matemáticas Aplicadas y Ciencias de la Computación

Director, tutor
Carlos Eduardo Álvarez Cabrera

Escuela de Ingeniería, Ciencia y Tecnología
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

Bogotá-Colombia
2022

Resumen

El problema de la *compresión sin perdida* consiste en implementar la *codificación (únicamente decodificable)* de un alfabeto, que asigna a cada cadena de símbolos del alfabeto el *código* de menor longitud posible. Encontrar esta representación de menor tamaño de un conjunto de datos puede ahorrar costos en el espacio de almacenamiento, tiempo en transferencia de los datos o número de operaciones en su procesamiento, dentro de un computador. Lo anterior hace de la *compresión sin perdida* un objetivo razonable dentro de las *Ciencias de la Computación* y representa un reto importante durante el desarrollo de muchas soluciones tecnológicas.

La *teoría de la información*, por su parte, ha establecido el formalismo matemático necesario para el estudio de medidas cuantitativas de *información* como la *entropía de Shannon* y ha encontrado su lugar dentro de la implementación de la *compresión sin perdida* al proveer algunas de las herramientas teóricas necesarias para el estudio de los modelos que describen las fuentes de datos en la teoría de la *codificación*.

Por otro lado, la cercana relación que se ha encontrado entre la *teoría de la información* y la *teoría de la compresión sin perdida* ha motivado a muchos autores a ingeniar formas de medir a través de la compresión de archivos la información en ellos. Lo que ha resultado en interesantes aplicaciones de la *compresión sin perdida* en el *aprendizaje automático*, particularmente, en la clasificación de textos escritos en lenguaje natural o cadenas de ADN.

En este escrito, se presenta una revisión monográfica acerca de cómo la *teoría de la información* se aplica a la *compresión sin perdida*. Para esto, se presentan algunas de las implementaciones de la compresión sin perdida en la *teoría de códigos*. Las demostraciones, gráficas, análisis y algoritmos en este escrito generalizan algunos de los hechos más importantes acerca de codificaciones binarias, que se han enunciado en la literatura, al caso general de un alfabeto de tamaño arbitrario.

Finalmente, se presenta una aplicación de la compresión sin perdida a la clasificación de lenguajes naturales, mediante la aplicación del *algoritmo de codificación LZ77* para estimar algunas medidas de información bien conocidas en la literatura, las cuales se emplean como medida de distancia para comparar los lenguajes entre sí. El resultado de la clasificación es presentado en la forma de árboles filogenéticos de los lenguajes.

Agradecimientos

Doy gracias a Dios por permitirme llegar hasta aquí.

Gracias a mi mamá y a mi papá quienes siempre me apoyan y me guían. Ustedes se merecen siempre todo lo mejor. Los amo.

Doy gracias a mi supervisor de trabajo de grado y profesor de carrera: Carlos Eduardo Álvarez Cabrera. Su constante apoyo ha sido fundamental para mí como estudiante y como persona. Le agradezco infinitamente todo lo enseñado.

Agradezco a todas las personas que me han acompañado durante este camino. Cada uno de ellas está presente en mí y me han inspirado para ser mejor persona.

Índice general

Capítulo 1

Introducción

1.1. Motivación

La idea de una *máquina de Turing* que retorna una secuencia infinita de $1s$ y $0s$, luego de haber operado sobre ella bajo reglas codificadas en otra secuencia de $1s$ y $0s$, lleva naturalmente a la pregunta de sí, dada una secuencia cualquiera, es posible determinar a partir de ella la secuencia de reglas codificadas que la generaron: más importante aún, cómo determinar la secuencia de reglas codificadas más corta capaz de generar la secuencia dada. Este problema ha sido formulado rigurosamente en *la teoría de la computación* como la *Complejidad de Kolmogórov*; introducida independientemente en la decada de los $60's$ por Solomonoff en su artículo [18], Kolmogorov en su artículo [12] y Chaitin en su artículo [2].

Algunas aproximaciones como [15] han definido una *cadena aleatoria* como aquella que no se puede comprimir y para la cual la forma más sucinta de transmitirla es ella misma. Al mismo tiempo, resultados como los presentados en [22] demuestran que, de manera general, la tarea de determinar la secuencia de reglas codificadas más corta (*Complejidad de Kolmogórov*) es incomputable. No obstante, los esfuerzos de la comunidad científica se han centrado últimamente en idear formas de comprimir cualquier secuencia, lo más que sea posible. En 1948 Claude E. Shannon probaría en [17] dos teoremas que tratan acerca de la transmisión de información sin perdida con la menor cantidad de datos posible a través de un canal de comunicación:

1. En el primero de estos teoremas, el **teorema de codificación de la fuente de Shannon**, quedaría demostrado que, teóricamente, existe la posibilidad de comprimir sin perdida la codificación de una fuente de datos, en un límite asintótico: Shannon demostró que existe un esquema de codificación, que omitiendo la codificación de las cadenas menos frecuentes de la fuente, logra disminuir la cantidad de mensajes a trasmitir a la cantidad mínima necesaria para que la probabilidad de cometer un error de codificación, por emitir una secuencia sin codificación, se desvanezca. Para esto, Shannon definió el concepto de conjunto de cadenas típicas de una fuente, el cual, cuando el tamaño de la cadena tiende a infinito, tiene una probabilidad unitaria, un tamaño exponencialmente más pequeño que el conjunto de todas las cadenas posibles y todos sus elementos son equiprobables, a partir de lo cuál codifica únicamente las cadenas del conjunto típico, reduciendo el número total de bits necesarios para codificar los mensajes generables por la fuente. También quedaría demostrado que la tasa de compresión sin perdida mínima es aquella del esquema de codificación de Shannon y que alcanzar cualquier otra tasa menor que esa significaría asumir una probabilidad de error significativa [17, 23].

2. En el segundo de estos teoremas, el **teorema de codificación del canal de Shannon**, quedaría demostrado que es posible codificar los datos que se transmiten a través de un canal inseguro con una cantidad aceptable de bits físicos, manteniendo una probabilidad de error en la decodificación aceptable, tal cómo se menciona en la referencia [23].

No en vano, el teorema de codificación de la fuente de Shannon se aprovecha para idear maneras de comprimir la codificación de una fuente y juzgar su rendimiento, mientras que el teorema de codificación del canal de Shannon sirve para la transmisión de los datos. Por lo cual, en cuanto a este escrito únicamente le interesa la compresión sin perdida, solo nos enfocaremos en el primer teorema, más adelante, en la sección 3.

El trabajo pionero de los autores *Abraham Lempel* y *Jacob Ziv*, en su artículo [25] de 1977, demostró que existen formatos de *compresión sin perdida* para los cuales el tamaño óptimo de un archivo comprimido tiende a estimar la *entropía* de la fuente generadora. Esta *entropía* fué una medida de *información* propuesta por *Claude Shannon* en la década de los $50s$ en su artículo [17]: esta medida es continua y es máxima cuando la probabilidad de generar cualquier símbolo en la secuencia es la misma. Lo anterior ha permitido calcular una aproximación a la *Complejidad de Kolmogórov* de una secuencia. Otras aproximaciones como la presentada en [20] confirmarían más adelante lo visto por *Lempel y Ziv*, y acabaría por reforzar la creencia de que la *Complejidad de Kolmogórov* y *la Entropía de Shannon* están más relacionadas de lo que podría parecer en un principio.

En la literatura se han propuesto formas de medir el *contenido de información* de un mensaje (algunas de las cuales se recogen en [14]): muchas veces, motivadas por dar respuesta a preguntas acerca de la *Complejidad de Kolmogórov*, [12], y la compresión de cadenas. En la teoría de la información, lo que actualmente se reconoce (y estudia) como contenido de información de un mensaje proviene de las contribuciones de Claude Shannon, en la década de 1950 con su trabajo [17], a la idea original de Ralph Hartley en: transmisión de información, [7]. Antes de Shannon, el estudio de la codificación de un mensaje para su transmisión habían llevado a Hartley a formular una primera medida cuantitativa del *contenido de información* de un mensaje, en términos de la codificación del mensaje ([11]).

El desarrollo de la *teoría de la información*, sobre las bases del trabajo de Shannon, han ayudado a dar respuesta a muchas de las dudas originales acerca de la *Complejidad de Kolmogórov* y la *compresión sin perdida* de cadenas. Más allá de eso, ha sido aplicada con éxito en el ámbito académico y comercial: por ejemplo, en la clasificación del lenguaje natural, que han trabajado autores como los del artículo [1], y la implementación del estándar de compresión de archivos GZIP en todos los sistemas operativos comercializados actualmente ([24]).

En este escrito, se explican algunas aplicaciones bien conocidas de la *teoría de la información* a la *compresión sin perdida* y la aplicación de ésta, a su vez, a la clasificación del lenguaje natural (recreando algunos de los resultados obtenidos en [1] y [3]). Para esto, se presentan algunas de las principales medidas de *información* en términos de la codificación de un mensaje. Luego, se presenta el algoritmo de codificación LZ77 y la relación que tiene con algunas de las medidas de *información* más conocidas, particularmente, aquellas medidas de *información* que se han formulado en términos de la codificación de un mensaje: contenido de información de Hartley y la entropía de Shannon. Finalmente, se presentan dos procedimientos para construir árboles filogenéticos del lenguaje natural, adaptados del trabajo de los autores de los artículo [1] y [3], reproduciendo y ampliando los resultados obtenidos allí, empleando el algoritmo de codificación LZ77 y algunas de las medidas de compresión presentadas: los resultados que se obtienen de llevar a cabo ese procedimiento son presentados en la forma de árboles filogenéticos para distintos lenguajes de entrada y son comparadas visualmente con la aproximación puramente lingüística del artículo [6].

1.2. Objetivo general

Realizar una revisión monográfica de los fundamentos de la teoría de la información y su aplicación a la compresión de cadenas.

1.2.1. Objetivos especificos

1. Revisar algunas de las definiciones de información más estudiadas en la literatura.

2. Revisar en la literatura de qué manera estas definiciones de información se aplican a la compresión de cadenas.

3. Implementar el algoritmo de Lempel-Ziv para la compresión de cadenas.

4. Observar los efectos del algoritmo de Lempel-ziv sobre una cadena en términos de las medidas de información.

5. Revisar una aplicación del algoritmo de Lempel-ziv para clasificar lenguajes.

Capítulo 2

Símbolos individuales

Suponga que un mensaje compuesto por símbolos de un alfabeto $\mathcal{A}$ se transmite a través de un medio físico que únicamente puede transmitir símbolos de un alfabeto $\mathcal{B}$: los símbolos del alfabeto $\mathcal{A}$ se *codifican* con símbolos del alfabeto $\mathcal{B}$.

Los *códigos*, que son el resultado de la *codificación* y que representan cada símbolo del alfabeto $\mathcal{A}$, están involucrados en el siguiente proceso de cuatro fases:

1. *Generación* del mensaje: en la que se asume, inicialmente, que todo símbolo del alfabeto $\mathcal{A}$ es generado con una probabilidad conocida.

2. *Codificación* del mensaje: en la que se requiere que a cada símbolo del alfabeto $\mathcal{A}$ se le asigne un código compuesto únicamente por símbolos del alfabeto $\mathcal{B}$.

3. *Transmisión* del mensaje: en la que únicamente se transmiten símbolos del alfabeto $\mathcal{B}$.

4. *Decodificación* del mensaje: en la que es necesario que el código asignado a cada símbolo del alfabeto $\mathcal{A}$ sea único, de modo que sea posible recuperar cada símbolo a partir de su código, sin ambigüedad [1].

Estos requerimientos para los códigos hacen de la *codificación de un alfabeto* un tipo muy específico de función matemática, la cual se formula en la sección 2.2. En la sección 2.1, se modela matemáticamente el proceso de *generación de mensajes*: restringiendo su formulación a la de una *fuente de símbolos individuales*.

2.1. Modelamiento de una *fuente de símbolos*

Asumiendo que los mensajes son generados símbolo a símbolo, es razonable pensar que existe una *fuente*, la cual emite símbolos del alfabeto $\mathcal{A}$ con cierta regularidad. Esta fuente puede caracterizarse por la probabilidad con la que emite cada símbolo del alfabeto $\mathcal{A}$. Por simplicidad, durante todo este capítulo se asume que la probabilidad de emitir un símbolo no depende de los símbolos que se han generado antes. Esta restricción permite pensar una fuente de símbolos como una variable aleatoria discreta X, la cual toma valores, x, en el alfabeto $\mathcal{A}$ ($x \in \mathcal{A}$), de acuerdo a cierta *función de masa de probabilidad*, $P_X(x)$. Definición adaptada de [5].

[1] La ambigüedad surge cuando un mismo código puede interpretarse como dos símbolos distintos.

Definición 2.1.1 (*Fuente de símbolos*)**.** Se define una *fuente de símbolos* de un alfabeto $\mathcal{A}$ como una variable aleatoria X que toma valores sobre el alfabeto $\mathcal{A}$ de acuerdo a cierta función de masa de probabilidad $P_X(x)$.

2.2. *Codificación de una fuente de símbolos*

La formulación de la *codificación de una fuente de símbolos*, tal cómo se definió una fuente de símbolos en la definición 2.1.1, consiste en: primero, especificar la manera cómo se *codifica* el alfabeto $\mathcal{A}$ y luego extender esa *codificación* a la variable aleatoria X, que modela la fuente, mediante la inclusión de la *información* de la función de masa de probabilidad, $P_X(\cdot)$, en la escogencia del código.

2.2.1. *Codificación de un alfabeto*

La manera cómo se formula la *codificación de un alfabeto* es la siguiente:

Definición 2.2.1 (*Codificación de un alfabeto*)**.** Dados un alfabeto $\mathcal{A}$, de tamaño α: $\mathcal{A} = \{a_1, a_2, \ldots, a_\alpha\}$ y otro alfabeto $\mathcal{B}$, de tamaño β: $\mathcal{B} = \{b_1, b_2, \ldots, b_\beta\}$, una *codificación* de $\mathcal{A}$ en términos de $\mathcal{B}$ es una función C que asigna a cada símbolo de $\mathcal{A}$ una tupla ordenada de símbolos de $\mathcal{B}$ de longitud $\ell > 0$:

$$C : \mathcal{A} \longrightarrow \mathcal{B}^*.^2$$

Para cada $a_i \in \mathcal{A}$, $C(a_i)$ denota la tupla ordenada asociada al símbolo a_i bajo el código C. [5, 21]

Las tuplas ordenadas que se asignan a los símbolos de $\mathcal{A}$ en la definición 2.2.1 son llamadas *palabras código* de longitud ℓ. La elección de ℓ en la codificación, así como de la regla de asignación en sí misma, dependen de los tamaños de los alfabetos $\mathcal{A}$ y $\mathcal{B}$. Naturalmente, los alfabetos $\mathcal{A}$ y $\mathcal{B}$ podrían no coincidir, particularmente, por estar compuestos por símbolos diferentes o por tener tamaños diferentes, o ambas cosas al mismo tiempo. Cuando los alfabetos tienen tamaños diferentes, existen las siguientes dos posibilidades:

- $\alpha \leq \beta$ (más símbolos individuales de $\mathcal{B}$ que símbolos de $\mathcal{A}$ que codificar)
- $\beta < \alpha$ (más símbolos de $\mathcal{A}$ que codificar que símbolos individuales de $\mathcal{B}$)

En el caso en el que $\alpha \leq \beta$, hay más o igual cantidad de símbolos individuales de $\mathcal{B}$ que símbolos de $\mathcal{A}$ que codificar. En este caso, la codificación puede consistir en una función biyectiva entre $\mathcal{A}$ y un subconjunto $\mathcal{S}$ de $\mathcal{B}$, tal que a cada símbolo de $\mathcal{A}$ le corresponde un único símbolo de $\mathcal{B}$. Lo anterior cumple, naturalmente, todos los requerimientos para un código, que se establecieron en la introducción, ya que permite codificar todos los símbolos del alfabeto $\mathcal{A}$ sin ambigüedad, con longitud de la palabra código igual a uno: $\ell = 1$.

Por otro lado, en el caso en el que $\beta < \alpha$, hay más símbolos de $\mathcal{A}$ que codificar que símbolos individuales de $\mathcal{B}$. Por lo cual, al menos, uno de los α símbolos de $\mathcal{A}$ deberá ser codificado con dos, o más, símbolos de $\mathcal{B}$, con el fin de posibilitar la codificación de todos los símbolos del alfabeto $\mathcal{A}$ sin ambigüedad. Ante esto, se tienen dos opciones:

1. Asignar a todos los símbolos del alfabeto $\mathcal{A}$ palabras código de la misma longitud, ℓ, con $\ell \geq 2$.
2. Asignar a cada símbolo del alfabeto $\mathcal{A}$ una palabra código de longitud distinta, con, al menos, una palabra código de longitud mayor o igual que dos.

[2] $\mathcal{B}^*$ denota el conjunto de todas las cadenas de longitud mayor que cero compuestas por la concatenación de símbolos del alfabeto $\mathcal{B}$.

Esta elección define dos tipos de códigos diferentes: (1) *códigos de longitud fija*, para palabras código de la misma longitud, y (2) *códigos de longitud variable*, para palabras código de longitudes distintas. En las siguientes dos secciones, 2.2.1.1 y 2.2.1.2, se definen y estudian estos dos tipos de códigos por separado.

2.2.1.1. Códigos de longitud fija

Para codificar sin ambigüedad todos los símbolos del alfabeto $\mathcal{A}$ con palabras código de la misma longitud, ℓ, la longitud ℓ debe garantizar que sea posible formar, al menos, α palabras código diferentes con combinaciones de los β símbolos del alfabeto $\mathcal{B}$: $\beta^{\ell} \geq \alpha$. Es decir, en el peor de los casos, ℓ debe ser tal que

$$\beta^{\ell} = \alpha,$$

de donde se sigue que la longitud ℓ mínima requerida es, en teoría,

$$\ell = log_{\beta}(\alpha). \tag{2.1}$$

Así que, una opción razonable para codificar sin ambigüedad todos los símbolos de $\mathcal{A}$ con palabras código de la misma longitud es con combinaciones de símbolos de $\mathcal{B}$ de longitud $log_{\beta}(\alpha)$. No obstante, la longitud de las palabras código de la ecuación 2.1 podría no ser un número entero, por lo que, en la práctica, para solucionar esto, se suele aproximar mediante el número entero mayor más cercano:

$$\ell = \lceil log_{\beta}(\alpha) \rceil. \tag{2.2}$$

Para la escogencia de la longitud de las palabras código de la ecuación 2.2, $\ell = \lceil log_{\beta}(\alpha) \rceil$, es claro que la longitud de las palabras código aumenta con el tamaño del alfabeto $\mathcal{A}$, α. Sin embargo, esto no quiere decir que la longitud de la codificación de un mensaje (pensada como la concatenación de las palabras código de los símbolos que componen el mensaje)[3] aumenta arbitrariamente al aumentar el tamaño del alfabeto con el que se escribe el mensaje: si bien, al cambiar el alfabeto con el que se escribió el mensaje originalmente, por otro más grande, aumenta la longitud ℓ, al mismo tiempo el mensaje podría escribirse con una menor cantidad de símbolos del alfabeto más grande. La manera cómo se escribe un mensaje con una menor cantidad de símbolos, cuando se escribe con un alfabeto de tamaño más grande, es escribiendo directamente combinaciones de, dos o más, símbolos del alfabeto original como símbolos individuales del alfabeto más grande. Así, un mensaje originalmente escrito con símbolos de un alfabeto de cierto tamaño puede escribirse con una menor cantidad de símbolos de un alfabeto más grande.

Por ejemplo, si se tiene un mensaje compuesto por n símbolos de un alfabeto $\mathcal{A}$, de tamaño α, y se considera otro alfabeto $\mathcal{A}'$, de tamaño $\alpha' = \alpha^2$, se puede notar que la longitud de la codificación del mensaje escrito con símbolos del alfabeto $\mathcal{A}'$ puede ser igual que la longitud de la codificación del mensaje original, sí el número de símbolos del alfabeto $\mathcal{A}'$ con el que se escribe el mensaje se reduce aproximadamente a la mitad:

$$n' \lceil log_{\beta}(\alpha') \rceil = n' \left\lceil log_{\beta}\left(\alpha^2\right) \right\rceil = n' \lceil 2 \cdot log_{\beta}(\alpha) \rceil = n \lceil log_{\beta}(\alpha) \rceil \qquad \text{si y solo sí} \qquad n' \approx \frac{n}{2}. \tag{2.3}$$

Es decir, sí se reemplazan pares de símbolos del alfabeto $\mathcal{A}$ por símbolos individuales del alfabeto $\mathcal{A}'$.

Un caso particular del ejemplo anterior es el siguiente: suponga que se tiene un mensaje compuesto por ocho símbolos de un alfabeto binario. Se pueden escribir todas las combinaciones de pares de símbolos del alfabeto binario como símbolos individuales de un alfabeto de cuatro símbolos, de modo que el mismo mensaje podría escribirse con un alfabeto de cuatro símbolos, del doble de símbolos, con la mitad de símbolos: Sea $\mathcal{A} = \{0, 1\}$ y $\mathcal{A}' = \{A, T, G, U\}$, observe que

$$00011011 \equiv \boxed{00}_{A} \boxed{01}_{T} \boxed{10}_{G} \boxed{11}_{U} \equiv ATGU \tag{2.4}$$

[3] La idea de codificar un mensaje como la concatenación ordenada de las palabras código de los símbolos que lo componen es la misma detrás de la *extensión de la codificación de un alfabeto*, la cual se define formalmente más adelante.

En el ejemplo 2.4, puede ver gráficamente cómo la longitud de un mensaje se reduce a la mitad, al reescribirlo con un alfabeto con el doble de símbolos, tal cómo se demuestra matemáticamente en la ecuación 2.3.

Así que, al aumentar el tamaño del alfabeto con el que se escribe un mensaje, disminuye la cantidad de símbolos necesarios para representar el mensaje. De manera general, la longitud de la codificación de un mensaje (pensada como la concatenación de las palabras código de los símbolos que componen el mensaje) se puede conservar exactamente igual al cambiar el alfabeto con el que se escribe el mensaje, cuando

$$\ell' = \lceil log_\beta(\alpha') \rceil = log_\beta(\alpha')$$

y el nuevo alfabeto consiste en una potencia del alfabeto original, $\mathcal{A}' = \mathcal{A}^k$ para $k \in \mathbb{Z}^+$, con longitud α^k, ya que así la longitud fija en el nuevo alfabeto, ℓ', es

$$\ell' = log_\beta\left(\alpha^k\right) = k \cdot log_\beta(\alpha),$$

por lo cual la longitud del mensaje en el nuevo alfabeto, n', se puede reducir como

$$n' = \frac{n}{k}.$$

De modo que, $n' \cdot \ell' = n \cdot \ell$. Además, observe que esta longitud fija se reescala bajo cambios en el tamaño del alfabeto $\mathcal{B}$, ya que

$$log_\gamma(\alpha) = log_\gamma\left(\beta^{log_\beta(\alpha)}\right) = log_\beta(\alpha) \cdot log_\gamma(\beta). \tag{2.5}$$

Con lo visto hasta ahora es posible formular uno de los primeros *algoritmos de codificación*, basado en las ideas de esta sección, *el algoritmo de codificación de Hartley*, llamado así, debido a que, como se verá más adelante, la cantidad de la ecuación 2.1 posee su propia interpretación dentro de la *teoría de la información*, gracias al trabajo de *Ralph Hartley*[4]: *contenido de información de Hartley*. El pseudocódigo en el Apéndice A implementa la codificación de longitud fija del alfabeto $\mathcal{A}$ sobre palabras código del alfabeto $\mathcal{B}$, mediante la longitud fija de las palabras código de la ecuación 2.2.

2.2.1.2. Códigos de longitud variable

A diferencia de lo que se entiende como códigos de longitud fija, los *códigos de longitud variable* asignan a cada símbolo del alfabeto $\mathcal{A}$ una palabra código de longitud distinta. Se extiende la notación para denotar longitudes distintas de las palabras código: se denota como $\ell(a_i)$ a la longitud de la palabra código que se le asigna al símbolo a_i, para $i = 1,2,\ldots,\alpha$.

Los *códigos de longitud variable* pueden pensarse como una generalización de los códigos de longitud fija. El estudio de los *códigos de longitud variable* y su aplicación a la *compresión de cadenas* permitirá redefinir los códigos de longitud fija desde la perspectiva de la *teoría de la información*. Se hablará más acerca de las longitudes variables de las palabras código en la siguientes secciones.

2.2.2. Codificación de una variable aleatoria

Teniendo en cuenta que una variable aleatoria X, que modela una fuente de símbolos, toma valores en un alfabeto $\mathcal{A}$, es posible extender la codificación del alfabeto a la variable aleatoria. De modo que la codificación depende no únicamente de su dominio, sino también de su función de masa de probabilidad, $P_X(\cdot)$. Siendo así que a cada palabra código, compuesta por símbolos del alfabeto $\mathcal{B}$, le corresponde una probabilidad. Esto permite pensar las palabras código como una función de la probabilidad de cada símbolo. Esto permitirá definir las palabras código en función de la probabilidad de cada símbolo, lo que permite llevar a cabo un tipo muy especial de codificación: *la compresión*, más adelante.

La siguiente definición de la *codificación de una variable aleatoria* ha sido tomada de la referencia [5].

[4] *Ralph V. L. Hartley fue un ingeniero estadounidense que vivió a finales del siglo diecinueve y principios del veinte.*

Definición 2.2.2 (*Código de una variable aleatoria*)**.** Un *código* C para una variable aleatoria X es un mapeo desde $\mathcal{A}$, el rango de la variable aleatoria X, en $\mathcal{B}^*$, el conjunto de cadenas de longitud finita compuestas por símbolos de un alfabeto $\mathcal{B}$:

$$\begin{aligned} &C : \mathcal{A} \longmapsto \mathcal{B}^* \\ &x \in \mathcal{A} \mapsto C(x) \in \mathcal{B}^* \end{aligned} \tag{2.6}$$

La *codificación de una variable aleatoria* consiste en definir un código para la variable aleatoria: determinar la longitud de las palabras código y la regla de asignación en sí misma para la codificación del alfabeto.

2.3. *Extensión de un código*

A partir de la definición 2.2.2 se puede formalizar el concepto de *extensión de un código*, que consiste en considerar la concatenación de las palabras código de los símbolos que componen un mensaje como el código del mensaje en sí. La anterior es la manera cómo se asumió que se codificaban los mensajes durante toda la sección 2.2. Se le llama *extensión de un código*, simplemente, porque al concatenar las palabras código de los símbolos que componen un mensaje, para codificar el mensaje, da la impresión que la codificación del alfabeto se está extendiendo a los mensajes del alfabeto. Formalmente, la *extensión de un código* constituye una codificación de un *alfabeto extendido*, $\mathcal{A}^n$, compuesto por cadenas de longitud n de símbolos del alfabeto $\mathcal{A}$, pero no constituye una codificación de una fuente por sí misma: faltaría definir apropiadamente una función de masa de probabilidad para el alfabeto extendido de mensajes. La definición de *extensión de un código* presentada a continuación ha sido adaptada de la referencia [21].

Definición 2.3.1 (*Extensión de un código*)**.** La n-ésima *extensión* C^* de un código C de una fuente de símbolos X es el mapeo que va desde el conjunto de cadenas finitas de longitud n de símbolos del alfabeto $\mathcal{A}$, el rango de la fuente X, hasta el conjunto de cadenas finitas de longitud n de elementos del conjunto $\mathcal{B}^*$:

$$C^* : \mathcal{A}^n \longrightarrow (\mathcal{B}^*)^n ,$$

definido como:

$$C(x_1 x_2 \dots x_n) = C(x_1)\, C(x_2) \dots C(x_n) \tag{2.7}$$

donde $x_1 x_2 \dots x_n$ denota la cadena que consiste en la concatenación de n simbolos consecutivos del alfabeto $\mathcal{A}$, mientras que $C(x_1)\, C(x_2) \dots C(x_n)$ denota la concatenación de las n palabras código correspondientes a cada símbolo.

Esta definición permite llevar a cabo la codificación de un mensaje que consiste en una secuencia de n símbolos del alfabeto $\mathcal{A}$ concatenados consecutivamente, de la siguiente manera: definiendo una codificación, símbolo a símbolo, del alfabeto $\mathcal{A}$ de la fuente X, mediante la definición 2.2.1, y luego concatenando las palabras códigos de cada símbolo en el orden de los n símbolos que componen el mensaje.

Ejemplo: suponga que $C(x_1) = 00$ y que $C(x_1) = 11$, entonces $C(x_1 x_2) = 0011$.

No obstante, en la extensión de un código existe la posibilidad de que una misma cadena de símbolos del alfabeto $\mathcal{B}$ (código de un mensaje) pueda ser interpretada como varios mensajes distintos: dependiendo del procesamiento en bloques (*parsing of the string*) de los símbolos del alfabeto $\mathcal{B}$ dentro de un mensaje codificado, se podrían extraer palabras código diferentes, que corresponden a combinaciones de símbolos del alfabeto $\mathcal{A}$ diferentes. Es decir, la extensión de un código de una fuente X permite ambigüedad en cuanto a la interpretación de la codificación del alfabeto extendido. Sin embargo, esto puede ser solucionado definiendo apropiadamente la codificación del alfabeto y los modos de *decodificación*.

Ejemplo: suponga que $C(x_1) = 00$ y que $C(x_1) = 11$, pero además $C(x_4) = 0$, $C(x_5) = 01$ y $C(x_6) = 1$, entonces $C(x_1 x_2) = C(x_4 x_5 x_6) = C(x_4 x_4 x_6 x_6) = 0011$.

2.4. *Decodificación*

Para *decodificar* sin ambigüedad un mensaje codificado mediante la extensión de un código, es decir, que cada cadena de símbolos del alfabeto $\mathcal{B}$, en el dominio de la extensión del código, pueda ser interpretada como exactamente una única cadena de símbolos del alfabeto $\mathcal{A}$, independientemente de la implementación del código (longitud fija o longitud variable), la regla de asignación del código debe cumplir con determinadas características que permitan una interpretación sin ambigüedad de los mensajes codificados.

2.4.1. Códigos no-singulares

La siguiente definición de *código no-singular* ha sido tomada de la referencia [21].

Definición 2.4.1 (*Código no-singular*)**.** Un código C de una fuente X es *no-singular* si cada símbolo del alfabeto de la fuente, $\mathcal{A}$, se mapea a una palabra código diferente en el dominio de palabras código, $\mathcal{B}^*$:

$$a_i \neq a_j \implies C(a_i) \neq C(a_j), \quad \text{para todo} \quad a_i, a_j \in \mathcal{A}. \tag{2.8}$$

Observe que la condición de *no-singularidad* de un código, en la definición 2.4.1, es suficiente para lograr una descripción sin ambigüedad de cada valor del rango de una fuente X, pero aún no garantiza la unicidad de cualquier cadena de símbolos concatenados del alfabeto $\mathcal{B}$ ni tampoco su decodificabilidad sin ambigüedad.

2.4.2. Códigos únicamente decodificables

La siguiente definición de *código únicamente decodificable* ha sido tomada de la referencia [21].

Definición 2.4.2 (*Código únicamente decodificable*)**.** Un código C de una fuente X es *únicamente decodificable* si su extensión es un código no-singular.

La condición de *decodificabilidad única* de un código, en la definición 2.4.2, garantiza que no existe ambigüedad en la extensión de un código.

2.4.3. Códigos libres de prefijos

Un tipo especial de códigos únicamente decodificables son los *códigos libres de prefijos*. Intuitivamente, un código es *libre de prefijos* cuando ninguna palabra código es *prefijo* [5] de ninguna otra palabra código. La definición formal, a continuación, ha sido adaptada de la referencia [5].

Definición 2.4.3 (*Código libre de prefijos*)**.** Un código C de una fuente X es *libre de prefijos* si para todo $x_1, x_2 \in \mathcal{A}$, $C(x_1)$ no es un prefijo de $C(x_2)$.

Por ejemplo, para el alfabeto $\mathcal{A} = \{a, b, c, d, e\}$, el código:

$$(a, 0) \qquad (b, 10) \qquad (c, 110) \qquad (d, 1110) \qquad (e, 1111)$$

es un código libre de prefijos, de acuerdo a la definición 2.4.3. Mientras que el código:

$$(a, 0) \qquad (b, 10) \qquad (c, 101) \qquad (d, 1110) \qquad (e, 1111)$$

no es un código libre de prefijos, ya que la palabra código asignada al símbolo b, 10, es prefijo de la palabra código asignada al símbolo c, 101.

[5] Una palabra código c_1 es un prefijo de otra palabra código c_2 cuando la cadena de β-bits c_1 es exactamente igual a los primeros *longitud de* c_1 β-bits de la cadena de β-bits c_2 (en el orden de izquierda a derecha).

Usando códigos libres de prefijos todo mensaje puede transmitirse como la secuencia de palabras código concatenadas de cada uno de los símbolos del alfabeto $\mathcal{A}$ que lo componen, en el orden en el que aparecen dentro del mensaje, leyéndolo de izquierda a derecha.

2.4.4. Interpretación de un código libre de prefijos

Las palabras código de una codificación del alfabeto $\mathcal{A}$ se pueden representar como los recorridos desde la raíz hasta las hojas de un árbol β-ario particular: un árbol β-ario incompleto (es decir, donde cada nodo tiene a lo sumo β hijos) de profundidad máx $\{\ell(a_i)\}_{i=1}^{\alpha}$ (es decir, de máx $\{\ell(a_i)\}_{i=1}^{\alpha}$ generaciones), en el cual cada arista del árbol se etiqueta con alguno de los símbolos del alfabeto $\mathcal{B}$. De este modo, los recorridos, descritos por las secuencias ordenadas de aristas recorridas, corresponden a las palabras código de la codificación del alfabeto $\mathcal{A}$. En la interpretación de un código libre de prefijos como un árbol β-ario, cada nodo puede ser: o bien, un *nodo interno* si tiene a lo sumo β hijos y al menos uno, o bien, un *nodo hoja* si no tiene ningún hijo. Observe que, de esta manera la codificación de un símbolo siempre termina en un nodo hoja y se puede saber cuándo termina una palabra código: termina cuando avanzar por la siguiente arista, etiquetada con el símbolo a continuación en el mensaje, hace que el recorrido se descuelgue del árbol.

Si cada nodo en el árbol es, o bien, una palabra clave ó un prefijo propio de una palabra clave, entonces el código descrito por el árbol es únicamente decodificable, ya que existe un único camino desde la raíz hasta cualquier nodo del árbol y nunca pasa por más de una palabra código en el transcurso: el camino desde la raíz del árbol hasta cualquier hoja es único, por lo que dada una palabra código siempre se puede reconstruir el camino desde la raíz. Observe la Figura 2.1, donde se visualiza la interpretación de un código libre prefijos como un árbol, de forma recursiva. Observe el ejemplo de la Figura 2.2.

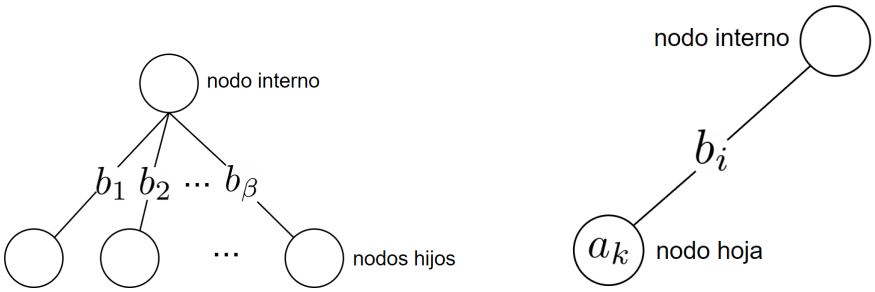

Figura 2.1: Visualización recursiva de un código libre de prefijos como un árbol β-*ario*

2.4.5. Existencia de un código libre de prefijos

Con la interpretación de la sección 2.4.4 de un código libre de prefijos, se entiende que $log_\beta(\alpha)$ puede interpretarse como la longitud fija de un código, pero también como la altura de un árbol β-ario completo con α nodos hoja, donde cada nodo hoja representa una palabra código diferente. Todo código de longitud fija con palabras código de longitud $log_\beta(\alpha)$, o mayor, es un código libre de prefijos del alfabeto $\mathcal{A}$, ya que un prefijo de una cadena que tenga su misma longitud es igual a la propia cadena. De manera general, sí se quieren asignar palabras código de longitudes distintas (código de longitud variable) y seguir implementando un código libre de prefijos, tales longitudes están restringidas por el siguiente resultado, extraído de [5]:

Teorema 2.4.1 (*Desigualdad de Kraft* (condición necesaria)[6])**.** Para cualquier código libre de prefijos C, definido sobre un alfabeto de tamaño β, las longitudes de las palabras código, $\{\ell(a_i)\}_{i=1}^{\alpha}$, cumplen que:

$$\sum_{i=1}^{\alpha} \beta^{-\ell(a_i)} \leq 1. \tag{2.9}$$

[6]El teorema 2.4.1 puede reformularse de la siguiente manera: dado un conjunto de longitudes $\{\ell(a_i)\}_{i=1}^{\alpha}$ que satisfacen la desigualdad 2.9, existe un código libre de prefijos con esas longitudes de las palabras código. La demostración es por construcción.

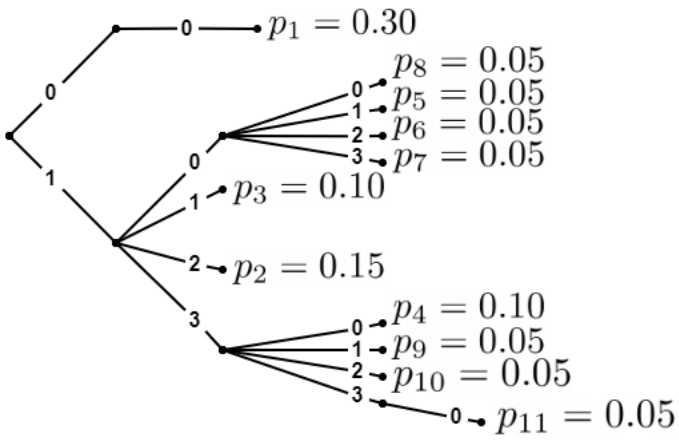

Figura 2.2: Ejemplo de la visualización de un código libre de prefijos como un árbol β-ario (no completo): el código para cada símbolo del alfabeto $\mathcal{A}$ en cada hoja del árbol se construye concatenando los símbolos del alfabeto $\mathcal{B}$ que se encuentran en el camino hacia la hoja, partiendo desde la raíz, mientras que p_i representa la probabilidad de que la fuente produzca dicho símbolo. En este ejemplo en particular, al símbolo con probabilidad p_1 se le está asignando el código 00, al que tiene probabilidad p_2 se le asocia el código 12 y así sucesivamente. Finalmente, éste código asigna a los símbolos y sus respectivas probabilidades las siguiente palabras código: $p_1 \longmapsto 00$, $p_2 \longmapsto 12$, $p_3 \longmapsto 11$, $p_4 \longmapsto 130$, $p_5 \longmapsto 101$, $p_6 \longmapsto 102$, $p_7 \longmapsto 103$, $p_8 \longmapsto 100$, $p_9 \longmapsto 131$, $p_{10} \longmapsto 132$, $p_{11} \longmapsto 133$.

Demostración. sin perdida de generalidad, suponga que las longitudes de las palabras código requeridas son asignadas de tal manera que:

$$\ell(a_1) \leq \ell(a_2) \leq \cdots \leq \ell(a_\alpha).$$

Considere el árbol β-ario completo de profundidad $\ell(a_\alpha) = \max\{\ell(a_i)\}_{i=1}^{\alpha}$; primero, tome cualquiera de los nodos del árbol que están en el nivel $\ell(a_1)$ de profundidad en el árbol y haga que este nodo corresponda a la primera palabra código de su código libre de prefijos. Desde ese momento, todos los nodos descendientes de este nodo se consideran como inadecuados para su inclusión en el código, y por tanto, no se incluyen en el código, ya que ninguna palabra código puede ser prefijo de ninguna otra palabra código. El total de nodos hojas que se excluyen del código, en esta primera iteración, es $\beta^{\ell(a_\alpha)-\ell(a_1)}$, ya que el árbol es completo.

En una siguiente iteración, tome alguno de los $\beta^{\ell(a_\alpha)-\ell(a_1)}$ nodos restantes, que aún no han sido descartados, que esté en el nivel $\ell(a_2)$ de profundidad en el árbol, como una nueva palabra código. Esta vez, se excluyen los $\beta^{\ell(a_\alpha)-\ell(a_2)}$ nodos hojas descendientes de este nodo. Continue de esta manera, asignando palabras código en los niveles de profundidad restantes: $\ell(a_3), \ell(a_4), .., \ell(a_\alpha)$. Despues de codificar los α símbolos del alfabeto $\mathcal{A}$ se habrán excluido un total de

$$\sum_{i=1}^{\alpha} \beta^{\ell(a_\alpha)-\ell(a_i)}$$

nodos hoja. No obstante, debe garantizarse que no se hayan excluido más nodos hojas que los $\beta^{\ell(a_\alpha)}$ que posee el árbol β-ario completo, es decir, se debe cumplir que

$$\sum_{i=1}^{\alpha} \beta^{\ell(a_\alpha)-\ell(a_i)} \leq \beta^{\ell(a_\alpha)}, \tag{2.10}$$

pues de lo contrario, no alcanzan los nodos hojas del árbol β-ario completo de altura $\ell(a_\alpha)$ para asignar uno a cada una de las α palabras código mediante la construcción anterior. Es decir, no se podría implementar un código libre de prefijos con las longitud de las palabras código dadas. □

A continuación, se verifica, por medio de la desigualdad de Kraft, 2.10, que siempre es posible implementar un código de longitud fija $log_\beta(\alpha)$ como un código libre de prefijos: suponga que se quiere construir un código libre de prefijos con longitudes

$$\ell(a_i) = log_\beta(\alpha) \quad \text{para } i = 1,2,\dots,\alpha.$$

Al contar el número de nodos hoja descartados del árbol, durante la construcción de este código libre de prefijos, se puede observar que

$$\sum_{i=1}^{\alpha} \beta^{\ell(a_\alpha)-\ell(a_i)} = \sum_{i=1}^{\alpha} \beta^{log_\beta(\alpha)-log_\beta(\alpha)} = \sum_{i=1}^{\alpha} \beta^0 = \sum_{i=1}^{\alpha} 1 = \alpha = \beta^{log_\beta(\alpha)} = \beta^{\ell(a_\alpha)}.$$

Por lo tanto, por la desigualdad 2.10 (también llamada *desigualdad de Kraft-McMillan*), es posible implementar el código de longitud fija $log_\beta(\alpha)$ como un código libre de prefijos, por medio de la construcción del teorema 2.4.1. Lo anterior garantiza que el código de longitud fija $log_\beta(\alpha)$ es decodificable de manera única: la manera cómo se llevaría a cabo la decodificación de este código es procesando bloques de símbolos del mensaje codificado, de la misma longitud, $log_\beta(\alpha)$: recuperando el símbolo del alfabeto $\mathcal{A}$ codificado con los $log_\beta(\alpha)$ símbolos de $\mathcal{B}$. Observe la Figura 2.3.

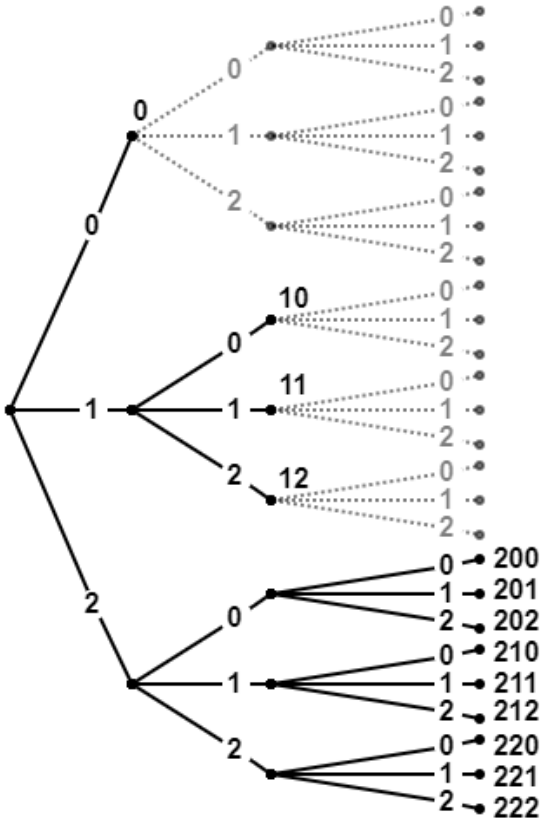

Figura 2.3: Verificación de la desigualdad de Kraft para la codificación de un alfabeto de 13 símbolos con: una palabra código de longitud 1, tres palabras código de longitud 2 y nueve palabras código de longitud 3. Gráficamente se puede ver cómo se implementaría el código, a partir de la construcción del teorema 2.4.1. También es posible comprobarlo mediante la desigualdad 2.9, ya que: $3^{-3}+3^{-2}+3^{-1} = \frac{1}{27}+\frac{1}{9}+\frac{1}{3} = \frac{13}{27} < 1$.

Cuando unas longitudes de las palabras código de entrada no satisfacen la desigualdad de Kraft, 2.4.1, es necesario ajustarlas para poder implementar un código libre de prefijos con esas longitudes: aumentar la profundidad del árbol, aumentando la longitud de algunas de las palabras código.

2.4.6. Modos de decodificación

La decodificación de la extensión de un código, formalmente, consiste en realizar la transformación inversa desde el dominio de palabras código compuestas por símbolos del alfabeto $\mathcal{B}$, hacia los símbolos del alfabeto $\mathcal{A}$. En este orden de ideas, la *decodificación de un mensaje* consiste en recuperar un mensaje escrito con el alfabeto $\mathcal{A}$, a partir de la cadena de símbolos del alfabeto $\mathcal{B}$ que se obtiene al reemplazar los símbolos del alfabeto en el mensaje por sus respectivas palabras código. Para llevar a cabo la decodificación sin ambigüedad, es indispensable saber distinguir correctamente las palabras código dentro del mensaje codificado: identificar dónde comienza y termina un nuevo símbolo del alfabeto $\mathcal{A}$. La manera cómo se logra esto es conociendo exactamente el número de símbolos de $\mathcal{B}$ que codifican un símbolo de $\mathcal{A}$: las longitudes de las palabras código.

En ese orden de ideas, la decodificación de un mensaje es más directa cuando todos los símbolos de $\mathcal{A}$ son codificados con palabras código de la misma longitud ℓ (códigos de longitud fija), ya que el procesamiento de un texto de símbolos del alfabeto $\mathcal{B}$ consiste en dividirlo en bloques de longitud ℓ, donde cada bloque codifica un símbolo del alfabeto $\mathcal{A}$, para luego usar el mapeo creado durante la codificación y recuperar los símbolos del mensaje original. Sin embargo, de manera general, las codificaciones asignan palabras código de longitudes distintas a los símbolos de $\mathcal{A}$ (códigos de longitud variable) y esto requiere algunas restricciones extras sobre las palabras código o la implementación del código: los códigos de longitud variable se emplean comunmente para *comprimir sin perdida* la codificación de un mensaje, de modo que siempre sea posible recuperar el mensaje original a partir del mensaje codificado (*comprimido*). La decodificación de un mensaje normalmente se realiza mediante la implementación de la extensión del código por medio de alguno de los siguientes mecanismos que garantizan que es posible decodificar de manera única las palabras código:

1. *Caracteres de escape.*

2. *Código libre de prefijos.*

Un *caracter de escape* es un símbolo del alfabeto $\mathcal{B}$ que no aparece en ninguna palabra código y cuyo único propósito es indicar dónde termina una palabra código. Esto hace que implementar un código de longitud variable por medio de *caracteres de escape* sea computacional-mente más costoso, ya que requiere enviar un caracter extra entre palabras. Además, se descartan un total de $\sum_{i=1}^{\alpha} \beta^{\ell(a_i)} - (\beta-1)^{\ell(a_i)}$ combinaciones, aquellas en las que aparece el caracter de escape, las cuales no podrán aprovecharse para codificar ningún símbolo del alfabeto $\mathcal{A}$. Por lo que, a veces, será necesario aumentar la longitud de las palabras código o el tamaño del alfabeto $\mathcal{B}$, para codificar de manera única todos los símbolos del alfabeto $\mathcal{A}$ mediante caracteres de escape, lo que en la práctica podría ser inviable. Ahora bien, tenga en cuenta que los caracteres de escape ayudan a decodificar un mensaje codificado únicamente cuando el código fuente del alfabeto es previamente únicamente decodificable, de lo contrario, utilizar caracteres de escape no puede garantizar la no ambigüedad.

Los *códigos libres de prefijos* pertenecen a la clase de *códigos únicamente decodificables*, que a su vez forman parte de la categoría de *códigos no singulares*. Sin embargo, a diferencia de un código únicamente decodificable, implementado mediante caracteres de escape, un código libre de prefijos permite aprovechar la propiedad libre de prefijos para realizar la decodificación del mensaje recorriéndolo una única vez, pues ninguna combinación de símbolos consecutivos es prefijo de dos símbolos distintos al mismo tiempo: el receptor podría decodificar un mensaje encontrando e interpretando, repetidamente, los prefijos que forman una palabra código válida. De manera general, esto no es posible con códigos únicamente decodificables que no son libres de prefijos, ya que el decodificador no sabría sí un prefijo constituye en sí mismo una palabra código completa o sí es simplemente el prefijo de alguna otra palabra código. Observe la figura 2.4. De hecho, siempre que un código es únicamente decodificable, aunque no sea libre de prefijos, es, en teoría, posible decodificar los mensajes que se generan con la extensión del código, independientemente del modo de decodificación. Aunque, la decodificación de un código únicamente decodificable, en general, puede requerir un preprocesamiento extra de los mensajes codificados, para identificar las palabra código que los componen.

Observe la figura 2.5.

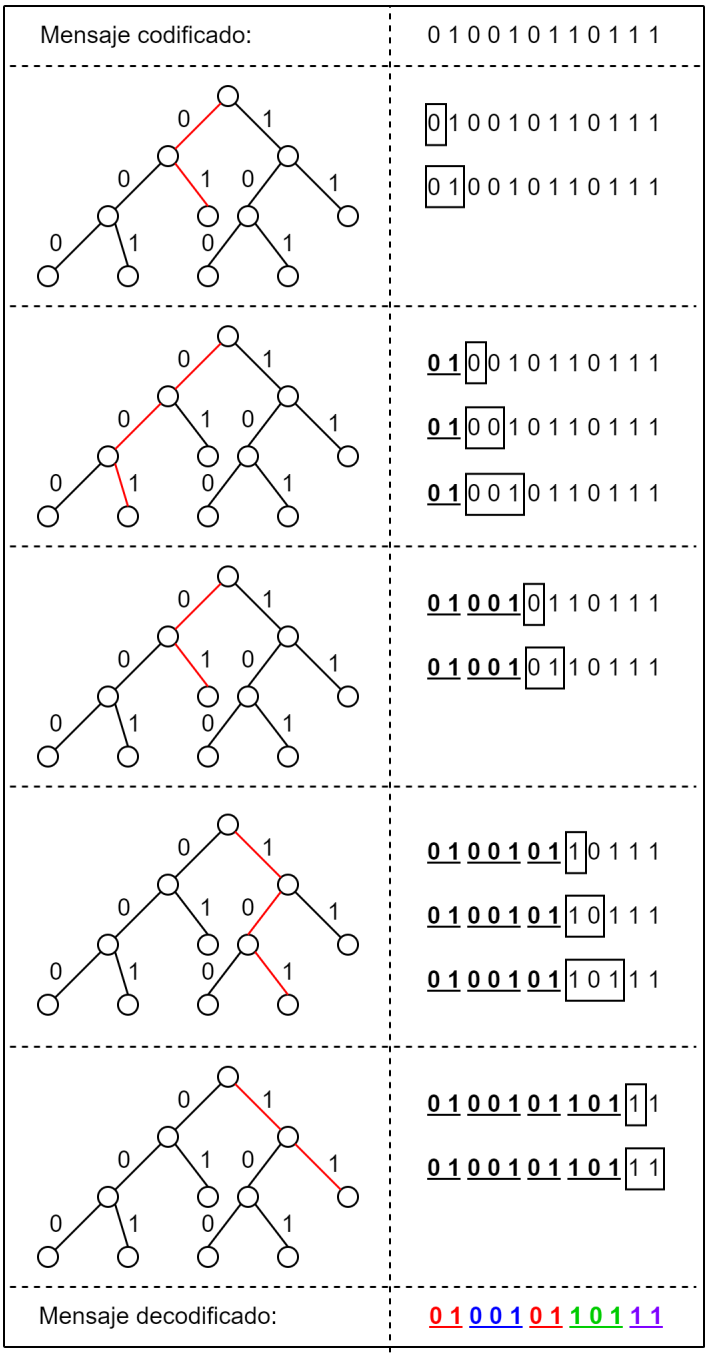

Figura 2.4: Decodificación de un mensaje codificado con un código libre de prefijos: la decodificación se lleva a cabo procesando la cadena de símbolos de $\mathcal{B}$ sobre la marcha, bajando por la arista del árbol que corresponde al siguiente símbolo en la cadena, hasta que se llega a un nodo sin hijos, en el cual bajar por ninguna arista hace que el flujo se desprenda del árbol. En ese momento se sabe que termina la palabra código actual y comienza una nueva palabra código. Esto se puede visualizar secuencialmente como los recorridos por las ramas de un árbol β-ario.

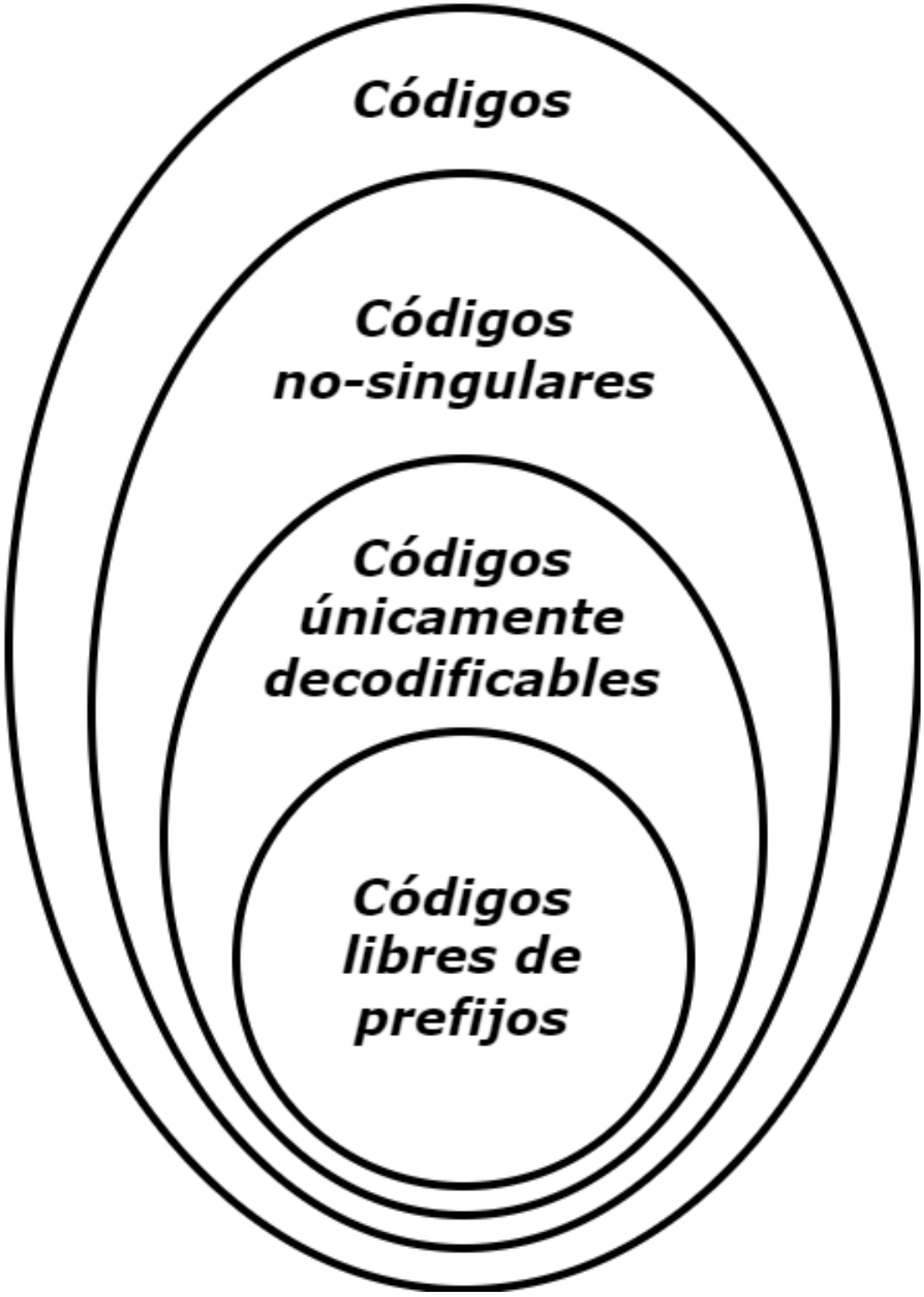

Figura 2.5: Clases de códigos y su jerarquía

2.5. Compresión sin perdida

La *compresión sin perdida* consiste en minimizar la *longitud promedio de la palabra código*, para todo código únicamente decodificable. Es posible implementar la *compresión sin perdida* por medio de un código únicamente decodificable. En esta sección, se define formalmente el concepto de *compresión sin perdida bajo conocimiento*, para el caso en el que se conoce la distribución con que una fuente X genera los símbolos, $P_X(\cdot)$, y se muestra cómo *la teoría de la información* se puede aplicar en la implementación de la *compresión sin perdida* bajo ignorancia, que es, bajo la suposición de que se desconoce la distribución $P_X(\cdot)$.

2.5.1. *Longitud promedio* de la palabra código

Dada una variable aleatoria X, con función de masa de probabilidad $P_X(x)$, que toma valores sobre un alfabeto $\mathcal{A}$; el promedio, ponderado respecto a la probabilidad de cada símbolo, de las longitudes de las palabras código de una codificación de $\mathcal{A}$ es lo que, en la literatura, se define como la *longitud promedio de la palabra código*. A continuación, se presenta la definición del concepto de *longitud promedio de la palabra código*, al igual que cómo se presenta en la referencia [21].

Definición 2.5.1 (*Longitud promedio de la palabra código*)**.** *La longitud promedio de la palabra código*, L, de un código C de una fuente X con función de masa de probabilidad $P_X(\cdot)$ se define como:

$$L(C) = \mathbb{E}[\ell(a_i)] = \sum_{i=1}^{\alpha} p(a_i)\ \ell(a_i), \qquad \text{con} \qquad \sum_{i=1}^{\alpha} p(a_i) = 1, \tag{2.11}$$

donde $p(a_i)$ denota el valor de la función de masa de probabilidad, $P_X(x = a_i)$, evaluada en el símbolo a_i, para cada $i = 1, \ldots, \alpha$ y $\ell(a_i)$ denota la longitud de la palabra código asociada al símbolo a_i en el código C.

La manera razonable de asignar las longitudes de las palabras código, para comprimir la codificación de una fuente, depende de la función de masa de probabilidad de la variable aleatoria X que modela la fuente: mientras más información se tenga sobre la distribución, más óptima puede llegar a ser la *compresión*.

2.5.2. *Compresión bajo conocimiento* de la distribución de la fuente

Comprimir la codificación de una fuente cuándo se conoce la función de masa de probabilidad de la variable aleatoria consiste en minimizar la longitud promedio de la palabra código, ecuación 2.11, mediante una escogencia apropiada de las longitudes de las palabras código, $\{\ell(a_i)\}_{i=1}^{\alpha}$, para unas probabilidades dadas, $\{p(a_i)\}_{i=1}^{\alpha}$. En general, *comprimir la codificación* de una fuente consiste en construir un código con longitud promedio de la palabra código mínima. De ahora en adelante, se entiende que un código es óptimo sí la longitud promedio de la palabra código asociada a éste es mínima. Definición adaptada de [5].

Definición 2.5.2 (*Compresión sin perdida*)**.** La *compresión sin perdida* de una fuente, se logra codificando la fuente con un código únicamente decodificable, al tiempo que se minimiza la longitud promedio de la palabra código. De ahora en adelante, la compresión sin perdida es simplemente llamada *compresión*.

Note que pueden haber muchos códigos óptimos: invertir todos los β-bits o intercambiar dos palabras código de la misma longitud de un código óptimo dará como resultado otro código óptimo. La construcción que se presenta en la sección 2.5.2.1, construye uno de esos códigos óptimos: una forma particular de representar todo código óptimo libre de prefijos de una fuente. No obstante, en la sección que le sigue, sección 2.5.2.2, se presenta otra aproximación, la cual construye *códigos semi-óptimos*. La importancia de construir estos últimos es que la existencia de los *códigos semi-óptimos* permitirá derivar algunas cotas para la compresión, lo que, finalmente, establecerá los límites de la compresión de un alfabeto, en la sección final de esta sección de compresión sin perdida, sección 2.5.4. Las demostraciones y gráficas en las siguientes secciones han sido adaptadas de la referencia [5], pero generalizadas para un alfabeto de cardinalidad β arbitraria.

2.5.2.1. Código canónico

A continuación, se demuestra que es posible representar todo código óptimo libre de prefijos en una *forma canónica*. Para esto no hace falta más que *intercambiar*, *recortar* y *reorganizar* las palabras código de un código libre de prefijos óptimo. El teorema 2.5.1, a continuación, ha sido adaptado de la referencia [5], generalizado al caso general de un β arbitrario.

Teorema 2.5.1 (*Forma canónica* de un código óptimo libre de prefijos)**.** Existe un código libre de prefijos óptimo, para una fuente X, con longitud promedio de la palabra código mínima, el cual satisface las siguientes propiedades:

1. Las longitudes de las palabras código son ordenadas de manera inversa con las probabilidades (*i.e* si $p\left(a_j\right) > p\left(a_k\right)$, entonces $l\left(a_j\right) \leq \left(a_k\right)$)

2. Las β palabras código más largas tienen la misma longitud y estas corresponden a los β símbolos menos probables.

3. Las β palabras código más largas difieren únicamente en su último β-bit.

Demostración. Sean $\{\ell(a_i)\}_{i=1}^{\alpha}$ las longitudes de las palabras código de un código libre de prefijos, de una fuente X, con longitud promedio de la palabra código mínima, $\mathbb{E}[\ell(a_i)]$, entonces:

1. Suponga que $p(a_j) > p(a_k)$ y considere un nuevo código, $\{\hat{\ell}(a_i)\}_{i=1}^{\alpha}$ con longitud promedio $\mathbb{E}[\hat{\ell}(a_i)]$, con las palabras código de los símbolos a_j y a_k intercambiadas. Entonces se tiene que

$$\begin{aligned}\mathbb{E}[\hat{\ell}(a_i)] - \mathbb{E}[\ell(a_i)] &= \sum_{i=1}^{\alpha} p(a_j)\hat{\ell}(a_j) - \sum_{i=1}^{\alpha} p(a_j)\ell(a_j) \\ &= p(a_j)\,\ell(a_k) + p(a_k)\,\ell(a_j) - p(a_j)\,\ell(a_j) - p(a_k)\,\ell(a_k) \\ &= (p(a_j) - p(a_k)) \cdot (\ell(a_k) - \ell(a_j)).\end{aligned} \tag{2.12}$$

   Sin embargo, dado que, por hipótesis, el código $\{\ell(a_i)\}_{i=1}^{\alpha}$ es óptimo, entonces $\mathbb{E}[\hat{\ell}(a_i)] - \mathbb{E}[\ell(a_i)] \geq 0$. No en vano, dado que se asumió que $p\left(a_j\right) > p\left(a_k\right)$, entonces $p\left(a_j\right) - p\left(a_k\right) > 0$. Por lo tanto, para que el signo de la multiplicación coincida debe ser que $\ell(a_k) \geq \ell(a_j)$.

2. Suponga que las β palabras código más largas no tienen la misma longitud, entonces es posible *recortar*, al menos, el último β-bit de la palabra código más larga, preservando la propiedad libre de prefijos y consiguiendo una nueva longitud promedio de la palabra código mínima. Sin embargo, esto es una contradicción, siempre que, por hipótesis, la longitud promedio de la palabra código ya era mínima. Por la propiedad 1, se sabe que las palabras código más largas corresponden a los símbolos menos probables.

3. Por la propiedad 2 se sabe que toda palabra código de longitud máxima tiene $\beta - 1$ nodos *hermanos*. Ahora bien, se pueden intercambiar las palabras código más largas para que los β símbolos de probabilidad más baja estén asociados como hermanos en el árbol. Esto último no cambiaría la longitud promedio de la palabra código. De esta manera, todo código libre de prefijos óptimo se puede transformar de modo que las palabras código más largas difieran únicamente en el último β-bit.

□

El teorema anterior, 2.5.1, demuestra que es posible transformar un código libre de prefijos óptimo de una fuente X, para que cumpla tres propiedades muy específicas. Esas tres propiedades caracterizan un nuevo tipo de código para la fuente X: *código canónico*. La definición de *código canónico* 2.5.3, que se presenta a continuación, ha sido adaptada de la referencia [5], para el caso general de un β arbitrario.

Definición 2.5.3 (*Código canónico*)**.** Un código óptimo libre de prefijos de una fuente X el cual cumple las tres propiedades del teorema 2.5.1, 1, 2 y 3, es llamado *código canónico*. En este sentido, un código canónico es la representación de un código óptimo libre de prefijos, en la cual las β palabras código más largas tienen la misma longitud, difieren únicamente en su último β-bit y corresponden a los β símbolos menos probables del alfabeto $\mathcal{A}$ de la fuente X.

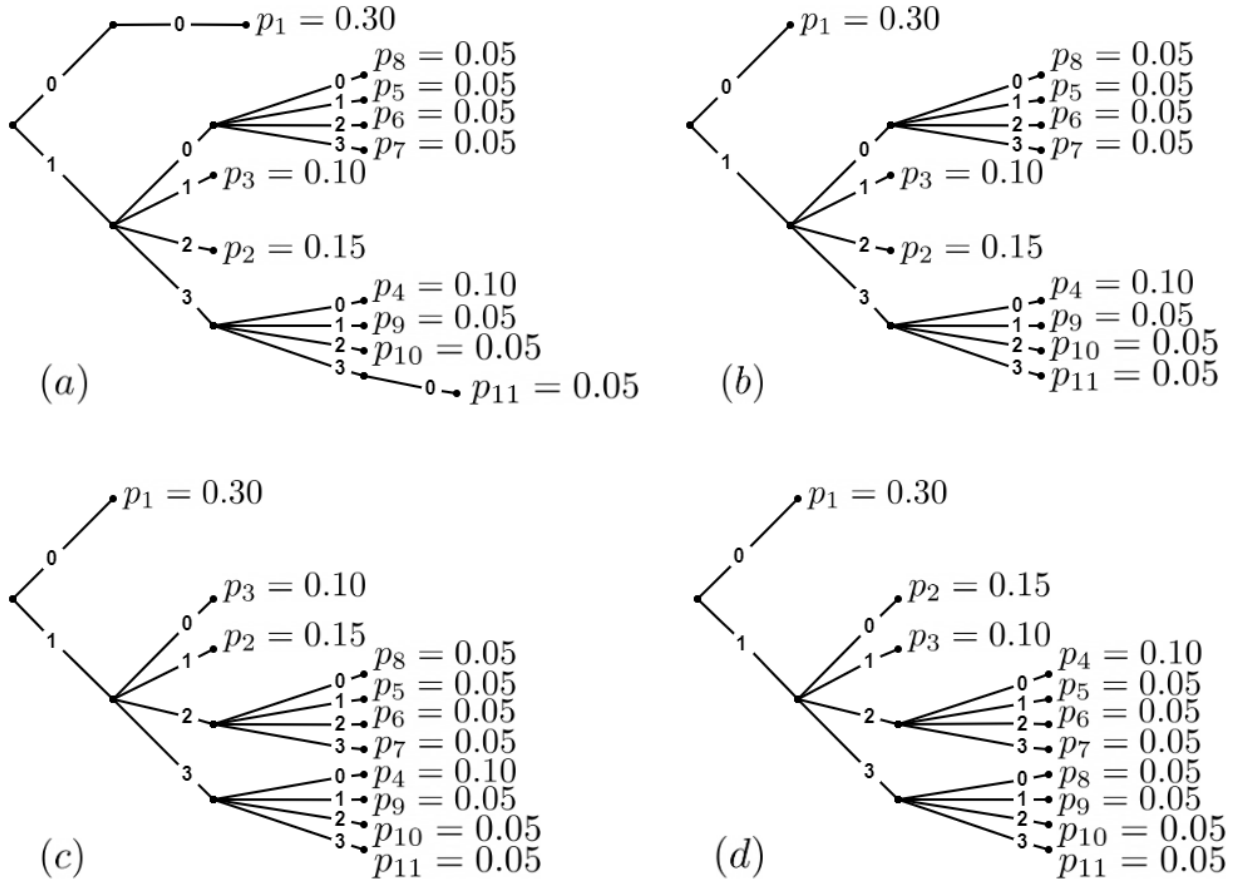

Figura 2.6: Imagen y descripción adaptada del libro [5]: ejemplificación para el caso $\beta = 4$ y $m = 11$. Dada una fuente de símbolos, X, para la cual $p_1 \geq p_2 \geq \cdots \geq p_m = p_{11}$; un posible código libre de prefijos para X se puede ver en la figura (a). Recortando aquellas ramas en las que la hoja no tiene hermanos, el código libre de prefijos se mejora (disminuye su longitud promedio de la palabra código) para obtener lo que se ve en la figura (b). A continuación, se reorganiza visualmente el árbol de tal modo que las longitudes de las palabras código quedan organizadas en orden decreciente, de arriba hacia abajo, para obtener lo que se ve en la figura (c). Finalmente, se reasignan las probabilidades a las hojas del árbol para mejorar la profundidad promedio del árbol, tal cómo se ve en la figura (d). Todo código canónico puede ser reorganizado e intercambiado a su forma canónica, parecido a lo que se muestra en la figura (d), donde $\ell_m \geq \ell_{m-1} \geq \cdots \geq \ell_2 \geq \ell_1$ y $\ell_m = \ell_{m-1}$, de modo que las $\beta = 4$ palabras código más largas difieren únicamente en su último β-bit. En efecto, el código libre de prefijos implementado como en la figura (d) es un código canónico de la fuente X.

En la figura 2.6 se ilustra cómo un código libre de prefijos óptimo puede ser transformado a forma canónica, haciendo que las β palabras código más largas difieran únicamente en el último β-bit. Ingenuamente, en el ejemplo, es fácil darse cuenta cuál es el código óptimo, sin embargo, en la práctica, determinar el código óptimo no es tan fácil, por lo que tampoco lo es determinar la forma canónica (el código canónico). En el caso más general, las propiedades 1 y 2 del teorema 2.5.1 son útiles para identificar cuándo un código libre de prefijos no es óptimo, ya que sí un código libre de prefijos no asigna las longitudes en orden opuesto a las probabilidades o las β palabras código más largas no tienen la misma longitud, entonces el código no puede ser óptimo. Esto último se sigue directamente del teorema 2.5.1. Mientras que, la propiedad 3 del teorema 2.5.1 es lo que garantiza que todo código óptimo libre de prefijos puede transformarse a forma canónica. En la ecuación 2.13, puede observar una comparación de los códigos libres de prefijos de la figura 2.6, mediante la longitud promedio de la palabra código: se puede notar que, en este ejemplo, al transformar un código libre de prefijos a su forma canónica la longitud promedio de la palabra código del código disminuye.

$$
\begin{array}{cc}
(a) & (b) \\
0{,}30 = p_1 \longmapsto 00 & 0{,}30 = p_1 \longmapsto 0 \\
0{,}15 = p_2 \longmapsto 12 & 0{,}15 = p_2 \longmapsto 12 \\
0{,}10 = p_3 \longmapsto 11 & 0{,}10 = p_3 \longmapsto 11 \\
0{,}10 = p_4 \longmapsto 130 & 0{,}10 = p_4 \longmapsto 130 \\
0{,}05 = p_5 \longmapsto 101 & 0{,}05 = p_5 \longmapsto 101 \\
0{,}05 = p_6 \longmapsto 102 & 0{,}05 = p_6 \longmapsto 102 \\
0{,}05 = p_7 \longmapsto 103 & 0{,}05 = p_7 \longmapsto 103 \\
0{,}05 = p_8 \longmapsto 100 & 0{,}05 = p_8 \longmapsto 100 \\
0{,}05 = p_9 \longmapsto 131 & 0{,}05 = p_9 \longmapsto 131 \\
0{,}05 = p_{10} \longmapsto 132 & 0{,}05 = p_{10} \longmapsto 132 \\
0{,}05 = p_{11} \longmapsto 1330 & 0{,}05 = p_{11} \longmapsto 133 \\
\text{----------} & \text{----------} \\
L = 2{,}5\ \beta\text{-bits} & L = 2{,}15\ \beta\text{-bits}
\end{array}
\tag{2.13}
$$

$$
\begin{array}{cc}
(c) & (d) \\
0{,}30 = p_1 \longmapsto 0 & 0{,}30 = p_1 \longmapsto 0 \\
0{,}15 = p_2 \longmapsto 11 & 0{,}15 = p_2 \longmapsto 10 \\
0{,}10 = p_3 \longmapsto 10 & 0{,}10 = p_3 \longmapsto 11 \\
0{,}10 = p_4 \longmapsto 130 & 0{,}10 = p_4 \longmapsto 120 \\
0{,}05 = p_5 \longmapsto 121 & 0{,}05 = p_5 \longmapsto 121 \\
0{,}05 = p_6 \longmapsto 122 & 0{,}05 = p_6 \longmapsto 122 \\
0{,}05 = p_7 \longmapsto 123 & 0{,}05 = p_7 \longmapsto 123 \\
0{,}05 = p_8 \longmapsto 120 & 0{,}05 = p_8 \longmapsto 130 \\
0{,}05 = p_9 \longmapsto 131 & 0{,}05 = p_9 \longmapsto 131 \\
0{,}05 = p_{10} \longmapsto 132 & 0{,}05 = p_{10} \longmapsto 132 \\
0{,}05 = p_{11} \longmapsto 133 & 0{,}05 = p_{11} \longmapsto 133 \\
\text{----------} & \text{----------} \\
L = 2{,}15\ \beta\text{-bits} & L = 2{,}15\ \beta\text{-bits}
\end{array}
$$

Note que, el teorema 2.5.1 no necesariamente garantiza que *canonicidad* implica longitud promedio de la palabra código mínima. En cualquier caso, en torno al concepto de canonicidad se pueden definir dos nuevas operaciones, sobre las palabras código del código canónico: *fusión* y *expansión*. Estas operaciones, como se demuestra a continuación, en los corolarios 2.5.4.1 y 2.5.4.2, conservan la óptimalidad de un código, al *expandir* o *fusionar* el alfabeto de la siguiente manera:

Fusión: En un código canónico para un alfabeto de m símbolos, C_m^*, los β símbolos con las menores probabilidades tienen palabras código de la misma longitud y difieren únicamente en el último β-bit de sus palabras código. Es posible *fusionar* las palabras código de estos símbolos en una única palabra código, eliminando el último β-bit de todas sus palabras código. El resultado de esta operación sería un código para un alfabeto de $(m-\beta+1)$ símbolos, $C_{m-\beta+1}$, donde uno de ellos es un súper-símbolo que representa los β símbolos que se fusionaron. De modo que, a partir de un código canónico C_m^* para un alfabeto de $m>0$ símbolos, caracterizado por una distribución

$$P=\{p(a_1),\ldots,p(a_{m-\beta-1}),p(a_{m-\beta}),\ldots,p(a_m)\},$$

es posible construir un código $C_{m-\beta+1}$, para una distribución de $(m-\beta+1)$ símbolos,

$$P'=\left\{p(a_1),\ldots,p(a_{m-\beta-1}),p(a_{m-\beta}),\left(\sum_{i=1}^{\beta}p(a_{m-\beta+i})\right)\right\}. \tag{2.14}$$

Observe la figura 2.7.

Expansión: A partir de un código óptimo, $C_{m-\beta+1}^*$, para la distribución

$$P'=\left\{p(a_1),\ldots,p(a_{m-\beta-1}),p(a_{m-\beta}),\left(\sum_{i=1}^{\beta}p(a_{m-\beta+i})\right)\right\}$$

es posible *expandir* este código para los m símbolos originales: tomando la palabra código que corresponde al súper-símbolo de probabilidad $\sum_{i=1}^{\beta}p(a_{m-\beta+i})$ y expandiéndolo, agregando b_0 al final de la palabra código para obtener una palabra código para el símbolo con probabilidad $p(a_\beta)$, b_1 para el símbolo con probabilidad $p(a_{\beta+1})$ y así sucesivamente: un b_i para el símbolo con probabilidad $p(a_{\beta+i})$. Hasta que se hallan construido β nuevas palabras código:

$$P=\{p(a_1),\ldots,p(a_{m-\beta-1}),p(a_{m-\beta}),\ldots,p(a_m)\}, \tag{2.15}$$

La construcción de la *expansión* y la *fusión* de los códigos se ilustra de la siguiente manera:

$C_{m-\beta+1}^*(P')$			$C_m^*(P)$		
$p(a_1)$	w_1'	ℓ_1'	$p(a_1)$	w_1	$\ell_1=\ell_1'$
$p(a_2)$	w_2'	ℓ_2'	$p(a_2)$	w_2	$\ell_2=\ell_2'$
$\vdots$			$\vdots$		
$p(a_{m-\beta-1})$	$w_{m-\beta-1}'$	$\ell_{m-\beta-1}'$	$p(a_{m-\beta-1})$	$w_{m-\beta-1}$	$\ell_{m-\beta-1}=\ell_{m-\beta-1}'$
$p(a_{m-\beta})$	$w_{m-\beta}'$	$\ell_{m-\beta}'$	$p(a_{m-\beta})$	$w_{m-\beta}$	$\ell_{m-\beta}=\ell_{m-\beta}'$
$\sum_{k=1}^{\beta}p(a_{m-\beta+k})$	$w_{m-\beta+1}'$	$\ell_{m-\beta+1}'$	$p(a_{m-\beta+1})$	$w_{m-\beta+1}$	$\ell_{m-\beta+1}=\ell_{m-\beta+1}'+1$
			$\vdots$		
			$p(a_{m-1})$	w_{m-1}	$\ell_{m-1}=\ell_{m-\beta+1}'+1$
			$p(a_m)$	w_m	$\ell_m=\ell_{m-\beta+1}'+1$

(2.16)

Lemma 2.5.2. Sí $L^*(P')$ es la longitud promedio de la palabra código mínima para el alfabeto

$$P' = \left\{ p\left(a_1\right),\ \ldots, p\left(a_{m-\beta}\right),\ \left(\sum_{i=1}^{\beta} p\left(a_{m-\beta+i}\right)\right)\right\},$$

de $(m-\beta+1)$ símbolos, entonces la *expansión* de esta codificación óptima a m símbolos, para la distribución $P = \{p\left(a_1\right), \ldots, p\left(a_{m-\beta}\right), p\left(a_{m-\beta+1}\right) \ldots, p\left(a_m\right)\}$, tiene longitud promedio de la palabra código

$$L\left(P\right) = L^*\left(P'\right) + \sum_{i=1}^{\beta} p\left(a_{m-\beta+i}\right). \tag{2.17}$$

Demostración. Sean $\{\ell'(a_i)\}_{i=1}^{m-\beta+1}$ las longitudes de las palabras código de un código libre de prefijos con longitud promedio de la palabra código mínima, $L^*\left(P'\right)$, para el alfabeto de $(m-\beta+1)$ símbolos,

$$P' = \left\{ p\left(a_1\right),\ \ldots, p\left(a_{m-\beta}\right),\ \left(\sum_{i=1}^{\beta} p\left(a_{m-\beta+i}\right)\right)\right\},$$

entonces

$$\begin{aligned}
L\left(P\right) &= \left(\sum_{i=1}^{m} p\left(a_i\right)\ \ell\left(a_i\right)\right) \\
&= \left(\sum_{i=1}^{m-\beta} p\left(a_i\right)\ \ell\left(a_i\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right) \ell\left(a_j\right)\right) \\
&= \left(\sum_{i=1}^{m-\beta} p\left(a_i\right)\ \ell'\left(a_i\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right) \left[\ell'\left(a_{m-\beta+1}\right)+1\right]\right) \\
&= \left(\sum_{i=1}^{m-\beta} p\left(a_i\right) \ell'\left(a_i\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right)\ \ell'\left(a_{m-\beta+1}\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right)\right) \\
&= \left(\sum_{i=1}^{m-\beta} p\left(a_i\right) \ell'\left(a_i\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right)\ \ell'\left(a_{m-\beta+1}\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right)\right) \\
&= \left(\sum_{i=1}^{m-\beta} p\left(a_i\right) \ell'\left(a_i\right)\right) + \left(\ell'\left(a_{m-\beta+1}\right) \cdot \sum_{j=m-\beta+1}^{m} p\left(a_j\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right)\right) \\
&= \left(\sum_{i=1}^{m-\beta} p\left(a_i\right) \ell'\left(a_i\right)\right) + \left(\ell'\left(a_{m-\beta+1}\right) \cdot p\left(a_{m-\beta+1}\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right)\right) \\
&= \left(\sum_{i=1}^{m-\beta+1} p\left(a_i\right) \ell'\left(a_i\right)\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right)\right) \\
&= L^*\left(P'\right) + \left(\sum_{j=m-\beta+1}^{m} p\left(a_j\right)\right) \quad \text{(Cambio de variable } j = m-\beta+i\text{)}
\end{aligned} \tag{2.18}$$

es, por construcción, la longitud promedio de la palabra código para el *alfabeto expansión* de m símbolos,

$$P = \{p\left(a_1\right), \ldots, p\left(a_{m-\beta}\right), p\left(a_{m-\beta+1}\right), \ldots, p\left(a_m\right)\}.$$

□

Lemma 2.5.3. Sí $L^*(P)$ es la longitud promedio de la palabra código mínima para el alfabeto de m símbolos, $P = \{p(a_1), \ldots, p(a_{m-\beta}), p(a_{m-\beta+1}), \ldots, p(a_m)\}$, entonces la *fusión* de esta codificación óptima a la distribución

$$P' = \left\{ p(a_1), \ \ldots, p(a_{m-\beta}), \ \left(\sum_{i=1}^{\beta} p(a_{m-\beta+i}) \right) \right\},$$

de $(m - \beta + 1)$ símbolos, tiene longitud promedio de la palabra código

$$L(P') = L^*(P) - \sum_{i=1}^{\beta} p(a_{m-\beta+i}). \tag{2.19}$$

Demostración. Sean $\{\ell(a_i)\}_{i=1}^m$ las longitudes de las palabras código de un código libre de prefijos con longitud promedio de la palabra código mínima, $L^*(P)$, para el alfabeto de m símbolos,

$$P = \{p(a_1), \ldots, p(a_{m-\beta}), p(a_{m-\beta+1}), \ldots, p(a_m)\},$$

entonces

$$\begin{aligned}
L(P') &= \left(\sum_{i=1}^{m-\beta+1} p(a_i)\, \ell'(a_i) \right) \\
&= \left(\sum_{i=1}^{m-\beta} p(a_i)\, \ell'(a_i) \right) + p(a_{m-\beta+1})\, \ell'(a_{m-\beta+1}) \\
&= \left(\sum_{i=1}^{m-\beta} p(a_i)\, \ell'(a_i) \right) + \left(\sum_{j=m-\beta+1}^{m} p(a_j) \right) \ell'(a_{m-\beta+1}) \\
&= \left(\sum_{i=1}^{m-\beta} p(a_i)\, \ell'(a_i) \right) + \left(\sum_{j=m-\beta+1}^{m} p(a_j)\, \ell'(a_{m-\beta+1}) \right) \\
&= \left(\sum_{i=1}^{m-\beta} p(a_i)\, \ell'(a_i) \right) + \left(\sum_{j=m-\beta+1}^{m} p(a_j) [\ell(a_j) - 1] \right) \\
&= \left(\sum_{i=1}^{m-\beta} p(a_i)\, \ell(a_i) \right) + \left(\sum_{j=m-\beta+1}^{m} p(a_j) [\ell(a_j) - 1] \right) \\
&= \left(\sum_{i=1}^{m-\beta} p(a_i)\, \ell(a_i) \right) + \left(\sum_{j=m-\beta+1}^{m} p(a_j)\, \ell(a_j) \right) - \left(\sum_{j=m-\beta+1}^{m} p(a_j) \right) \\
&= \left(\sum_{k=1}^{m} p(a_k)\, \ell(a_k) \right) - \left(\sum_{j=m-\beta+1}^{m} p(a_j) \right) \\
&= L^*(P) - \left(\sum_{j=m-\beta+1}^{m} p(a_j) \right) \quad \text{(Cambio de variable } j = m - \beta + i\text{)}
\end{aligned} \tag{2.20}$$

es, por construcción, la longitud promedio de la palabra código para el *alfabeto fusión* de $(m - \beta + 1)$ símbolos,

$$P' = \left\{ p(a_1), \ \ldots, p(a_{m-\beta}), \ \left(\sum_{i=1}^{\beta} p(a_{m-\beta+i}) \right) \right\}.$$

□

Teorema 2.5.4. Sean $L(P')$ y $L(P)$ las longitudes promedio de las palabras código óptimas para los alfabetos de longitudes $(m-\beta+1)$ y m, respectivamente. Y sean $L^*(P')$ y $L^*(P)$ las longitudes promedio de las palabras código óptimas para los *alfabetos fusión y expansión*, respectivamente. Se cumple que:

$$(L(P') - L^*(P')) + (L(P) - L^*(P)) = 0. \tag{2.21}$$

Demostración. Sean $L^*(P')$ y $L^*(P)$ las longitudes promedio de las palabras código óptimas para los alfabetos de longitudes $(m-\beta+1)$ y m, respectivamente. Por los lemas 2.5.2 y 2.5.3 se sabe que:

$$L(P') = \left[L^*(P) - \left(\sum_{j=m-\beta+1}^{m} p(a_j)\right)\right] \qquad y \qquad L(P) = \left[L^*(P') + \left(\sum_{j=m-\beta+1}^{m} p(a_j)\right)\right],$$

donde $L(P)$ denota la longitud promedio de la palabra código del *alfabeto expansión* y $L(P')$ denota la longitud promedio de la palabra código del *alfabeto fusión*. Por lo cual

$$L(P') + L(P) = \left[L^*(P) - \left(\sum_{j=m-\beta+1}^{m} p(a_j)\right)\right] + \left[L^*(P') + \left(\sum_{j=m-\beta+1}^{m} p(a_j)\right)\right] = L^*(P) + L^*(P') \tag{2.22}$$

□

Corolario 2.5.4.1. La expansión de la codificación óptima para la distribución P' tiene longitud promedio de la palabra código óptima:

$$L(P) = L^*(P). \tag{2.23}$$

Demostración. Por definición de longitud promedio de la palabra código óptima, se tiene que

$$L(P) - L^*(P) \geq 0. \tag{2.24}$$

Sin embargo, dado que la cantidad de la ecuación 2.22 es igual a cero, cuando la única forma en la que la suma de dos cantidades no negativas es cero es cuando ambas cantidades son iguales a cero, entonces

$$L(P) - L^*(P) = 0. \tag{2.25}$$

□

Corolario 2.5.4.2. La fusión de la codificación canónica para la distribución P tiene longitud promedio de la palabra código óptima:

$$L(P') = L^*(P'). \tag{2.26}$$

Demostración. Por definición de longitud promedio de la palabra código óptima, se tiene que

$$L(P') - L^*(P') \geq 0. \tag{2.27}$$

Sin embargo, dado que la cantidad en la ecuación 2.22 es igual a cero, cuando la única forma en la que la suma de dos cantidades no negativas es cero es cuando ambas cantidades son iguales a cero, entonces

$$L(P') - L^*(P') = 0. \tag{2.28}$$

□

Por lo tanto, sí se parte de un código óptimo, $C^*_{m-\beta+1}$, para un alfabeto de $(m-\beta+1)$ símbolos, con probabilidades

$$P' = \left\{ p(a_1), \ldots, p(a_{m-\beta}), \left(\sum_{i=1}^{\beta} p(a_{m-\beta+i}) \right) \right\},$$

entonces es posible construir un código óptimo, C^*_m, para m símbolos, con probabilidades $P = \{p(a_1), \ldots, p(a_m)\}$, expandiendo la palabra código con probabilidad $\sum_{i=m-\beta+1}^{m} p(a_i)$, de modo que el nuevo código para P también es óptimo: empezando con un código para un *alfabeto fusión* de β símbolos, para el cual la codificación óptima es obvia, entonces haciendo inducción sobre m, y *expansión* de la codificación en cada paso, se puede construir un código óptimo para cualquier alfabeto. Esto sugiere que todo código canónico para un alfabeto de m símbolos se puede pensar como proveniente de la *expansión* del código óptimo de su respectivo *alfabeto fusión* de $(m-\beta+1)$ símbolos, ya que los β símbolos más improbables tienen palabras código de la misma longitud y por construcción difieren únicamente en su último β-bit. Así que, se puede alcanzar el código libre prefijos con longitud promedio de la palabra código mínima, conservando la canonicidad del código mediante las operaciones de *fusión* y *expansión*. Para el código resultante, no será necesario verificar la desigualdad de Kraft, 2.4.1, ya que al conservar la canonicidad, también se conserva la propiedad libre de prefijos.

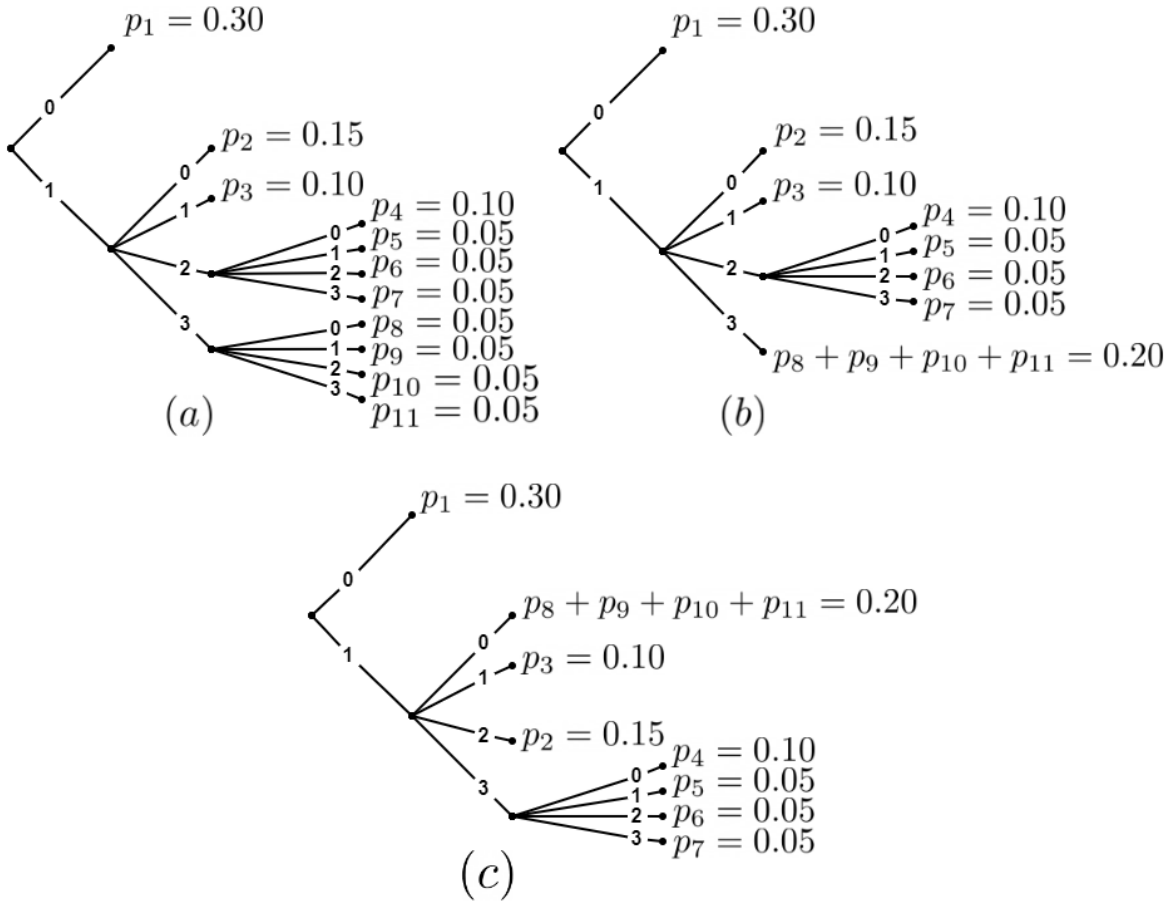

Figura 2.7: Paso inductivo por *fusión* y/o *expansión*: dados $p_1 \geq p_2 \geq \cdots \geq p_m = p_{11}$, un código canónico para los $m = 11$ símbolos se ilustra en la figura (a). Luego, fusionando los $\beta = 4$ símbolos más improbables se obtiene el código óptimo, pero desordenado, de la figura (b). Finalmente, se pueden reordenar las probabilidades en orden decreciente para obtener el código canónico para $m - \beta + 1 = 8$ símbolos que se puede ver en la figura (c).

2.5.2.1.1 *Codificación de Huffman*

Los resultados de los corolarios 2.5.4.1 y 2.5.4.2 sugieren la existencia de un algoritmo de codificación recursivo, basado en las operaciones de expansión y fusión: este algoritmo iterará hasta un caso base, en el que la codificación sea óptima, de modo que en cada paso de vuelta satisfaga las hipótesis de los lemas 2.5.2 y 2.5.3. Así se garantiza la canonicidad del código resultante, el cual es un llamado un *código de Huffman* [7].

En el Apéndice B, se presenta detalladamente la implementación de la *codificación de Huffman* mediante una función `huffman_coding(P, B)`. Este algoritmo ha sido adaptado de la referencia [5] para un β arbitrario. A continuación, se enuncia el resultado principal de esta sección, el cual fue tomado de la referencia [5].

Corolario 2.5.4.3 (Óptimalidad de los códigos de Huffman)**.** La codificación de Huffman de una fuente X es óptima. Es decir, sí C^* es una codificación de Huffman para una fuente X, con alfabeto $\mathcal{A}$, y C es cualquier otra codificación únicamente decodificable de X, entonces $L(C^*) \leq L(C)$.

Demostración. La codificación de Huffman codifica óptimamente un alfabeto de β símbolos en el caso base y luego se asegura de conservar la óptimalidad del código del caso base, habiendo construido de manera canónica la fusión del paso anterior, de modo que la extensión de la óptimalidad del código para $(m - \beta + 1)$ símbolos a m símbolos está garantizada por los corolarios 2.5.4.1 y 2.5.4.2, a partir de la construcción. Por esta razón, el árbol libre de prefijos resultante implementa el código canónico del alfabeto para m símbolos. Luego haciendo inducción sobre m se sigue que la codificación de Huffman es óptima para cualquier alfabeto. □

Observe que, el algoritmo de codificación de Huffman es *voráz*, en el sentido de que trata de alcanzar la óptimalidad inmediata (local) en cada paso, ubicando los β símbolos menos probables en los niveles más profundos del árbol actual. Sin embargo, esta óptimalidad local es suficiente para garantizar la óptimalidad global, por medio de la canonicidad de los códigos fusión y expansión, como demuestra el corolario 2.5.4.3.

Ahora bien, sabiendo que el código canónico es óptimo, se pudo demostrar que la codificación de Huffman es óptima, para cualquier distribución de probabilidad de la fuente. Aún así, ninguno de los resultados presentados hasta ahora permite afirmar nada acerca de las limitaciones de la implementación: se sabe cómo construir la codificación óptima para un alfabeto y, de ahí, la longitud promedio mínima para cualquier código libre de prefijos, pero aún no se sabe qué tan global es el óptimo de la implementación, en el sentido de que no se conoce qué tan lejos del verdadero óptimo se encuentra el óptimo de la implementación. En definitiva, se conoce el óptimo de la implementación, el cual es la longitud promedio del código de Huffman, pero no el límite de compresión en sí, la longitud promedio de la palabra código mínima.

Debido a que la codificación de Huffman es óptima y que en ese sentido la longitud promedio de la palabra código asociada a este es menor o igual que la longitud promedio de cualquier otro código libre de prefijos, se sabe que cualquier cota superior para la longitud promedio de cualquier otra codificación libre de prefijos también acota superiormente la longitud promedio de la codificación de Huffman. Sabiendo esto, se puede intentar hallar la menor cota superior posible para determinar qué tan tanto es lo máximo que la implementación se puede llegar a alejar del límite verdadero. Con esto, luego se pueden intentar controlar aquellos factores que ocasionan que los óptimos de la implementación difieran de los óptimos teóricos: codificando bloques de símbolos, para eliminar excesos por redondeo, o estimando de manera más precisa la distribución a priori de los símbolos del alfabeto. Otra forma de verlo, es que la codificación de Huffman, por ser óptima, puede servir para estimar el verdadero límite de compresión. Una vez se deriven las cotas para la longitud promedio de la palabra código de los *códigos semi-óptimos*, en la sección 2.5.2.2, se sabrá qué tan buena es la estimación y si, en efecto, se puede medir razonablemente el límite de compresión mediante códigos óptimos.

[7] *David A. Huffman* fue un ingenierio estadounidense que vivió durante el siglo veinte.

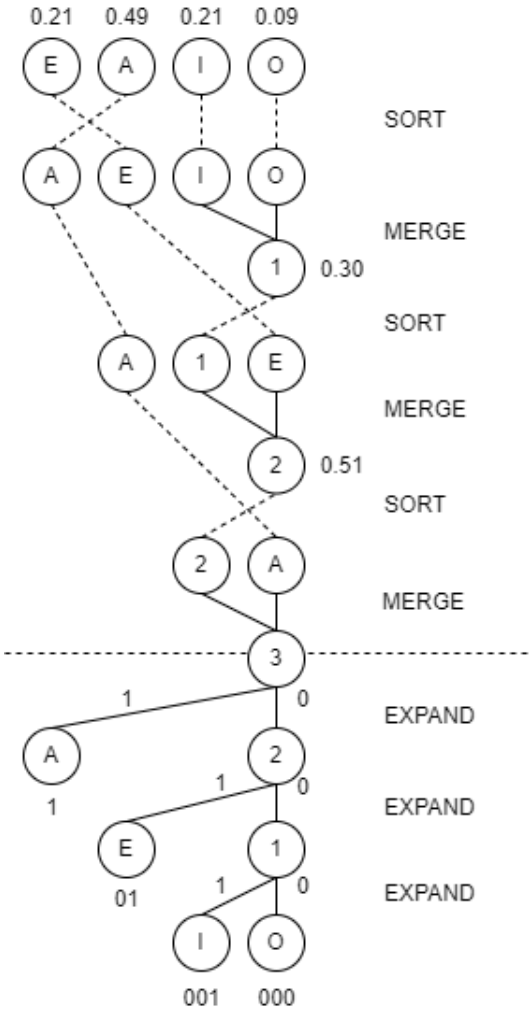

Figura 2.8: Ejemplo de la codificación de Huffman, para un alfabeto de cuatro símbolos no-equiprobables, en un alfabeto de $\beta = 2$ símbolos.

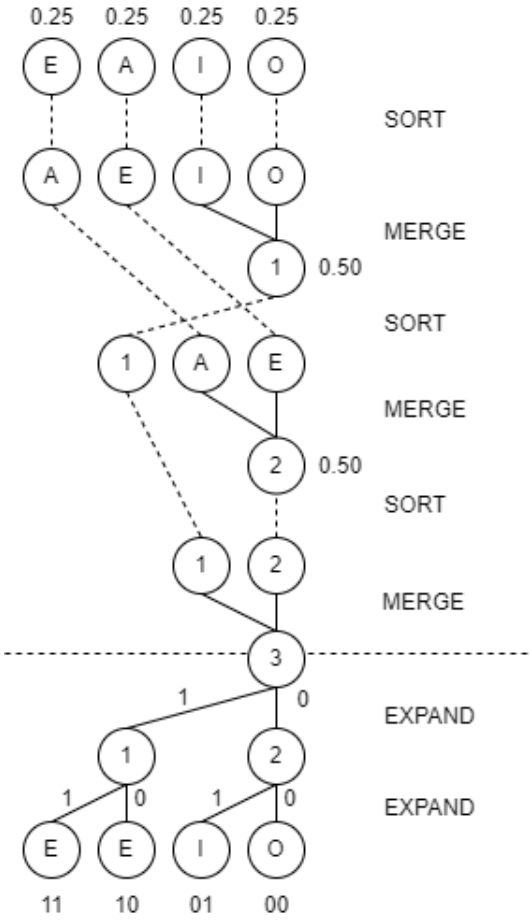

Figura 2.9: Ejemplo de la codificación de Huffman, para un alfabeto de cuatro símbolos equiprobables, en un alfabeto de $\beta = 2$ símbolos.

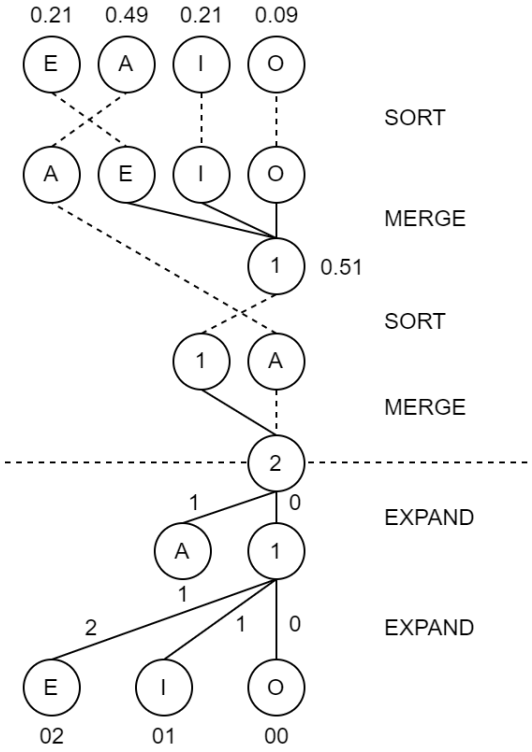

Figura 2.10: Ejemplo de la codificación de Huffman, para un alfabeto de cuatro símbolos no-equiprobables, en un alfabeto de $\beta = 3$ símbolos.

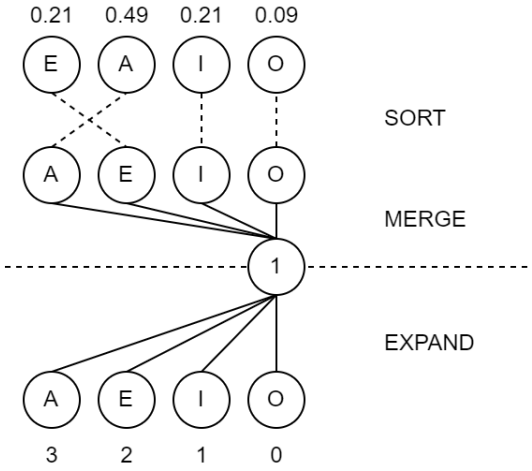

Figura 2.11: Ejemplo de la codificación de Huffman, para un alfabeto de cuatro símbolos equiprobables, en un alfabeto de $\beta = 4$ símbolos.

2.5.2.2. *Código semi-óptimo*

Todo código libre de prefijos es un código únicamente decodificable. Ya se sabe que cuando se conocen la función de masa de probabilidad de la fuente para cada símbolo del alfabeto $\mathcal{A}$, se puede construir un código únicamente decodificable (libre de prefijos) óptimo: la codificación de Huffman.

La óptimalidad de la codificación de Huffman está garantizada por su construcción, lo cual se demostró en el corolario 2.5.4.3. Sin embargo, cuando se determina el mínimo de la longitud promedio de la palabra código, como una función de las longitudes de las palabras código:

$$L[\ell(a_1), \ell(a_2), \ldots, \ell(a_\alpha)] = \left(\sum_{i=1}^{\alpha} p(a_i) \cdot \ell(a_i) \right), \tag{2.29}$$

el resultado son las longitudes de las palabras código para las cuales el valor de la longitud promedio de la palabra código es mínimo. Aún así, aunque se conozcan las longitudes de las palabras código óptimas, la implementación de estas (para un código libre de prefijos) hace que, en promedio, la longitud promedio de la palabra código no pueda ser más óptima que aquella de la codificación de Huffman. Por esta razón, a este tipo de código se le llama *código semi-óptimo*. El teorema 2.5.5, a continuación, demuestra, precisamente, que el proceso de minimización arroja valores para las longitudes óptimas que pueden no ser números enteros, y por tanto, que deben ser aproximados al momento de hacer una implementación. Esta es la razón por la que la longitud promedio de un mensaje codificado de esta manera es siempre mayor o igual que el obtenido por medio de la codificación de Huffman. Este resultado (y su demostración) fue tomado de la referencia [5].

Teorema 2.5.5 (Longitud promedio de la palabra código mínima)**.** Las longitudes de las palabras código de un código libre de prefijos de una fuente X, que minimizan la longitud promedio de la palabra código son:

$$\ell^* (a_i) = log_\beta \left(\frac{1}{p(a_i)} \right). \tag{2.30}$$

Demostración. Para obtener unas longitudes de las palabras código que implementen un código libre de prefijos, se debe condicionar la solución a solo aquellas palabras código que satisfagan la desigualdad de Kraft-McMillan, 2.10. La desigualdad de Kraft-McMillan establece una condición necesaria, sobre las longitudes de las palabras código del código, que debe ser satisfecha para la implementación del código como un código libre de prefijos: que el número de nodos hojas descartados durante la construcción del código, para mantener la propiedad libre de prefijos, no exceda el máximo número de nodos disponibles o, dicho de otra forma;

$$\sum_{i=1}^{\alpha} \beta^{-\ell(a_i)} \leq 1.$$

El problema de optimización de la longitud promedio de la palabra código consiste en: minimizar la longitud promedio de la palabra código, en función de las longitudes de las palabras código, sujetas a la condición de implementar un código libre de prefijos: determinar

$$\min_{\{\ell(a_i)\}_{i=1}^{\alpha}} L[\ell(a_1), \ell(a_2), \ldots, \ell(a_\alpha)],$$

sujeto a las siguientes restricciones sobre longitudes enteras de las palabras código:

$$\sum_{i=1}^{\alpha} \beta^{-\ell(a_i)} \leq 1. \qquad \ell_i \geq 0 \ \text{ para todo } \ i = 1,2,\ldots,\alpha \tag{2.31}$$

Para abordar este problema de optimización se puede emplear el método *KKT* (Karush-Kuhn-Tucker), el cual generaliza el método de los *multiplicadores de Lagrange*. El mínimo factible se obtiene al desactivar las

restricciones sobre la longitud de las palabras código, manteniendo activa únicamente la restricción dada por la ecuación de Kraft ([26]). Después de aplicar lo anterior, se obtiene que la expresión a minimizar es:

$$J[\ell(a_1),\dots,\ell(a_\alpha)\,;\,\lambda] = \left(\sum_{i=1}^{\alpha} p(a_i)\cdot\ell(a_i)\right) + \lambda\left(\sum_{i=1}^{\alpha}\beta^{-\ell(a_i)} - 1\right). \tag{2.32}$$

A continuación, se calculan las derivadas parciales de la función J, respecto a $\ell(a_1),\dots,\ell(a_\alpha)$, y se igualan a cero, para obtener el siguiente sistema de ecuaciones:

$$\frac{\partial J}{\partial\,\ell(a_i)} = p(a_i) - \lambda\,\beta^{-\ell(a_i)}\,ln(\beta) = 0 \qquad para \qquad i = 1,2,\dots,\alpha. \tag{2.33}$$

Despejando a $\beta^{-\ell(a_i)}$ de la desigualdad 2.33, para cada $i = 1,2,\dots,\alpha$, se obtiene que

$$\beta^{-\ell(a_i)} = \frac{p(a_i)}{\lambda\,ln(\beta)} \qquad para \qquad i = 1,2,\dots,\alpha. \tag{2.34}$$

Ahora bien, observe que sí se reemplaza el valor de $\beta^{-\ell(a_i)}$, de la ecuación 2.34, en la ecuación 2.31, para cada $i = 1,2,\dots,\alpha$, entonces se sigue que

$$\sum_{i=1}^{\alpha}\beta^{-\ell(a_i)} = \sum_{i=1}^{\alpha}\frac{p(a_i)}{\lambda\,ln(\beta)} = 1. \tag{2.35}$$

No obstante, se puede reemplazar el número 1 por la suma total de las probabilidades en la ecuación 2.35, de modo que

$$\sum_{i=1}^{\alpha}\frac{p(a_i)}{\lambda\,ln(\beta)} = \sum_{i=1}^{\alpha} p\,(a_i)\,, \tag{2.36}$$

de donde se sigue que en valor de λ es

$$\lambda = \frac{1}{ln\,(\beta)}. \tag{2.37}$$

Así que, la igualdad 2.34 se puede reescribir como

$$\beta^{-\ell(a_i)} = \frac{p\,(a_i)}{\lambda ln\,(\beta)} = \frac{p\,(a_i)}{\left(\frac{1}{ln(\beta)}\right)ln\,(\beta)} = p\,(a_i)\,. \tag{2.38}$$

Despejando las longitudes de las palabras código óptimas de la ecuación 2.38, se obtienen las siguientes longitudes de las palabras código como óptimas para minimizar la longitud promedio de la palabra código:

$$\ell^*(a_i) = log_\beta\left(\frac{1}{p(a_i)}\right) \qquad para \qquad i = 1,2,\dots,\alpha. \tag{2.39}$$

□

Para las longitudes de las palabras código óptimas, ecuación 2.39, la longitud promedio de la palabra código es la longitud promedio de la palabra código mínima para todo código libre de prefijos:

$$\sum_{i=1}^{\alpha} p\,(a_i)\,log_\beta\left(\frac{1}{p\,(a_i)}\right) \tag{2.40}$$

Es posible notar que, la manera cómo se minimiza la longitud promedio de la palabra código de un código de longitud variable (y, por tanto, de longitud fija), es haciendo que mientras más probable sea un símbolo, más corta sea la longitud de su palabra código. Esto ya se veia cuándo la codificación de Huffman asignaba a los niveles más profundos de su árbol libre prefijos los símbolos más improbables. Sin embargo, también se puede notar que las longitudes de las palabras código óptimas no necesariamente son números enteros, por lo que no siempre se podrá implementar un código con estas longitudes. En las secciones 2.5.2.2.1 y 2.5.2.2.2 se presentan dos implementaciones particulares de código semi-óptimo y se derivan dos cotas superiores para su longitud promedio de la palabra código, las cuales también se cumplen para la codificación de Huffman.

2.5.2.2.1 *Codificación de Shannon*

Cuando se conocen las probabilidades de aparición de los símbolos del alfabeto $\mathcal{A}$, se puede minimizar la longitud promedio de la palabra código, definiendo las longitudes de las palabras código como:

$$\ell(a_i) = log_\beta\left(\frac{1}{p(a_i)}\right) \qquad para \qquad i = 1,2,\dots,\alpha. \tag{2.41}$$

Para esta escogencia particular de las longitudes de las palabras código, la longitud promedio de la palabra código es

$$\sum_{i=1}^{\alpha} p(a_i)\cdot log_\beta\left(\frac{1}{p(a_i)}\right), \tag{2.42}$$

el valor mínimo de la longitud promedio de la palabra código para todo código libre de prefijos. Es posible verificar que, en efecto, la escogencia de las longitudes de las palabras código de la ecuación 2.41 implementa un código libre de prefijos, ya que

$$\begin{aligned}
\sum_{i=1}^{\alpha} \beta^{\ell(a_\alpha)-\ell(a_i)} &= \sum_{i=1}^{\alpha} \beta^{log_\beta(1/p(a_\alpha))-log_\beta(1/p(a_i))} \\
&= \sum_{i=1}^{\alpha} \beta^{log_\beta(p(a_i)/p(a_\alpha))} \\
&= \sum_{i=1}^{\alpha} \frac{p(a_i)}{p(a_\alpha)} \\
&= \frac{1}{p(a_\alpha)}\sum_{i=1}^{\alpha} p(a_i) \\
&\leq \frac{1}{p(a_\alpha)} = \beta^{log_\beta(1/p(a_\alpha))} = \beta^{\ell(a_\alpha)}.
\end{aligned} \tag{2.43}$$

Existen procedimientos capaces de construir un código libre de prefijos para, aproximadamente, las longitudes de las palabras código de la ecuación 2.41. Uno de estos procedimientos es conocido como el *algoritmo de codificación de Shannon*[8], el cual se explica a continuación.

El algoritmo de *codificación de Shannon*, así como otros, requiere longitudes de las palabras código números enteros, sin embargo, note que las longitudes de las palabras código de la ecuación 2.41 podrían no ser un número entero, así que se asigna el entero, mayor, más cercano:

$$\hat{\ell}(a_i) = \left\lceil log_\beta\left(\frac{1}{p(a_i)}\right)\right\rceil \qquad para \qquad i = 1,2,\dots,\alpha. \tag{2.44}$$

Para comprobar que estas nuevas longitudes de las palabras código también implementan un código libre de prefijos, basta con notar que

$$log_\beta\left(\frac{1}{p(a_i)}\right) \leq \left\lceil log_\beta\left(\frac{1}{p(a_i)}\right)\right\rceil \qquad para \qquad i = 1,2,\dots,\alpha, \tag{2.45}$$

de donde se sigue que

$$\beta^{-log_\beta(1/p(a_i))} \geq \beta^{-\lceil log_\beta(1/p(a_i))\rceil} \qquad para \qquad i = 1,2,\dots,\alpha. \tag{2.46}$$

Así que, por desigualdad 2.43, se cumple que

$$\sum_{i=1}^{\alpha} \beta^{log_\beta(1/p(a_\alpha))-\lceil log_\beta(1/p(a_i))\rceil} \leq \sum_{i=1}^{\alpha} \beta^{log_\beta(1/p(a_\alpha))-log_\beta(1/p(a_i))} \leq \beta^{\ell(a_\alpha)} \leq \beta^{\hat{\ell}(a_\alpha)}. \tag{2.47}$$

[8] *Claude E. Shannon fue un ingeniero y criptógrafo estadounidense que vivió durante el siglo veinte.*

Las implementaciones de la codificación de Shannnon, por definición, garantizan que la longitud promedio de la palabra código, del código resultante, siempre está, a lo sumo, a un β-bit de diferencia de la longitud promedio de la palabra código óptima: dado que

$$\hat{\ell}(a_i) = \left\lceil log_\beta \left(\frac{1}{p(a_i)} \right) \right\rceil \leq \left(log_\beta \left(\frac{1}{p(a_i)} \right) \right) + 1 \qquad para \qquad i = 1,2,\ldots,\alpha,$$

entonces se sigue que

$$\begin{aligned} \sum_{i=1}^{\alpha} p(a_i) \cdot \hat{\ell}(a_i) &= \sum_{i=1}^{\alpha} p(a_i) \cdot \left\lceil log_\beta \left(\frac{1}{p(a_i)} \right) \right\rceil \\ &\leq \sum_{i=1}^{\alpha} p(a_i) \cdot \left[\left(log_\beta \left(\frac{1}{p(a_i)} \right) \right) + 1 \right] \\ &= \left[\sum_{i=1}^{\alpha} p(a_i) \cdot \left(log_\beta \left(\frac{1}{p(a_i)} \right) \right) \right] + \left[\sum_{i=1}^{\alpha} p(a_i) \right] \\ &= \left[\sum_{i=1}^{\alpha} p(a_i) \cdot \left(log_\beta \left(\frac{1}{p(a_i)} \right) \right) \right] + 1 \end{aligned}$$

De manera general, la aproximación de Shannon asigna las palabras código en orden de los símbolos más probables a los menos probables, eligiendo cada palabra código como la primera palabra, lexicográficamente, de la longitud correcta que mantiene la propiedad libre de prefijos.

Por ejemplo, el siguiente algoritmo, que hace uso de *frecuencias acumuladas*, es una implementación particular de la codificación de Shannon, para $\beta = 2$:

1. Ordenar los símbolos del alfabeto de acuerdo a su probabilidad, en orden decreciente, sin perdida de generalidad, tal que:
$$p(a_1) \geq p(a_2) \geq \cdots \geq p(a_\alpha).$$

2. Definir las *frecuencias acumuladas* como:
$$c_1 = 0 \qquad y \qquad c_i = \sum_{j=1}^{i-1} p(a_j) \quad para \quad i = 2,3,\ldots,\alpha. \tag{2.48}$$

3. Asignar la palabra código para el símbolo a_i como los primeros
$$\hat{\ell}(a_i) = \left\lceil log_2 \left(\frac{1}{p(a_i)} \right) \right\rceil$$
dígitos binarios de la expansión binaria de la frecuencia acumulada c_i. [9]

El rasgo más característico de la codificación de Shannon son las longitudes de las palabras código predefinidas en la ecuación 2.44. Esta elección siempre garantiza una codificación casi óptima, hasta un β-bit de diferencia respecto a la longitud promedio de la palabra código mínima para todo código libre de prefijos, definida en la ecuación 2.40.

[9] Esta implementación del algoritmo de codificación de Shannon fue propuesta por el mismo *Claude E. Shannon* en su artículo [17].

Ahora bien, dado que la codificación de Shannon se implementa como un código libre de prefijos y la codificación de Huffman construye el código libre de prefijos óptimo, entonces la longitud promedio de la palabra código de la codificación de Huffman es menor o igual que la longitud promedio de la palabra código de la codificación de Shannon, para cualquier fuente de símbolos X, para la cual se conoce su función de masa de probabilidad. Por lo tanto, la longitud promedio de la palabra código de la codificación de Huffman de una fuente X, siempre está a lo sumo a un β-bit del límite de compresión teórico óptimo: la longitud promedio de la palabra código mínima para todo código libre de prefijos, definida en la ecuación 2.40.

2.5.2.2.2 *Codificación de Fano*

Existen algoritmos de codificación para los cuales las longitudes de las palabras código no son predefinidas, y aún así consiguen asignar las palabras código de manera casi óptima. Uno de estos procedimientos es *la codificación de Fano*. La implementación de la codificación de Fano que se presenta a continuación fue generalizada para β símbolos, a partir de la formulación de la referencia [13]. El algoritmo de Fano se implementa como el siguiente procedimiento de cinco pasos:

1. Ordenar la listas de símbolos según su frecuencia, de manera decreciente, sin perdida de generalidad, tal que:
$$p(a_1) \geq p(a_2) \geq \cdots \geq p(a_\alpha).$$

2. Dividir la lista en β partes diferentes, de modo que las frecuencias totales, $q_k = \sum_{i=k/\beta}^{\alpha} p(a_i)$ para $k = 1, \ldots, \beta$, de cada grupo sean lo más similares posible: de modo que la cantidad
$$\frac{1}{\beta} \cdot \sum_{j=1}^{\beta} \sum_{k=1}^{\beta} |q_j - q_k| \quad \text{es mínima.}$$

3. Asignar el dígito b_i al i-ésimo grupo, para $i = 1,2, \ldots, \beta$. De modo que, las palabras código de los símbolos en cada grupo comiencen con un símbolo del alfabeto $\mathcal{B}$ diferente, que los de cualquier otro grupo.

4. Aplicar recursivamente los pasos 2 y 3 a cada uno de los β grupos, subdividiendo cada grupo en β subgrupos y agregando β-bits a las palabras código hasta que cada símbolo sea el miembro único de su propio grupo.

Para la construcción de Fano se ha demostrado en [13] que la cota de la longitud promedio de la palabra código es
$$\left(\sum_{i=1}^{\alpha} p(a_i) \cdot \ell(a_i) \right) \leq \left[\sum_{i=1}^{\alpha} p(a_i) \cdot log_\beta \left(\frac{1}{p(a_i)} \right) \right] + 1 - \min_{1 \leq i \leq \alpha} \{p(a_i)\}. \tag{2.49}$$
Es importante notar que en el paso 2 de la construcción del algoritmo de Fano existe cierta ambigüedad, en el sentido en que se pueden definir más de una división posible, lo que da como resultado códigos distintos: las longitudes de las palabras código no están predefinidas, el algoritmo las elige de acuerdo al criterio del paso 2. No obstante, cómo se señala en la referencia [13], cualquiera que sea el código resultante, siempre cumplirá la cota de la ecuación 2.49. Ahora bien, note que la cota superior de la codificación de Fano es ligeramente más ajustada que la cota para la codificación de Shannon.

El rasgo más característico de la codificación de Fano es que construye un árbol β-ario que representa un código libre de prefijos, dividiendo desde la raíz hasta las hojas. A esta aproximación se le llama *arriba-abajo*. A diferencia del algoritmo de codificación de Fano, el algoritmo de codificación de Huffman realiza la construcción del árbol desde las hojas hasta su raíz, lo que se podría llamar una aproximación *abajo-arriba*.

2.5.3. Compresión bajo *ignorancia* de la distribución de la fuente

En esta sección se generaliza todo lo visto en la sección anterior. Y es que, en la compresión sin perdida bajo conocimiento se asume la fuerte suposición de que se tiene completo conocimiento acerca de la función de masa de probabilidad de la variable aleatoría que modela la fuente. Sin embargo, esto último no siempre es el caso. Cómo se verá más adelante (al final de esta sección), asumir esto para construir las palabras código, cuando la distribución verdadera es diferente, puede llevar a que el límite de compresión estimado por la implementación esté más alejado de lo que se cree del verdadero límite de compresión teórico.

Para generalizar las ideas de compresión sin perdida bajo conocimiento, al caso bajo ignorancia de la distribución de la fuente, se recurre a *la teoría de la información*.

2.5.3.1. Medidas de información

En esta sección se presentan dos de las medidas de información más conocidas, luego se explica la relación que existe entre ellas y la manera cómo se aplican a la implementación de la compresión bajo ignorancia.

2.5.3.1.1 *Contenido de información de Hartley*

Con lo visto hasta ahora, es posible formular una de las primeras medidas de *información*, tal cómo se entiende hoy en día: el *contenido de información de Ralph Hartley*. La intuición detras de esta es la siguiente:

> *El contenido de información, I_H, en un mensaje que consiste de n símbolos, cada uno de los cuales forman parte de un alfabeto de α símbolos diferentes, debe ser proporcional a la longitud n, del mensaje, multiplicada por alguna función del tamaño del alfabeto, $f(\alpha)$, tal que*
>
> $$I_H = n\, f(\alpha).$$
>
> *(Hartley. R, 1928, [7]).*

La manera cómo se formaliza el *contenido de información de Ralph Hartley* por mensaje es como la longitud de la codificación del mensaje, por medio del código de longitud fija que se definió en el capítulo uno, 2.3: $n \cdot log_\beta(\alpha)$. De esta manera, el contenido de información se conserva bajo cambios en el tamaño del alfabeto $\mathcal{A}$, como se demostró en el capítulo uno, de acuerdo a los requerimientos de información de Hartley.

Lo anterior posibilita definir, también, el concepto de contenido de información de Hartley por símbolo como el contenido de información de Hartley por mensaje divido entre el número de símbolos que componen el mensaje: $log_\beta(\alpha)$, longitud fija de la palabra código mínima. Observe que el contenido de información por símbolo se reescala bajo cambios en el tamaño del alfabeto $\mathcal{B}$:

$$log_\gamma(\alpha) = log_\gamma\left(\beta^{log_\beta(\alpha)}\right) = log_\beta(\alpha) \cdot log_\gamma(\beta). \tag{2.50}$$

Debido a la manera cómo se formuló el contenido de información de Ralph Hartley: como el número de símbolos del alfabeto $\mathcal{B}$ necesarios para codificar todos los símbolos del alfabeto $\mathcal{A}$, la unidad de medida para el contenido de información de Hartley es β-bits.

Observe que para la escogencia de $\ell(a_i) = log_\beta(\alpha)$, para todo $i = 1,2,\dots,\alpha$, independientemente de la probabilidad de cada símbolo, la longitud promedio de la palabra código es el contenido de información de Hartley por símbolo:

$$\mathbb{E}\left[log_\beta(\alpha)\right] = \sum_{i=1}^{\alpha} p(a_i) \cdot log_\beta(\alpha) = log_\beta(\alpha) \cdot \sum_{i=1}^{\alpha} p(a_i) = log_\beta(\alpha).$$

2.5.3.1.2 *Entropía de Shannon*

La primera medida de información, el contenido de información Hartley, es la mínima longitud de las palabras, para un código de longitud fija, con la que se puede obtener un código únicamente decodificable. Recuerde que el contenido de información de Hartley se formuló a través de la teoría de códigos, por medio de la longitud fija mínima de un código libre de prefijos del alfabeto. Aunque en ese momento no se explicitó, la formulación del contenido de información de Hartley se encuentra fuertemente sujeto a requerimientos de implementación del código. No en vano, la idea fundamental detrás del contenido de información de Hartley es medir la *información* mediante las longitudes de las palabras código de un código libre de prefijos de la fuente. Sin embargo, el contenido de información de Hartley no tiene en cuenta las características de la fuente. Es posible generalizar el contenido de información de Hartley para incluir la función de masa de probabilidad propia de la fuente, y para cualquier código libre de prefijos (no exclusivamente de longitud fija).

A continuación, se presenta la formulación teórica de esta generalización. Seguido de esto, se comprueba que, efectivamente, puede generalizarse para todo código libre de prefijos.

Los requerimientos para una medida de *incertidumbre*, o *entropía*, $H\left[\{p(a_i)\}_{i=1}^{\alpha}\right] = H\left[P\right]$, descritos por Claude E. Shannon en un artículo original de 1948, [17], son los siguientes:

- $H\left[P\right]$ es una función continua de las probabilidades, $p\left(a_i\right)$ para todo $i = 1,2,\ldots,\alpha$.
- Si todas las probabilidades, $p(a_i)$, son iguales, $p(a_i) = 1/\alpha$, entonces $H\left[P\right]$ es una función monótona creciente de α.
- $H\left[P\right]$ es una cantidad aditiva.

Estos requerimientos pretenden formalizar la idea de Shannon acerca de cómo se debería comportar una medida de *información*. Esta entropía fue propuesta como una generalización del contenido de información de Hartley, por Claude E. Shannon, en 1948, en *A mathematical Theory of Communication*, [17].

Shannon demostró analíticamente, estableciendo los requerimientos anteriores como axiomas, que la siguiente cantidad, como función de las probabilidades de los símbolos del alfabeto, es, en efecto, la única cantidad que satisface todos los axiomas:

$$H\left[\{p(a_i)\}_{i=1}^{\alpha}\right] = \sum_{i=1}^{\alpha} p(a_i) \cdot log_{\beta}\left(\frac{1}{p(a_i)}\right). \tag{2.51}$$

Esta cantidad es conocida, hoy en día, como la *entropía de Shannon*.

La entropía de Shannon, es una medida de *información* en el sentido sugerido, originalmente, por Ralph Hartley. Ciertamente se puede pensar la entropía de Shannon como una generalización del contenido de información de Harley: mide la información en términos de las mismas unidades fundamentales que el contenido de información de Hartley, β-bits, al tiempo que generaliza para asignar a los símbolos más inesperados mayor cantidad de información en el sentido de Harltey, es decir, palabras código más largas. Actualmente, se considera la entropía de Shannon como la medida cuantitativa de *información* más completa que se tiene, debido a que cumple los requerimientos de Shannon, lo cuales, cómo el mismo demostró en [17], son suficientes para solucionar apropiadamente muchos de los problemas en ingeniería de comunicaciones.

Bajo la perspectiva de Hartley, se interpreta que los símbolos más probables contienen menos información que los más improbables. Esta curiosa relación entre la minimización de la longitud promedio de la palabra código y la manera cómo Hartley midió la *información* resulta ser bastante intuitiva: piense, por ejemplo, que dos personas juegan a adivinar una palabra, únicamente con la siguiente *información*; una letra en esa palabra. Entonces, piense en la cantidad de posibilidades que habría si resulta que la *información* que se le

dio fue que la palabra contiene la letra e, el símbolo más probable en el inglés y el español. Mientras que por otro lado, piense en cuántas posibilidades habría sí en lugar de la letra e, se le hubiera dicho que la palabra contiene la letra x. En el segundo caso, las posibilidades se reducen considerablemente, teniendo en cuenta que la x es mucho menos probable, en el español y en el inglés, que la letra e, porque hay menos palabras que contengan la letra x que palabras que contengan la letra e. Por lo cual, al asignar palabras mas largas a símbolos menos probables se logra capturar satisfactoriamente lo que se entendería como *información* según Hartley, ya que los símbolos menos probables poseen más información y tendrán los códigos más largos.

2.5.3.2. *Principio de máxima entropía*

Cuando la distribución de frecuencias es desconocida, no es posible comprimir la codificación de la fuente con ninguno de los procedimientos vistos en la sección anterior, compresión bajo conocimiento: el algoritmo de Shannon predefine las longitudes de las palabras código en función de las probabilidades de los símbolos del alfabeto, mientras que los algoritmos de Fano y Huffman construyen árboles libres de prefijos basados en las frecuencias de los símbolos en cada iteración.

Ahora mismo, el problema no es conocer cuál es la compresión óptima, pues se conoce la manera de obtenerla, para cualquier distribución de la fuente, ya que existen los procedimientos que la computan, dada la distribución de los símbolos del alfabeto. El problema, ahora mismo, dado que no se conoce la distribución de probabilidad de la fuente, es determinar una distribución la cual alimentarle al algoritmo de Huffman (el compresor óptimo), lo cual determinaría, hasta un β-bit de precisión, el límite de compresión óptima que se busca. Así que, el problema de la compresión bajo ignorancia es equivalente al problema de determinar una distribución razonable la cual alimentarle al algoritmo de Huffman: la elección de la distribución no puede ser arbitraría, ya que es posible que a distintas distribuciones les correspondan compresiones distintas. Se requiere una manera no arbitraria de determinar la distribución de frecuencias que mejor represente la situación de desconocimiento actual acerca de la fuente.

La teoría de la información, que se ha venido presentando, es una herramienta que puede ayudar a solucionar, razonablemente, el problema de la compresión bajo ignorancia. En este capítulo, se maneja la interpretación de *medida de incertidumbre* que se le da, dentro de la teoría de la información, a la longitud promedio de la palabra código mínima de los códigos libres de prefijos para un alfabeto. Así mismo, se presenta un poco de la motivación que llevó a los pioneros de esta disciplina a otorgar ese significado particular a la ecuación 2.51. Esto ayudará a introducir el formalismo necesario para solucionar el problema de determinar la distribución desconocida, en este capítulo: considerar la entropía de Shannon por su significado en la teoría de la información puede dar respuesta a la cuestión acerca de cómo estimar de manera objetiva una distribución desconocida, a partir de las restricciones que en el momento se sepa que debe cumplir, por medio del *principio de máxima entropía*.

Observe que, aún cuando no se conoce exactamente la distribución de probabilidad de los símbolos de una fuente, es común disponer de *información* general acerca de la distribución en cuestión. Normalmente se entiende que conforme menos *conocimiento* se tenga, menos precisa es la descripción que se puede hacer acerca de la distribución de probabilidad, por lo que, a menos que se tenga *información total*, terminan existiendo una cantidad infinita de distribuciones diferentes las cuales caben en la descripción. Inversamente, mientras más *información* o *conocimiento* se tenga, más completa y restringida es la descripción de la distribución en cuestión. En efecto, dicha información está dada en forma de restricciones que la distribución debe cumplir, al punto que, únicamente bajo *total conocimiento*, la distribución es única. Para cualquier otro caso existen una cantidad infinita de distribuciones posibles que considerar.

Dentro de la teoría de la información es usual que se interprete la entropía de Shannon como una medida de incertidumbre, como la falta de conocimiento que se tiene, en promedio, acerca de un sistema: si la información que contiene un símbolo es la longitud de la palabra código asignada a este, en la codificación que comprime, entonces la longitud promedio de la plabra código, la entropía de Shannon de la ecuación 2.51

mide la incertidumbre acerca del sistema. En este caso, el sistema está compuesto por símbolos de un alfabeto que aparecen con determinada probabilidad dentro de los mensajes que pueden escribirse con ese alfabeto. La incertidumbre acerca de la distribución hace que no se sepa qué símbolos proporcionan más información que otros, lo cual debe reflejarse en una longitud promedio de la palabra código mínima mayor, es decir, una mayor entropía.

Al escoger la distribución de frecuencias es importante garantizar que la elección no sea sesgada, en el sentido de que no asume más conocimiento del que realmente se tiene acerca del sistema. Por lo que, es razonable elegir, entre las distribuciones de probabilidad que son consistentes con las propiedades conocidas del sistema, la distribución que maximiza la entropía. De esta manera, la distribución de probabilidad, según el conocimiento actual, se deriva mediante un proceso de maximización con restricciones. Este método asegura que no se incluye más conocimiento, en la descripción del sistema, de lo que ya se tiene.

La idea del parrafo anterior es la idea básica detras del *principio de máxima entropía*, también conocido como el *principio de sesgo mínimo* presentado en la referencia [11]. La formulación matemática del principio de máxima entropía a continuación a sido tomado de la referencia [11].

Definición 2.5.4 (*Principio de máxima entropía*)**.** Derivar una distribución de probabilidad, $P = \{p(a_i)\}_{i=1}^{\alpha}$, sujeta a restricciones de la forma:

$$\sum_{i=1}^{\alpha} p(a_i)\, f_k(a_i) = F_k, \tag{2.52}$$

para $k = 1,2,\ldots,r$, mediante la maximización de la función de entropía,

$$H[P] = \sum_{i=1}^{\alpha} p(a_i)\, log_{\beta}\left(\frac{1}{p(a_i)}\right),$$

sujeta a las reestricciones 2.52 y a la condición de normalización sobre los parámetros, $\{p(a_i)\}_{i=1}^{\alpha}$:

$$\sum_{i=1}^{\alpha} p(a_i) = 1. \tag{2.53}$$

El principio de máxima entropía establece un problema de máximización bajo reestricciones. Para solucionar este problema, se puede emplear el método de los multiplicadores de Lagrange, muy similar a cómo se obtuvo el límite óptimo de compresión:

$$\begin{aligned} &\text{Maximizar:} \quad L[p(a_1),\ldots,p(a_\alpha)] = \\ &= H[P] + \sum_{k=1}^{r} \lambda_k \left(F_k - \sum_{i=1}^{\alpha} p(a_i)\, f_k(a_i) + \right) + (\mu - 1)\left(1 - \sum_{i=1}^{\alpha} p(a_i)\right). \end{aligned} \tag{2.54}$$

Se han introducido los multiplicadores de Lagrange, λ_k, para cada una de las r restricciones, y $(\mu - 1)$ para la restricción de normalización (Se elige $(\mu - 1)$, en lugar de solo μ, porque hace que las expresiones en la derivación sean un poco más simples). Ahora bien, el problema de optimización se resuelve encontrando: $P = \{p(a_i)\}_{i=1}^{\alpha}$, λ_k, y μ, los cuales den ceros en las correspondientes derivadas parciales de la función de Lagrange, L, en la ecuación 2.54.

2.5.3.3. *Distribución de máxima entropía*

A la distribución resultante de solucionar el problema de máximización 2.54 ([11]), se le conoce como la *distribución de máxima entropía*. En esta sección, solucionamos el problema de maximización 2.54, lo cual dá

como resultado la distribución de frecuencias más razonable, de acuerdo al criterio de teoría de la información del principio de máxima entropía. Una vez se haya determinado la distribución, esta será el argumento al algoritmo de codificación de Huffman para determinar el límite de la compresión bajo desconocimiento.

Al derivar la ecuación 2.54, respecto a cada uno de los parámetros, $p(a_j)$, se tiene que:

$$\frac{\partial L}{\partial\, p(a_j)} = \frac{\partial H\,[P]}{\partial p\,(a_i)} - \sum_{k=1}^{r} \lambda_k\, f_k(a_j) - \mu + 1. \tag{2.55}$$

Donde, la derivada de la entropía respecto al parámetro $p\,(a_i)$ se puede calcular como:

$$\begin{aligned}
\frac{\partial H\,[P]}{\partial p\,(a_i)} &= \frac{\partial}{\partial p\,(a_i)} \left(\sum_{i=1} p\,(a_i)\ log_\beta \left(\frac{1}{p\,(a_i)} \right) \right) \\
&= \frac{\partial}{\partial p\,(a_i)} \left(-\sum_{i=1} p\,(a_i)\ log_\beta\,(p\,(a_i)) \right) \\
&= -\frac{\partial}{\partial p\,(a_i)} \left(\sum_{i=1} p\,(a_i)\ log_\beta\,(p\,(a_i)) \right) \\
&= -\left(log_\beta\,(p\,(a_i)) + p\,(a_i) \cdot \frac{\partial}{\partial p\,(a_i)} \left(log_\beta\,(p\,(a_i)) \right) \right),
\end{aligned} \tag{2.56}$$

donde, se lleva a cabo el siguiente cálculo:

$$\begin{aligned}
\frac{\partial}{\partial p\,(a_i)} \left(log_\beta\,(p\,(a_i)) \right) &= \frac{\partial}{\partial p\,(a_i)} \left(log_\beta \left(e^{ln(p(a_i))} \right) \right) \\
&= \frac{\partial}{\partial p\,(a_i)} \left(ln\,(p\,(a_i)) \cdot log_\beta\,(e) \right) \\
&= log_\beta\,(e) \cdot \frac{\partial}{\partial p\,(a_i)} \left(ln\,(p\,(a_i)) \right) \\
&= log_\beta\,(e) \cdot \frac{1}{p\,(a_i)}.
\end{aligned} \tag{2.57}$$

Así que

$$\frac{\partial H\,[P]}{\partial p\,(a_i)} = -\left(log_\beta\,(p\,(a_i)) + p\,(a_i) \cdot \left(log_\beta\,(e) \cdot \frac{1}{p\,(a_i)} \right) \right) = -log_\beta\,(p\,(a_i)) - log_\beta\,(e)\,. \tag{2.58}$$

Por lo tanto,

$$\frac{\partial L}{\partial\, p(a_j)} = log_\beta \left(\frac{1}{p\,(a_i)} \right) + log_\beta \left(\frac{1}{e} \right) - \sum_{k=1}^{r} \lambda_k\, f_k(a_j) - \mu + 1. \tag{2.59}$$

Luego, igualando a cero, $\partial L/\partial p(a_j) = 0$, se sigue que:

$$log_\beta \left(\frac{1}{p\,(a_i)} \right) + log_\beta \left(\frac{1}{e} \right) - \sum_{k=1}^{r} \lambda_k\, f_k(a_j) - \mu + 1 = 0, \tag{2.60}$$

es decir, que

$$log_\beta \left(\frac{1}{p\,(a_i)} \right) = -log_\beta \left(\frac{1}{e} \right) + \sum_{k=1}^{r} \lambda_k\, f_k(a_j) + \mu - 1 \tag{2.61}$$

y por tanto

$$log_\beta\,(p\,(a_i)) = log_\beta \left(\frac{1}{e} \right) - \sum_{k=1}^{r} \lambda_k\, f_k(a_j) - \mu + 1. \tag{2.62}$$

De donde se sigue que

$$p(a_j) = \beta^{\left(1-log_\beta\left(\frac{1}{e}\right)-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)-\mu\right)}. \tag{2.63}$$

Una distribución de probabilidades de esta forma es llamada *distribución de Máxima entropía* ([11]).

Los valores de las variables de Lagrange se pueden derivar a partir de las reestricciones, en la ecuación 2.52: la condición de normalización sobre los parámetros $\{p(a_i)\}_{i=1}^{\alpha}$, ecuación 2.53, determina la variable de Lagrange μ como una función de las demas variables de Lagrange, λ_k $(k = 1,2,...\,r)$, ya que

$$\begin{aligned}
\sum_{j=1}^{\alpha} p(a_j) &= \sum_{j=1}^{\alpha}\left[\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)-\mu\right)}\right] \\
&= \beta^{\left(1-log_\beta\left(\frac{1}{e}\right)-\mu\right)}\cdot\sum_{j=1}^{\alpha}\left[\beta^{\left(-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)\right)}\right] \\
&= \beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)}\cdot\frac{1}{\beta^{\mu}}\cdot\sum_{j=1}^{\alpha}\left[\beta^{\left(-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)\right)}\right] \\
&= 1,
\end{aligned} \tag{2.64}$$

implica que

$$\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)}\cdot\sum_{j=1}^{\alpha}\left[\beta^{\left(-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)\right)}\right] = \beta^{\mu}. \tag{2.65}$$

Por lo cual,

$$\mu = log_\beta\left(\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)}\cdot\sum_{j=1}^{\alpha}\left[\beta^{\left(-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)\right)}\right]\right). \tag{2.66}$$

Finalmente, las variables de Langrange λ_k están determinadas por las reestricciones 2.52. Observe que, si denotamos

$$Z(\lambda_1,...,\lambda_r) = \left(\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)}\cdot\sum_{j=1}^{\alpha}\left[\beta^{\left(-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)\right)}\right]\right), \tag{2.67}$$

entonces, las reestricciones de la ecuación 2.52, se pueden reexpresar, utilizando la ecuación 2.66 y la regla de la cadena, como:

$$\begin{aligned}
\frac{\partial\mu}{\partial\lambda_k} &= \frac{\partial}{\partial\lambda_k}\left(log_\beta\left(Z\left(\lambda_1,\ldots,\lambda_r\right)\right)\right) \\
&= \frac{\partial}{\partial Z}\left(log_\beta\left(Z\left(\lambda_1,\ldots,\lambda_r\right)\right)\right)\cdot\frac{\partial}{\partial\lambda_k}Z\left(\lambda_1,\ldots,\lambda_k\right) \\
&= \frac{1}{Z(\lambda_1,...,\lambda_r)}\cdot\sum_{i=1}^{\alpha}\left(-f_k(a_i)\right)\left(\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)}\cdot\sum_{j=1}^{\alpha}\left[\beta^{\left(-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)\right)}\right]\right) \\
&= -\sum_{i=1}^{\alpha} f_k(a_i)p(a_i) \\
&= -F_k.
\end{aligned} \tag{2.68}$$

para cada $k = 1,2,...,r$. De donde se puede deducir el valor de cada λ_k, integrando la ecuación 2.68.

Así se encuentran las distribuciones de máxima entropía. Ahora se sabe, que las distribuciones asumidas conocidas en la sección anterior, compresión bajo conocimiento, deberían haber sido distribuciones de máxima entropía, para ser consistentes con este razonamiento de la *teoría de la información.*

Ahora bien, observe que sí no se asume ningún conocimiento *a priori* acerca de la distribución, más allá de la condición de normalización, entonces las reestricciones de la ecuación 2.52 desaparecen, es decir los multiplicadores de Langrange λ_k, son todos cero para $k = 1,2,\ldots,r$. Por lo que la ecuación 2.63 quedaría expresada como:

$$p(a_j) = \beta^{\left(1-log_\beta\left(\frac{1}{e}\right)-\mu\right)}, \tag{2.69}$$

mientras que el valor del multiplicador de Langrange μ, en la ecuación 2.66, sería

$$\begin{aligned}
\mu &= log_\beta\left(\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)} \cdot \sum_{j=1}^{\alpha}\left[\beta^{\left(-\sum_{k=1}^{r}\lambda_k\, f_k(a_j)\right)}\right]\right) \\
&= log_\beta\left(\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)} \cdot \sum_{j=1}^{\alpha}\beta^0\right) \\
&= log_\beta\left(\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)} \cdot \alpha\right) \\
&= log_\beta\left(\beta^{\left(1-log_\beta\left(\frac{1}{e}\right)\right)}\right) + log_\beta\left(\alpha\right) \\
&= \left(1 - log_\beta\left(\frac{1}{e}\right)\right) + log_\beta\left(\alpha\right).
\end{aligned} \tag{2.70}$$

Por lo tanto,

$$\begin{aligned}
p(a_j) &= \beta^{\left(1-log_\beta\left(\frac{1}{e}\right)-\mu\right)} \\
&= \beta^{-log_\beta(\alpha)} \\
&= \beta^{log_\beta\left(\frac{1}{\alpha}\right)} \\
&= \frac{1}{\alpha},
\end{aligned} \tag{2.71}$$

para $j = 1,2,\ldots,\alpha$.

La ecuación 2.71 puede interpretarse de la siguiente manera: «Cuando no se tiene ningún conocimiento *a priori* acerca de cómo se comporta el sistema, entonces para comprimir, lo más razonable es asumir que la distribución es la distribución uniforme.»

Lo anterior demuestra que los códigos de longitud variable no siempre son la mejor opción para comprimir la codificación de un alfabeto y que los código de longitud fija pueden ser de mucha utilidad: cuando no se conocen las frecuencias relativas de los símbolos del alfabeto $\mathcal{A}$, ni tampoco se dispone de ningún conocimiento *a priori*, lo más razonable es no asignar palabras código de longitud variable, siempre que los códigos de longitud fija, concretamente, el contenido de información de Hartley, es el límite de la compresión de la codificación del alfabeto $\mathcal{A}$, en el caso en el que se desconocen las frecuencias relativas de los símbolos del alfabeto $\mathcal{A}$. Por lo tanto, es posible notar que los códigos de longitud fija son la mejor ópción para comprimir, cuando se tiene completo desconocimiento acerca de la función de masa de probabilidad de la fuente.

Observe, además, que el contenido de información de Hartley se puede derivar, a partir de la entropía de Shannon, para un tipo muy particular de distribución de máxima entropía: la distribución que se obtiene al no tener ningún conocimiento a priori de la distribución. Así que, Ralph Hartley, en efecto, tuvo razón a medir la información de la manera en la que lo hizo, aunque no consideró que solamente la estaba midiendo apropiadamente para el caso en el que no se tiene ninguna información a priori. Lo cual tiene sentido, ya que al derivar su medida, Hartley no consideró ninguna restricción sobre la distribución.

2.5.4. Límites en la implementación de la compresión

En esta sección, se derivan las cotas más generales de la estimación del límite de compresión mediante su implementación a través de la codificación de alfabetos.

Antes, se clasificó la codificación de Shannon como un código semi-óptimo. Esto se debe a que la codificación de Shannon implementa un código libre de prefijos y se sabe que el código libre de prefijos óptimo se construye con el algoritmo de Huffman. Por lo tanto, la codificación de Shannon es, en el mejor de los casos tan óptimo como la codificación de Huffman. No obstante, sabemos que la codificación de Shannon garantiza que, a lo sumo, la longitud promedio del código resultante está a un β-bit de diferencia respecto al mínimo teórico alcanzable. No obstante, la codificación de Huffman, por ser la codificación óptima, también cumple está cota. Más aún, como se vió en la sección de la codificación de Fano, la cota es un poco más ajustada que eso. Sin embargo, la cota de la codificación de Shannon es suficiente para lo que interesa a este escrito. Por lo tanto, la longitud promedio de la palabra código de la codificación óptima libre de prefijos de una fuente (codificación de Huffman) siempre está a lo sumo a un β-bit de distancia del mínimo teórico alcanzable.

No en vano, lo que lo anterior sugiere es que existe una discrepancia entre los óptimos teóricos de la codificación de una fuente y los óptimos realmente alcanzables en la práctica: en parte, debido a que las longitudes de las palabras código deben implementarse con palabras código de longitudes enteras. Como ya se demostró, todos los procedimientos de codificación libre de prefijos de alfabetos -símbolos individuales- óptimos producen, incluso en el mejor de los casos, palabras código para las cuales su longitud promedio de la palabra código por símbolo está a lo sumo a un β-bit de distancia de su valor mínimo óptimo:

$$H\left[\{p(a_i)\}_{i=1}^{\alpha}\right] \leq \left(\sum_{i=1}^{\alpha} p(a_i)\cdot \ell(a_i)\right) \leq H\left[\{p(a_i)\}_{i=1}^{\alpha}\right] + 1. \tag{2.72}$$

Esto genera un exceso de β-bits a la hora de transmitir un mensaje de longitud mayor que uno, mediante la extensión del código. La cota por símbolo, en la ecuación 2.72, se puede extender, linealmente, al caso de m símbolos: observe que codificando el alfabeto $\mathcal{A}$, mediante la codificación de Shannon, se obtiene que para un mensaje de longitud m, compuesto por símbolos de $\mathcal{A}$, la longitud promedio del mensaje codificado óptimo es

$$L_{opt} = m\cdot \mathbb{E}(\ell(a_i)) = m\cdot \left(\sum_{i=1}^{\alpha} p(a_i)\cdot \ell(a_i)\right), \tag{2.73}$$

por ser la combinación lineal de variables aleatorias, donde

$$m\cdot H\left[\{p(a_i)\}_{i=1}^{\alpha}\right] \leq L_{opt} \leq m\cdot H\left[\{p(a_i)\}_{i=1}^{\alpha}\right] + m. \tag{2.74}$$

De modo que el exceso de β-bits crece a una tasa $O(m)$, siendo m la longitud del mensaje codificado.

Sin embargo, lo anterior es únicamente cierto en el hipotético caso en el que se tiene conocimiento completo de las probabilidades de los símbolos del alfabeto. En la práctica, es de esperar disponer solo de un conocimiento parcial de las frecuencias, situación en la cuál es razonable emplear el *principio de máxima entropía*, cómo se hizo en la sección anterior. En ese caso, la distribución resultante es la mejor estimación que se puede hacer de la distribución verdadera desconocida, pero no ser suficiente para alcanzar el mínimo teórico alcanzable. Esta diferencia, que se suma al exceso de β-bits debido a la aproximación entera, es significativa a la hora de codificar los símbolos de un mensaje suficientemente largo, mediante cualquiera de los procedimientos de codificación de alfabetos conocidos (códigos canónicos o códigos semi-óptimos): suponga que la verdadera distribución de las frecuencias es $\{p(a_i)\}_{i=1}^{\alpha}$, mientras que las palabras código se definen en base a una aproximación, $\{q(a_i)\}_{i=1}^{\alpha}$, de la distribución verdadera:

$$\ell(a_i) = \left\lceil log_{\beta}\left(\frac{1}{q(a_i)}\right)\right\rceil. \tag{2.75}$$

En este caso, la longitud promedio de la implementación de las palabras código es tal que:

$$\begin{aligned}\mathbb{E}(\ell(a_i)) &= \sum_{i=1}^{\alpha} p(a_i)\left\lceil log_\beta\left(\frac{1}{q(a_i)}\right)\right\rceil \\ &\leq \sum_{i=1}^{\alpha} p(a_i)\left(log_\beta\left(\frac{1}{q(a_i)}\right)+1\right) \\ &= \sum_{i=1}^{\alpha} p(a_i)\left(log_\beta\left(\frac{p(a_i)}{q(a_i)}\cdot\frac{1}{p(a_i)}\right)+1\right) \\ &= \sum_{i=1}^{\alpha} p(a_i)\left(log_\beta\left(\frac{p(a_i)}{q(a_i)}\right)+log_\beta\left(\frac{1}{p(a_i)}\right)+1\right) \\ &= \sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{p(a_i)}{q(a_i)}\right)+\sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{1}{p(a_i)}\right)+\sum_{i=1}^{\alpha} p(a_i).\end{aligned} \tag{2.76}$$

Mientras que,

$$\begin{aligned}\mathbb{E}(\ell(a_i)) &= \sum_{i=1}^{\alpha} p(a_i)\left\lceil log_\beta\left(\frac{1}{q(a_i)}\right)\right\rceil \\ &\geq \sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{1}{q(a_i)}\right) \\ &= \sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{p(a_i)}{q(a_i)}\cdot\frac{1}{p(a_i)}\right) \\ &= \sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{p(a_i)}{q(a_i)}\right)+\sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{1}{p(a_i)}\right).\end{aligned} \tag{2.77}$$

Con lo cual,

$$\left(\sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{p(a_i)}{q(a_i)}\right)\right)+H\left[P\right]\leq\mathbb{E}(\ell(a_i))\leq\left(\sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{p(a_i)}{q(a_i)}\right)\right)+H\left[P\right]+1. \tag{2.78}$$

Observe que lo anterior afecta la compresión por cualquier aproximación: tanto a la construcción de los códigos canónicos como la definición de los códigos semi-óptimos, es decir, afecta tanto al algoritmo de Huffman como al algoritmo de Shannon. Pues, observe que la cantidad:

$$\left(\sum_{i=1}^{\alpha} p(a_i)log_\beta\left(\frac{p(a_i)}{q(a_i)}\right)\right)=\left[\sum_{i=1}^{\alpha} p(a_i)\cdot log_\beta\left(\frac{1}{q(a_i)}\right)\right]-\left[\sum_{i=1}^{\alpha} p(a_i)\cdot log_\beta\left(\frac{1}{p(a_i)}\right)\right], \tag{2.79}$$

en realidad, está midiendo qué tan subóptima es la asignación de las longitudes de las palabras código de la implementación (basadas en una distribución de máxima entropía): al comparar el mínimo implementado con el mínimo teórico, permite conocer qué tanto las longitudes escogidas se parecen a las longitudes óptimas. Lo que en últimas, sugiere que el exceso de β-bits en el que se incurre debido a desconocimiento de la verdadera distribución será cero únicamente cuando la distribución de la implementación sea exactamente igual a la verdadera. Por eso, se dice que esta cantidad representa el exceso de β-bits en el que se incurre debido al desconocimiento de la verdadera distribución de la fuente. Y por esto mismo sabemos que afecta de igual manera la implementación de la compresión mediante la codificación de Huffman, como por la codificación de Shannon-Fano, ya que más allá de la implementación escogida, este fenómeno se debe directamente al desconocimiento de la distribución, de las cuales ambas distribuciones dependen fuertemente para definir las longitudes de las palabras código óptimas.

La desigualdad 2.78, formula la idea de que dos de los principales impedimentos para lograr el mínimo teórico alcanzable para la codificación de una fuente, por medio de la codificación individual de símbolos del alfabeto de la fuente, son: las aproximaciones enteras de las longitudes de las palabras código óptimas y la ignorancia (o conocimiento) parcial que naturalmente se tiene acerca de una fuente de símbolos. El primer impedimento, el exceso de β-bits debido a la innevitable aproximación entera de la implementación, no debería ser por sí solo razón suficiente para descartar esta codificación, ya que su tasa de crecimiento es del orden lineal ($O(m)$, siendo m la longitud del mensaje transmitido). Mientras que, el exceso producido por la falta de conocimiento acerca de la distribución verdadera siempre puede, y debe, ser minimizado: por lo general, se acepta el primer tipo de exceso, por ser propio de una implementación (aunque evitable) y no ser computacionalmente muy costoso, mientras que se trata de minimizar esto último, logrando una aproximación más precisa de la distribución verdadera. A continuación, se define el exceso por desconocimiento, desde el punto de vista de la *teoría de la información*. Lo cual dará herramientas para refinar aún más la compresión, minimizando el exceso de β-bits debido al desconocimiento de la verdadera distribución. Es decir, lo que nos dará herramientas para saber cómo estimar de mejor manera la verdadera distribución de una fuente, para aprovechar la capacidad de ambas implementaciones de la compresión bajo conocimiento.

2.5.4.1. *Divergencia Kullback-Leibler*

En *teoría de la información*, la cantidad 2.79 recibe el nombre de *Divergencia Kullback-Leibler* o *Información Relativa*, [5]. A partir de su formulación, en la ecuación 2.79, la Divergencia Kullback-Leibler puede interpretarse como la cantidad de *información* que se adquiere en el proceso de transitar desde una distribución *a priori*, $\{q_i\}_{i=1}^{\alpha}$, a una nueva distribución de la fuente, $\{\hat{q}_i\}_{i=1}^{\alpha}$: observe que la cantidad

$$\sum_{i=1}^{\alpha} q(a_i) log_{\beta} \left(\frac{q(a_i)}{\hat{q}(a_i)} \right) \tag{2.80}$$

siempre es no negativa, ya que es la diferencia respecto al mínimo teórico, al tiempo que es cero únicamente cuando ambas distribuciones son idénticas. Sin embargo, aunque implementa una distancia (como pudiera ser una distancia cuadrática) no implementa una métrica matemática, ya que no satisface la desigualdad triangular [5]. Aún así, su noción de distancia puede utilizarse para medir cuánta información se obtiene a partir de una observación: piense que se parte de la distribución de máxima entropía como una primera estimación de la distribución verdadera, entonces sí la distribución de máxima entropía es una aproximación muy buena, se sigue que, aunque no se conozca, su divergencia respecto a la verdadera distribución será pequeña. Por lo cual, el margen de maximización de la divergencia Kullback-Leibler será muy poco, siendo el margen exactamente cero cuando la distribución de máxima entropía sea exactamente igual a la distribución verdadera. Es razonable asumir que toda observación de un sistema lleva, naturalmente, a un estado de mayor conocimiento acerca de su comportamiento (y su distribución), más aún, que al hacer una medición se obtiene una restricción adicional que permite obtener una nueva distribución mas cercana a la verdadera. La ganancia de información debido a la medición se puede dar en términos de β-bits usando la divergencia de Kullback-Leibler. Así se podría partir de la distribución de máxima entropía y tratar de realizar una observación que proporcione la mayor cantidad de información, es decir, que maximice la divergencia Kullback-Leibler, que es, la cantidad de información que se puede extraer de esa observación. De modo que, luego de la observación, la información que se habrá adquirido será, en el mejor de los casos, suficiente para calcular una nueva distribución de máxima entropía lo más parecida a la distribución verdadera.

Por lo tanto, lo que el concepto de divergencia Kullback-Leibler sugiere es que la mejor manera de eliminar el exceso de β-bits debido al desconocimiento de la distribución verdadera de una fuente es buscando la manera de realizar una observación precisa que maximice la divergencia, partiendo de la distribución de máxima entropía, la mejor estimación a priori que se puede lograr con el conocimiento actual. Así se estaría acercando lo más posible a la verdadera distribución de la fuente, ya que la observación estaría descubriendo nuevas reestricciones sobre la distribución que posibilitarián calcular una nueva distribución de máxima entropía más precisa.

Capítulo 3

Bloques de símbolos

3.1. Modelamiento de una fuente de símbolos

Una fuente de símbolos se puede pensar, de manera general, como una variable aleatoria discreta X, definida sobre un alfabeto $\mathcal{A}$, cuya función de masa de probabilidad, $P_X(\cdot)$, puede variar dependiendo del momento dentro de una cadena de símbolos en el que se encuentre la fuente. Particularmente, la probabilidad de generar un símbolo en algún momento puede depender de la cadena de símbolos que la fuente ha generado hasta ese momento, de modo que en el i-ésimo momento de la fuente pueden haber hasta $\alpha^{(i-1)}$ funciones de masa de probabilidad distintas, una para cada posible cadena de símbolos generada por la fuente hasta ese momento:

$$P_{X_i}(x_i) \equiv P(X_i = x_i) \longrightarrow P(X_i = x_i | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}).$$

Por esta razón, una única fuente puede modelarse como una secuencia de variables aleatorias discretas distintas (proceso estocástico), $\{X_i\}_{i=1}^{\infty}$, no necesariamente independientes e idénticamente distribuidas, donde el subíndice se utiliza para denotar que la variable aleatoria tiene una distribución, función de masa de probabilidad, específica en el i-ésimo momento:

$$\texttt{fuente:=} \quad X_1 X_2 \ldots X_{i-1} X_i,$$

para todo $i \in \mathbb{Z}^+$.

Mediante este formalismo, la dependencia de la probabilidad de un símbolo dada la cadena de símbolos anteriores se puede formalizar en términos de las funciones de masa de probabilidad conjunta, de las i variables aleatorias, como:

$$P(X_i = x_i | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}) = \frac{P(X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}, X_i = x_i)}{P(X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1})},$$

o equivalentemente, cambiando la notación:

$$P_{X_i}(x_i | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}) = \frac{P_{X_1 X_2 \ldots X_{i-1} X_i}(x_1, x_2, \ldots, x_{i-1}, x_i)}{P_{X_1 X_2 \ldots X_{i-1}}(x_1, x_2, \ldots, x_{i-1})}.$$

Formalmente, sí la probabilidad de generar un símbolo en cualquier momento solo depende de los últimos $(n-1)$ símbolos generados por la fuente, entonces se dice que la fuente tiene memoria de tamaño n:

$$P_{X_i}(x_i | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}) = P_{X_i}(x_i | X_{i-n} = x_{i-n}, X_{i-n+1} = x_{i-n+1}, \ldots, X_{i-1} = x_{i-1}).$$

Definición 3.1.1 (*Memoria de una fuente*)**.** Se define la *memoria de un fuente* como la longitud de la cadena de símbolos previos de la cual depende el resultado actual. En este sentido se dice que si

$$P_{X_i}\left(x_i|X_1=x_1,X_2=x_2,\ldots,X_{i-1}=x_{i-1}\right)=P_{X_i}\left(x_i|X_{i-n}=x_{i-n},X_{i-n+1}=x_{i-n+1},\ldots,X_{i-1}=x_{i-1}\right),$$

para todo $i\in\mathbb{Z}^+$, entonces la fuente tiene memoria de tamaño n. [21]

3.1.1. Fuente sin memoria

Una fuente sin memoria es una fuente con memoria de tamaño cero ($n=0$), tal que la secuencia de variables aleatorias que modelan la fuente son independientes e idénticamente distribuidas:

$$P_{X_i}\left(x_i|X_{i-n}=x_{i-n},X_{i-n+1}=x_{i-n+1},\ldots,X_{i-1}=x_{i-1}\right)=P_{X_i}\left(x_i\right)\qquad y\qquad P_{X_i}\left(x_i\right)=P_X\left(x_i\right),$$

para alguna variable aleatoria discreta X, para todo $i\in\mathbb{Z}^+$.

De acuerdo a la definición, para una fuente sin memoria se cumple, dada la independencia, que

$$\begin{aligned}
P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}X_i}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i\right)&=P_{X_i}\left(x_i\right)\cdot P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1}\right)\\
&=P_{X_i}\left(x_i\right)\cdot\left[P_{X_{i-1}}\left(x_{i-1}\right)\cdot P_{X_{i-n}X_{i-n+1}\ldots X_{i-2}}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-2}\right)\right]\\
&\vdots\\
&=P_{X_i}\left(x_i\right)\cdot P_{X_{i-1}}\left(x_{i-1}\right)\cdots\left[P_{X_{i-n+2}}\left(x_{i-n+2}\right)\cdot P_{X_{i-n}X_{i-n+1}}\left(x_{i-n},x_{i-n+1}\right)\right]\\
&=P_{X_i}\left(x_i\right)\cdot P_{X_{i-1}}\left(x_{i-1}\right)\cdots P_{X_{i-n+2}}\left(x_{i-n+2}\right)\cdot\left[P_{X_{i-n+1}}\left(x_{i-n+1}\right)\cdot P_{X_{i-n}}\left(x_{i-n}\right)\right]\\
&=\prod_{k=0}^{n}P_{X_{i-k}}\left(x_{i-k}\right),
\end{aligned}$$

mientras que, dado que las variables están idénticamente distribuidas, cada X_i tiene la misma distribución que una variable aleatoria discreta X:

$$P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}X_i}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i\right)=\prod_{k=0}^{n}P\left(X_{i-k}=x_{i-k}\right)=\prod_{k=0}^{n}P\left(X=x_{i-k}\right).$$

Definición 3.1.2 (*Fuente sin memoria*)**.** Una *fuente sin memoria* es una fuente de símbolos con memoria para la cual

$$P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}X_i}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i\right)=\prod_{k=0}^{n}P\left(X=x_{i-k}\right).\tag{3.1}$$

para alguna variable aleatoria discreta X, para todo $i\in\mathbb{Z}^+$ y toda cadena $x_{i-n}x_{i-n+1}\ldots x_{i-1}x_i\in\mathcal{A}^k$. [21]

3.1.2. Modelamiento mediante una fuente sin memoria

Algunos fenómenos pueden ser considerados fuentes de información sin memoria, en el sentido en que se asume que la probabilidad de observar un evento es independiente de cualquier otro evento. Algunos ejemplos de este tipo de fenómenos podrían ser:

1. La configuración de caras y sellos que se obtendrían como resultado de arrojar mil dados no sesgados al mismo tiempo.

2. La secuencia de resultados que se obtendrían al sacar la bola del baloto de la urna, luego de ser agitada, mil veces consecutivas (volviendo a poner la bola dentro de la urna cada vez).

3.1.3. Cadena de Markov

El concepto de *cadena de Markov* formaliza la idea de una fuente con memoria para la cual su función de masa de probabilidad, en cualquier instante i, depende únicamente de lo qué sucedió en el instante anterior $(i-1)$. Formalmente, una cadena de Markov se define como:

Definición 3.1.3 (*Cadena de Markov*)**.** Una *Cadena de Markov* es una fuente de símbolos con memoria para la cual

$$P_{X_i}\left(x_i|X_{i-n}=x_{i-n}, X_{i-n+1}=x_{i-n+1}, \ldots, X_{i-1}=x_{i-1}\right) = P_{X_i}\left(x_i|X_{i-1}=x_{i-1}\right). \tag{3.2}$$

para todo $n \in \mathbb{N}$ y toda cadena $x_1x_2\ldots x_n \in \mathcal{A}^k$. [21]

3.1.4. Modelamiento mediante cadenas de Markov

En esta sección se presentan algunas aplicaciones de modelos de Markov para mostrar que son muy útiles. Esto justifica por qué es razonable detenerse en una memoria de tamaño uno, en lugar de trabajar con correlaciones de mayor orden, además de que son matemáticamente más simples de analizar.

Las cadenas de Markov son una forma matemáticamente simple de modelar la dependencia entre estados consecutivos de un sistema. No obstante su simplicidad, existen una gran variedad de fenómenos que se comportan como cadenas de Markov. Entre los que interesan a este escrito se encuentran:

- **Generación de textos**: la aplicación de las cadenas de Markov para modelar las frecuencias de vocales y consonantes en textos escritos en lenguaje natural es reconocida hoy en día como la primera aplicación de una cadena de Markov. En su artículo original Andrei A. Markov estudió una secuencia de más de 20,000 caracteres dentro del poema Eugeny Onegin, descubriendo que la probabilidad de una vocal después de una letra vocal es 0,128, mientras que la probabilidad de una vocal después de una letra consonante es 0,663. Mientras que, la probabilidad estacionaria de las vocales es $p = 0{,}432$. Lo resultados fueron consistentes, y casi que pueden utilizarse cadenas de Markov para escribir poemas. Esta aplicación fue propuesta como una de las cinco aplicaciones más importantes de las cadenas de Markov en [9].

- **Compresión de textos en inglés**: En su artículo original de 1948, [17], Claude Shannon introduce el concepto de entropía a través de un modelamiento Markoviano del lenguaje inglés. Allí, se demostró que tales modelos idealizados, que él propuso, capturan mucha de la regularidad estadística del lenguaje inglés. Incluso, aunque no se logre una descripción completa de la estructura del sistema, él demostró que es posible comprimir satisfactoriamente, en base en estos modelos, textos en inglés a través de técnicas de codificación de entropía como la codificación aritmética.

3.1.5. Fuente estacionaria

Ahora, formalizamos el concepto de una fuente estacionaria mediante el formalismo presentado en la referencia [11]. La idea detrás de una fuente estacionaria es que las probabilidades para una secuencia de k símbolos dada, $x_1x_2\ldots x_k$, no depende de la posición absoluta dentro de la secuencia, sino únicamente de los símbolos que han sido generados con anterioridad en la secuencia. Aquí, se consideran únicamente secuencias de símbolos que son generadas por una fuente estacionaria, lo que implica la invariación de translación de las probabilidades.

Definición 3.1.4. Fuente estacionaria
Una fuente $\mathbb{X}_n$ es una fuente estacionaria, si para todo $k, m \in \mathbb{Z}^+$:

$$P\left(X_1=x_1, X_2=x_2, \ldots, X_k=x_k\right) = P\left(X_{1+m}=x_1, X_{2+m}=x_2, \ldots, X_{k+m}=x_k\right). \tag{3.3}$$

3.2. Teorema de codificación de la fuente

De acuerdo al formalismo presentado, la probabilidad condicional está definida en términos de las probabilidades conjuntas:

$$P_{X_i}\left(x_i|X_{i-n}=x_{i-n},X_{i-n+1}=x_{i-n+1},\ldots,X_{i-1}=x_{i-1}\right)=\frac{P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}X_i}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i\right)}{P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1}\right)}.$$

No en vano, la probabilidad conjunta de toda cadena de n símbolos consecutivos generados por una fuente con memoria sea puede formular en términos de las probabilidades condicionales como:

$$\begin{aligned}&P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}X_i}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i\right)=\\&=P_{X_i}\left(x_i|X_{i-n}=x_{i-n},X_{i-n+1}=x_{i-n+1},\ldots,X_{i-1}=x_{i-1}\right)\cdot P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1}\right).\end{aligned}$$

3.2.1. Teorema de propiedad de equipartición asintótica

Para una fuente sin memoria la probabilidad de todo bloque de n símbolos se puede reescribir como:

$$P_{X_{i-n}X_{i-n+1}\ldots X_{i-1}X_i}\left(x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i\right)=\prod_{k=0}^{n}P\left(X=x_{i-k}\right)=\prod_{k=1}^{\alpha}\left[P_X\left(a_k\right)\right]^{N(a_k|x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i)},$$

donde $\alpha=|\mathcal{A}|$, es el tamaño del alfabeto de la fuente, ya que la asumpción de **i.i.d** (independencia e idénticamente distribuida) permite permutar la cadena de símbolos, $x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i$, como:

$$x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i\longrightarrow\underbrace{a_1a_1\ldots a_1}_{N(a_1|x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i)}\underbrace{a_2a_2\ldots a_2}_{N(a_2|x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i)}\cdots\underbrace{a_\alpha a_\alpha\ldots a_\alpha}_{N(a_\alpha|x_{i-n},x_{i-n+1},\ldots,x_{i-1},x_i)},$$

de modo que el cálculo de la probabilidad es invariante bajo esta permutación. La cantidad $N\left(a_k|\mathbb{X}_n\right)$ cuenta el número de apariciones del símbolo a_k dentro de una observación particular de la cantidad aleatoria $\mathbb{X}_n$.

Se define la *entropía muestral* de una fuente independiente e idénticamente distribuida $\mathbb{X}_n$, al igual que en [23], como:

$$\begin{aligned}\frac{1}{n}\cdot log\left(\frac{1}{P_{\mathbb{X}_n}\left(\mathbb{X}_n\right)}\right)&=-\frac{1}{n}\cdot log\left(P_{\mathbb{X}_n}\left(\mathbb{X}_n\right)\right)\\&=-\frac{1}{n}\cdot log\left(\prod_{k=1}^{\alpha}\left[P_X\left(a_k\right)\right]^{N(a_k|\mathbb{X}_n)}\right)\\&=-\frac{1}{n}\cdot\sum_{k=1}^{\alpha}log\left(\left[P_X\left(a_k\right)\right]^{N(a_k|\mathbb{X}_n)}\right)\\&=-\frac{1}{n}\cdot\sum_{k=1}^{\alpha}N\left(a_k|\mathbb{X}_n\right)\cdot log\left(P_X\left(a_k\right)\right)\\&=\sum_{k=1}^{\alpha}\frac{N\left(a_k|\mathbb{X}_n\right)}{n}\cdot\left[-log\left(P_X\left(a_k\right)\right)\right]\\&=\sum_{k=1}^{\alpha}\frac{N\left(a_k|\mathbb{X}_n\right)}{n}\cdot log\left(\frac{1}{P_X\left(a_k\right)}\right).\end{aligned}$$

Por la ley de los números grandes, se sabe que dadas n variables aleatorias i.i.d, la distribución empírica, $N\left(a_i|\mathbb{X}_n\right)$, converge en probabilidad a la distribución poblacional, $P_X\left(a_i\right)$, del símbolo a_i:

$$\lim_{n\to\infty}\left\{P\left[\left|\frac{N\left(a_k|\mathbb{X}_n\right)}{n}-P_X\left(a_k\right)\right|>\delta\right]\right\}=0, \tag{3.4}$$

para todo $\delta > 0$. De modo que es poco probable que, dada una muestra de n variables aleatorias independientes e idénticamente distribuidas, la distribución empírica sea distinta a la distribución poblacional, conforme n tiende a infinito. En el límite, la distribución empírica es exactamente igual a la distribución poblacional.

Por lo cual, la ley de los números grandes garantiza que es muy probable que la fuente $\mathbb{X}_n$ satisfaga lo siguiente:

$$\lim_{n \to \infty} \left\{ P \left[\left| \sum_{k=1}^{\alpha} \frac{N\left(a_k | \mathbb{X}_n\right)}{n} \cdot log\left(\frac{1}{P_X\left(a_k\right)}\right) - \sum_{k=1}^{\alpha} P_X\left(a_k\right) \cdot log\left(\frac{1}{P_X\left(a_k\right)}\right) \right| \leq \delta \right] \right\} = 1,$$

o equivalentemente,

$$\lim_{n \to \infty} \left\{ P \left[\left| \frac{1}{n} \cdot log\left(\frac{1}{P_{\mathbb{X}_n}\left(\mathbb{X}_n\right)}\right) - H\left[X\right] \right| \leq \delta \right] \right\} = 1.$$

para todo $\delta > 0$. De modo que, la entropía muestral es un estimador consistente de la entropía de la fuente X.

Una observación particular de $\mathbb{X}_n$, $x_1 x_2 \ldots x_n$, se dice ser una *cadena típica* si su entropía muestral es cercana a la entropía $H\left[X\right]$. El conjunto de las cadenas típicas es llamado *conjunto típico*.

El tamaño del conjunto típico es aproximadamente $2^{n\,H[X]}$, mientras que el conjunto de todas las secuencias posibles de longitud n de α símbolos es α^n. El tamaño del conjunto de todas las cadenas posibles de longitud n se puede escribir como:

$$\alpha^n = 2^{n\,log(\alpha)},$$

lo cual sucede cuando $H\left[X\right] = log\left(\alpha\right)$ *i.e* cuando todos los símbolos del alfabeto son equiprobables.

Comparando el tamaño del conjunto típico con el tamaño del conjunto de todas las posibles cadenas de tamaño n de α símbolos, el conjunto típico es exponencialmente más pequeño que el conjunto de todas las posibles cadenas, siempre que la variable aleatoria que modela la fuente independiente no esté distribuida uniformemente.

Las características más importantes del conjunto típico, listadas en la referencia [23], son las siguientes:

1. **Probabilidad unitaria:** la probabilidad de que una cadena emitida por la fuente sea una cadena típica tiende a uno conforme n se vuelve más grande. Equivalentemente, el conjunto típico concentra toda la probabilidad de entre todas las posibles cadenas de longitud n, conforme n tiende a infinito.

2. **Cardinalidad exponencialmente más pequeña:** el tamaño del conjunto típico, $2^{n\,H[X]}$, es exponencialmente más pequeño que $2^{n\,log(\alpha)} = \alpha^n$, el tamaño del conjunto de todas las posibles cadenas de longitud n de α símbolos.

3. **Equipartición:** la probabilidad de una cadena típica es aproximadamente uniforme: $2^{-n\,H[X]}$.

El punto 1 se sigue directamente del teorema de la ley de los números grandes y se interpreta como que:

« *En el límite asintótico (cuando n tiende a infinito), es muy probable que la fuente de información emita una secuencia cuya entropía muestral sea cercana a la entropía verdadera de la fuente. Por el contrario, es muy poco probable que la fuente de información emita una secuencia que no satisfaga esta propiedad.* »

Los puntos 2 y 3 fueron demostrados originalmente por Shannon en [17]. La probabilidad del punto 3 es fácilmente deducible, si se asume que el conjunto típico concentra toda la probabilidad de entre las cadenas

de longitud n, su tamaño es $2^{n\,H[X]}$ y la distribución sobre el conjunto de las cadenas típicas es uniforme. Estas tres características del conjunto típico conforman el teorema de la **propiedad de equipartición asintótica**.

3.2.2. Esquema teórico de compresión de Shannon

El número total de cadenas de longitud n de símbolos del alfabeto $\mathcal{A}$ es α^n. Por lo tanto, es necesario que la longitud de las palabras código, ℓ, sea tal que:

$$2^{\ell} = \alpha^n \longleftarrow \texttt{total de cadenas a codificar}.$$

Sin embargo, no todas las posibles cadenas que puede generar la fuente son finalmente generadas con la misma probabilidad. Esto se puede aprovechar para disminuir el número de cadenas a codificar, ya que a una cadena muy improbable no merece la pena asignarle un código único, si casi nunca va a ser generada por la fuente.

Se puede observar que al disminuir el número total de cadenas a codificar, disminuye el número de bits necesarios para codificarlas todas de manera únicamente decodificable (sin ambigüedad). Hasta qué punto se puede reducir el número de cadenas a codificar, sin que ninguna cadena generable por la fuente se quede por fuera (lo que causaria un evento de error), depende de la misma fuente: no es necesario codificar exahustivamente todas las cadenas posibles de longitud n, sino solo aquellas que la fuente produce con mayor frecuencia. Observe que asignar un código único a una cadena que nunca es generada por la fuente no aporta en nada a la compresión sin perdida de la fuente, y de hecho resulta en un desperdicio de bits que son reservados para mensajes que nunca son emitidos por la fuente.

En este orden de ideas, una manera cómo se puede implementar la compresión sin perdida de los mensajes de una fuente es omitiendo la codificación de las cadenas que son tan improbables que prácticamente nunca son emitidas por la fuente, reduciendo así el número de bits reservados para codificar todas las posibles cadenas que emite la fuente, al tiempo que minimizando la probabilidad de un evento de error. Entonces, el problema de la compresión sin perdida de una fuente consiste en determinar el conjunto de cadenas más pequeño que acumula toda la probabilidad de entre el conjunto de todos los mensajes generables por la fuente.

Con la noción de conjunto típico, Shannon ideó una estrategia de compresión en este sentido. La estrategia es comprimir únicamente las cadenas típicas que la fuente emite: establecer una función de codificación invertible que mapea el conjunto de cadenas típicas al conjunto de todas las cadenas binarias de longitud $n\,H\,[X]$; mientras que las $\alpha^n - 2^{n\,H[X]}$ cadenas restantes son ignoradas, no porque no sean factibles, sino porque es muy improbable que sucedan. De hecho, si la fuente emitiera una cadena atípica se declararía un error, ya que no existe su codificación. Así, para un valor de n muy grande, se transmitiría la menor cantidad de mensajes que mantienen el error de codificación bajo, reduciendo el número de bits reservados para codificar los mensajes de la fuente.

Este esquema de codificación es un compresor sin perdida confiable en el límite asintótico (cuando n tiende a infinito), porque la probabilidad de un evento de error (que la fuente emita una cadena atípica) se desvanece conforme n se vuelve más grande, debido a la propiedad de probabilidad unitaria del conjunto típico, en el teorema de la propiedad de equipartición asintótica.

Lo que el esquema de codificación de Shannon plantea es que codificar una fuente no consiste en codificar exahustivamente todos los mensajes posibles de longitud n, sino solo aquellos que la fuente emite con una probabilidad distinta de cero. El teorema de codificación de la fuente de Shannon demuestra que el conjunto de cadenas de longitud n, que una fuente sin memoria emite, en el límite asintótico (cuando n tiende a infinito), es el conjunto de cadenas típicas, que es exponencialmente más pequeño que el conjunto de todas las cadenas posibles de longitud n. Observe la figura 3.1.

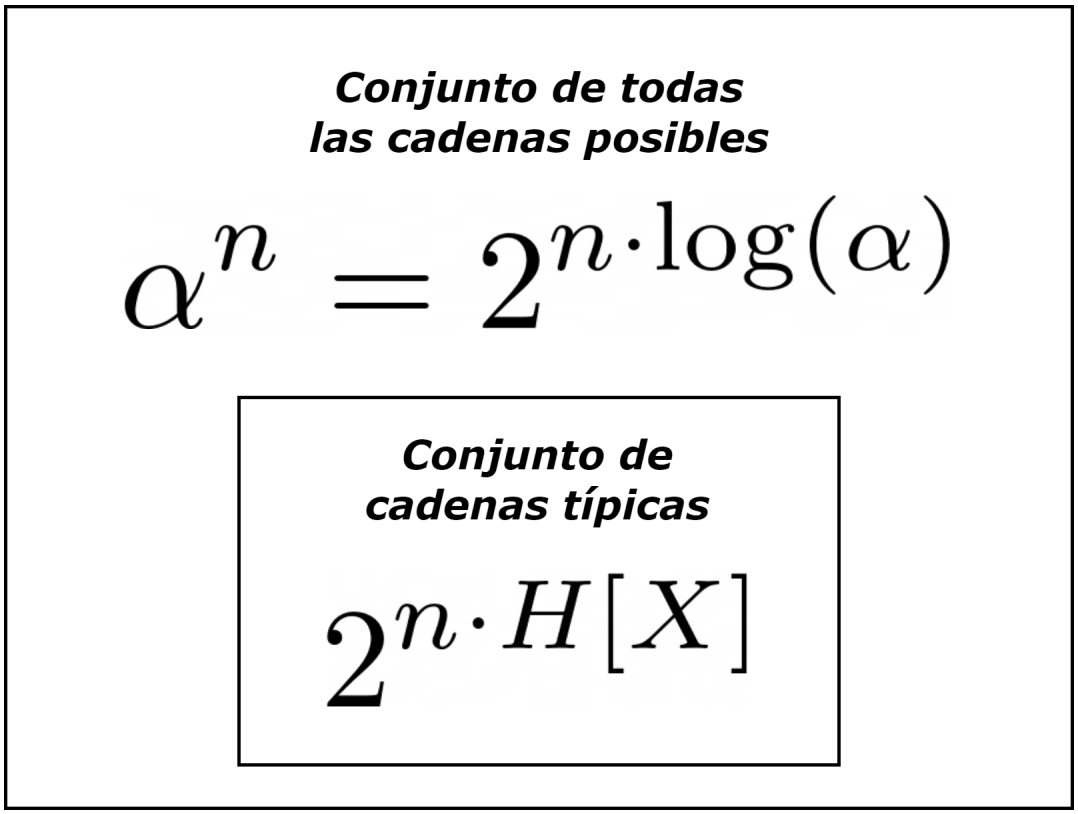

Figura 3.1: Conjunto de cadenas típicas como un subconjunto exponencialmente más pequeño que el conjunto de todas las cadenas posibles de longitud n. Existen un total de $2^{n \cdot H[X]}$ cadenas típicas, las cuales pueden codificarse sin ambigüedad sobre un alfabeto binario con palabras código de longitud $n \cdot H[X]$. En el límite asintótico, el conjunto típico concentra todas la probabilidad de las posibles cadenas que puede generar la fuente. Solamente codificando las cadenas del conjunto típico se codifican la mayoría de las cadenas que la fuente emite, y en el límite asintótico se codifican la totalidad de las cadenas que la fuente emite con alguna probabilidad distinta de cero. De modo que, codificando el conjunto de cadenas típicas no se codifican exahustivamente todas las cadenas posibles de longitud n, pero si todas las cadenas de longitud n que la fuente puede emitir. Este ahorro en los mensajes que la fuente puede transmitir genera un ahorro en el número de bits reservados para codificarlos todos.

3.2.3. Tasa de compresión mínima

La `tasa de compresión` del esquema de compresión de Shannon, calculado como

$$\texttt{tasa de compresión} := \frac{\texttt{número de } \beta\texttt{-bits (noiseless source bits)}}{\texttt{número de } \alpha\texttt{-bits (source bits)}},$$

es igual a la entropía de la fuente:

$$\texttt{tasa de compresión} = \frac{n\,H[X]}{n} = H[X].$$

La tasa de compresión puede interpretarse como el número de bits promedio por símbolo, y consecuentemente, la entropía de Shannon puede interpretarse como la medida del número de bits por símbolo mínimos requeridos, en promedio, para describir una fuente de información [21].

3.2.4. Extensión a fuentes con memoria

Uno puede preguntarse sí la tasa de compresión de datos del esquema de codificación de Shannon es la mejor que se puede lograr: ¿esta tasa es óptima? (se puede lograr una tasa de compresión más baja si no fuera óptima). Más aún, teniendo en cuenta que se asume una fuente sin memoria para demostrar el teorema de equipartición asintótica, es lógico preguntarse sí, de manera general, lo mismo se aplica para una fuente con memoria. Resulta que si. Para extender el teorema de codificación de la fuente de Shannon a una fuente con memoria es preciso formular matemáticamente la manera cómo Shannon demostró la óptimalidad de su esquema de codificación. Para esto se emplea el marco teórico presentado en la referencia [23].

Todo teorema de codificación consiste en dos partes: derivar un esquema y su tasa de compresión (parte directa), para finalmente demostrar sí dicha tasa de compresión es óptima (parte recíproca) [23].

Para el caso del teorema de codificación de la fuente de Shannon, lo que se demuestra es que:

Parte directa: « *Si la tasa de compresión es mayor que la entropía de la fuente, entonces existe un esquema de codificación que puede lograr una compresión de datos sin pérdidas en el sentido de que es posible reducir arbitrariamente la probabilidad de error por una decodificación incorrecta.* »

La formulación matemática de la intuición anterior, presentada en la referencia [21], es la siguiente:

Teorema 3.2.1. Teorema de codificación de la fuente de Shannon (parte directa)
Dada una fuente sin memoria $\mathbb{X}_n$, caracterizada por una variable aleatoria discreta X, definida sobre un alfabeto $\mathcal{A}$, de acuerdo a una función de masa de probabilidad $P_X(x)$:

$$\text{para todo } \epsilon > 0 \text{ y } n \text{ suficientemente grande, existe un código } C(\mathbb{X}_n) \text{ invertible, tal que : } \quad \frac{L}{n} \leq H[X] + \epsilon. \tag{3.5}$$

donde L es la longitud promedio de la palabra código de $C(\mathbb{X}_n)$ y L/n es la tasa de compresión $i.e$ el número de bits promedio por símbolo.

Parte recíproca: « *Si existe un esquema de codificación que puede lograr una compresión de datos sin pérdidas con una probabilidad arbitrariamente pequeña de error de decodificación, entonces la tasa de compresión es mayor que la entropía de la fuente.* »

Teorema 3.2.2. Teorema de codificación de la fuente de Shannon (parte recíproca)
Dada una fuente sin memoria $\mathbb{X}_n$, caracterizada por una variable aleatoria discreta X, definida sobre un alfabeto $\mathcal{A}$, de acuerdo a una función de masa de probabilidad $P_X(x)$. Para todo $v > 0$ y para cualquier esquema de codificación $C(\mathbb{X}_n)$, tal que para n grande se tiene que

$$\frac{L}{n} = H[X] - v,$$

entonces para todo $\delta > 0$, se cumple que

$$P(err) = 1 - \delta - 2^{-\frac{vn}{2}}. \tag{3.6}$$

Para extender los resultados de una fuente sin memoria a una fuente con memoria de tamaño n se puede emplear el formalismo presentado en la referencia [21]. Allí, se aprovecha el hecho de una fuente estacionaría y se muestra que la tasa de entropía, definida en [5, 21], asume el papel de la entropía en el caso de una fuente con memoria, con algunas diferencias sutiles. Para esto se considera la fuente con memoria de tamaño n, $\mathbb{X}_n$, definida sobre el alfabeto $\mathcal{A}$. Luego, cada bloque de k símbolos consecutivos se piensa como un símbolo

generado por una fuente extendida $\mathbb{X}_n^k$, sobre el alfabeto $\mathcal{A}^k$. De este modo, dada la fuente extendida, se define la fuente sin memoria correspondiente $\mathbb{X}_n^{k,*}$, llamada fuente adjunta, que consiste en confinar la memoria de la fuente a k símbolos, de modo que los bloques de longitud k generados por la fuente extendida, $\mathbb{X}_n^k$ son independientes entre sí, la cual constituye una fuente sin memoria en sí misma, definida sobre el alfabeto $\mathcal{A}^k$.

3.2.5. Detalles de la implementación

Algunas consideraciones se derivan del teorema de codificación de la fuente de Shannon. En primer lugar, un procedimiento de codificación, como el utilizado en [21], para demostrar el teorema es muy poco práctico, tal como los propios autores lo mencionan. En segundo lugar, Shannon demostró que la probabilidad de un evento de error se desvanece siempre y cuando se tome n lo suficientemente grande, es decir, si se codifican conjuntamente -infinitamente- largas secuencias de símbolos. Esta es otra cuestión que plantea una serie de problemas prácticos, ya que la compresión sin perdida es únicamente posible, mediante el esquema de compresión de Shannon, bajo una longitud infinita del mensaje, n.

Sin embargo, desde que Shannon demostró sus teoremas de codificación, la comunidad científica se ha centrado en idear maneras de tratar de alcanzar el límite de compresión de Shannon, sin perdida, aún sin disponer de la capacidad del límite asintótico. La idea original de Shannon ha inspirado muchos de los algoritmos de compresión sin perdida actuales, que evitan depender del límite asintótico y que aún así conservan la idea fundamental detrás del teorema de codificación de la fuente de Shannon: implementar la compresión mediante la codificación de bloques de símbolos consecutivos generados por la fuente.

A diferencia de la estrategia teórica de Shannon, en la que se introduce un pequeño error de codificación y éste se hace desaparecer, la estrategia práctica de compresión sin perdida asume que no existe un error de codificación, para luego, con esta restricción, tratar de conseguir el código cuya tasa de compresión sea lo más cercana posible al límite que establece el esquema de codificación Shannon. En el capítulo anterior se definió cómo codificar de manera óptima un alfabeto y extender esa codificación óptima a los mensajes de la fuente. Ahora, se compara esta aproximación con una nueva, que adapta la idea fundamental de Shannon, considerando una fuente que genera bloques de símbolos consecutivos, para saber cuál aproximación puede lograr aproximar de mejor manera el límite de compresión óptimo establecido por Shannon.

3.3. Extensión de una fuente

La codificación óptima de una fuente (codificación de Huffman) es capaz de comprimir, en el caso más general, hasta un β-bit de diferencia más la *divergencia Kullback-Leibler* (entre la distribución de máxima entropía y la distribución verdadera de la fuente) respecto al límite de compresión teórico óptimo, la entropía de Shannon.

Cuando se tiene completo conocimiento de la función de masa de probabilidad, el exceso de β-bits de la *divergencia Kullback-Leibler* es cero. Mientras que el exceso de β-bits por símbolo, debido a la implementación del código, no necesariamente es cero y se acumula en la longitud final de la codificación de un mensaje, al codificar éste mediante la extensión del código, creciendo a una tasa lineal respecto a la longitud del mensaje. No obstante, es posible eliminar parte de ese exceso de β-bits en la compresión final de un mensaje, repartiendo el exceso a lo largo de varios símbolos. Esto se logra considerando la *extensión de la fuente*, la cual es una fuente de símbolos, los cuales consisten en combinaciones de símbolos del alfabeto de la fuente.

Hasta ahora, se ha llevado a cabo la compresión de los mensajes generados por una fuente X por medio de la codificación de los mensajes mediante la extensión de los códigos óptimos de X. Sin embargo, utilizar la extensión de un código óptimo para comprimir la codificación de un mensaje es poco eficiente, debido a varias razones:

1. El exceso de β-bits por símbolo: la implementación de la compresión mediante la codificación de una fuente X causa que los códigos óptimos de la fuente presenten un exceso de β-bits por símbolo, respecto al mínimo teórico global, el cual se acumula para varios símbolos, al extender el código óptimo.

2. No tiene en cuenta las correlaciones entre los símbolos: no se tiene en cuenta que muchas fuentes generan los símbolos con distinta probabilidad dependiendo de los símbolos que se han generado antes, por lo que se pierde certeza acerca de los símbolos siguientes.

En lugar de extender el código óptimo de una fuente X para codificar los mensaje generados por la fuente, se puede extender la fuente en sí misma. Esto permite extender los resultados obtenidos para fuentes de símbolos, en el capítulo anterior, a fuentes de mensajes de longitud k. El formalismo presentado a continuación es el mismo que se utiliza en la referencia [21].

Definición 3.3.1 (*Extensión de una fuente*)**.** Dada una fuente X, que toma valores sobre un alfabeto $\mathcal{A}$, la fuente X^k que toma valores sobre el alfabeto $\mathcal{A}^k$, de acuerdo a una función de masa de probabilidad conjunta,

$$\begin{aligned} P : \mathcal{A}^k &\longrightarrow \mathcal{B}^* \\ x_1 x_2 \dots x_k &\longmapsto P\left(x_1 x_2 \dots x_k\right), \end{aligned}$$

se define como la k-ésima *extensión de la fuente*. Donde $\mathcal{A}^k$ denota el conjunto, de cardinalidad α^k, de todas las combinaciones posibles de k símbolos consecutivos del alfabeto $\mathcal{A}$. La probabilidad conjunta sugiere que la extensión de una fuente puede pensarse como una secuencia de variables aleatorias $X_1 X_2 \dots X_k$. [21]

El concepto de extensión de una fuente generaliza el concepto de fuente de símbolos que se estudió durante el primer capítulo: en ese momento no se había dicho nada acerca de la dependencia entre símbolos, por lo que bastaba con considerar una fuente como una única variable aleatoria discreta X. Al considerar la extensión de una fuente es necesario tener en cuenta las correlaciones entre símbolos, de modo que a cada mensaje, que son los nuevos símbolos de la fuente extendida, le corresponda su propia probabilidad. Dado que la fuente ya no se concibe como un elemento estático, sino que puede considerarse en distintos momentos dentro de una secuencia, la distribución de probabilidad de la fuente cambia dependiendo del momento en el que se encuentra la fuente dentro de la secuencia y del tamaño de la memoria de la fuente.

3.3.1. Función de masa de probabilidad conjunta

Ahora bien, para especificar de manera completa la extensión de una fuente es necesario calcular su función de masa de probabilidad conjunta. Dependiendo del tamaño de la memoria de una fuente, la distribución de masa de probabilidad conjunta se calcula de diferente manera. Para esto, se aprovecha la definición de probabilidad condicional:

$$P(X|Y) = \frac{P(YX)}{P(Y)}, \tag{3.7}$$

de dónde se puede calcular la probabilidad conjunta, en base a la probabilidad condicional como:

$$P(YX) = P(Y) \cdot P(X|Y). \tag{3.8}$$

Esto permite modelar la función de masa de probabilidad conjunta para la extensión de una fuente, mediante el modelo de la fuente, haciendo uso de las probabilidades condicionales.

No en vano, para el modelamiento con Cadenas de Markov se asume que

$$P(X_n|X_1X_2\ldots X_{n-1}) = P(X_n|X_{n-1}), \tag{3.9}$$

entonces, de manera general para cualquier extensión de longitud k de una fuente, se puede calcular la probabilidad conjunta en función de la probabilidades condicionales como

$$P(X_1X_2\ldots X_n) = P(X_n|X_{n-1})\,P(X_{n-1}|X_{n-2})\ldots P(X_3|X_2)\,P(X_2|X_1)\,P(X_1), \tag{3.10}$$

sí la fuente se modela como una Cadena de Markov.

Mientras que, cómo ya se vío, para una fuente sin memoria se tiene que

$$P(X_n|X_1X_2\ldots X_{n-1}) = P(X_n),$$

de donde se sigue que

$$P(X_1X_2\ldots X_n) = P(X_n)\,P(X_{n-1})\ldots P(X_3)\,P(X_2)\,P(X_1)$$

Las anteriores son dos formas de extender una fuente de símbolos: observe que es posible determinar la función de probabilidad conjunta para cadenas de cualquier longitud. De esta manera, la fuente extendida con alfabeto $\mathcal{A}^k$ y su probabilidad conjunta constituyen en conjunto una fuente de símbolos bien definida. Ahora bien, considerando esta fuente como una fuente de símbolos de la vista en el capítulo uno, los resultados del capítulo anterior aplican, que es, la codificación de Huffman sigue siendo óptima, las cotas de compresión aún se mantienen para el nuevo alfabeto, aunque mejoran la compresión respecto a la extensión del código óptimo de la fuente de símbolos sin memoria.

Estos son simplemente dos formas en las que se puede modelar una fuente de símbolos (con memoria). Son dos maneras convenientemente prácticas, ya que las probabilidades condicionales se pueden calcular de manera suficientemente fácil a partir de una colección suficientemente larga de mensajes o simplemente se puede asumir independencia o predefir las probabilidades condicionales.

Ahora bien, uno podría cuestionarse legítimamente sí, efectivamente, las fuentes de símbolos se comportan como procesos Markovianos y pueden modelarse siempre como una cadena de Markov, pues de no ser así, o de aproximar equivocadamente la distribución de los símbolos, se sabe que se estaría incurriendo en un exceso de β-bits en la compresión óptima. Mientras mejor sea el modelo de la fuente y más acertadas sean las probabilidades condicionales que se utilicen, más se disminuye el exceso de β-bits de la divergencia Kullback-Leibler, debido al desconocimiento, en la codificación del alfabeto extendido de la fuente extendida. Por tanto, modelar una fuente como un proceso independiente (o una cadena de Markov) cuando no lo es, conlleva innevitablemente a un exceso de β-bits debido a que la probabilidad conjunta de los símbolos, para la fuente extendida, posiblemente no será una buena estimación de la distribución verdadera del alfabeto extendido, así que la divergencia Kullback-Leibler será significativa y la cota no será tan ajustada.

3.4. Desempeño de la extensión de una fuente

La codificación óptima de la extensión de una fuente X permite comprimir más y mejor un mensaje que la extensión del código óptimo del alfabeto de la fuente X. Esto se debe, a que, cómo se demostró antes, la codificación óptima de cualquier fuente siempre conlleva un exceso de β-bits, debido a la implementación del código. Sin embargo, este exceso de β-bits afecta tanto a la codificación óptima de la extensión de una fuente, como a la codificación óptima de la fuente en sí: este exceso está presente en ambas, pero la diferencia radica en que, si bien, la codificación óptima de la extensión de una fuente inevitablemente presenta dicho exceso de β-bits, éste exceso se está repartiendo a lo largo de múltiples símbolos (el exceso de β-bits se da por bloque de símbolos y no por símbolos individuales) y no se acumula para cada uno de los símbolos que componen un mensaje. Es decir, es la diferencia entre tener tener un solo exceso por un bloque de tamaño k y tener k excesos diferentes, que se suman, por el mismo bloque.

Ejemplo: considere una fuente de símbolos X, definida sobre un alfabeto de dos símbolos, $\mathcal{A} = \{A, B\}$, que es codificado con palabras código compuestas por símbolos del alfabeto $\mathcal{B} = \{0, 1\}$. El código de Huffman (y el único código libre de prefijos posible) para este alfabeto es:

$$C(A) = 0$$
$$C(B) = 1,$$

el cual también es óptimo. Sin embargo, sí se tiene en cuenta la función de masa de probabilidad de la fuente X, sí el símbolo A llega a ser más probable que el símbolo B, o viceversa, es de esperar que el código se pueda comprimir más allá de un β-bit por símbolo y el código anterior ya no parecería tan (globalmente) óptimo. Por ejemplo, suponga, sin perdida de generalidad, que A tiene una probabilidad de $P(A) = 0{,}1$, mientras que B tiene probabilidad de $P(B)\,0{,}9$. En esta situación, se puede calcular el límite de compresión teórico como:

$$\sum_{i=1}^{\alpha} p(a_i)\, log_\beta \left(\frac{1}{p(a_i)}\right) = (0{,}1)\, log_\beta \left(\frac{1}{0{,}1}\right) + (0{,}9)\, log_\beta \left(\frac{1}{0{,}9}\right) = 0{,}469.$$

Así que, teóricamente, es posible mejorar el resultado de la codificación de Huffman (un β-bit por símbolo) casi que el doble. Sin embargo, esta mejora no es factible mediante la implementación de la compresión por medio de la codificación de símbolos individuales. Más aún, si ahora, se extiende esta codificación a un mensaje de longitud m, mediante la extensión del código óptimo (de un β-bit por símbolo), entonces la codificación de un mensaje consiste en la concatenación de m β-bits:

$$C(A \cdot B \cdot A \cdot B \cdot A \cdot B) = C(A) \cdot C(B) \cdot C(A) \cdot C(B) \cdot C(A) \cdot C(B) = 0 \cdot 1 \cdot 0 \cdot 1 \cdot 0 \cdot 1.$$

Es decir, que la longitud del mensaje codificado tiene la misma longitud que el mensaje original. Por lo cual, en este caso, ni siquiera se estaría logrando compresión alguna.

A diferencia de lo que sucede cuando primero se codifica de manera óptima la fuente X y luego se extiende el código óptimo, si primero se extiende la fuente X y luego se codifica de manera óptima la extensión de la fuente, la longitud promedio del mensaje disminuye. Para ilustrar esta idea considere el siguiente ejemplo, el cual ha sido adaptado de la referencia [25]:

> Imagine una larga secuencia de píxeles, en donde cada píxel, individualmente, solo pueden ser o blanco (W) ó negro (B). Ahora, suponga que la probabilidad de que cualquier píxel dado sea blanco es $P(W) = 0{,}3$, mientras que la probabilidad de que sea negro es $P(B) = 0{,}7$. Además, suponga que cada píxel se va a codificar de manera binaria como un cero o un uno (0 ó 1).

En este caso, la fuente de símbolos X consiste en la secuencia de píxeles en sí, para $\mathcal{A} = \{W,B\}$ y donde $\mathcal{B} = \{0{,}1\}$. Sí esta vez, en lugar de extender el código óptimo de la fuente X, primero se extiende la fuente X y luego se codifica de manera óptima (con la codificación de Huffman), y sí se asume que la fuente no tiene memoria (la fuente es un proceso independiente), entonces el resultado es el siguiente:

Codificación de Huffman de la fuente original (sin extender):

$$\begin{aligned} P(W) &= 0{,}7 \qquad C(W) = 1 \\ P(B) &= 0{,}3 \qquad C(B) = 0 \end{aligned} \tag{3.11}$$

Longitud promedio de la palabra código: 1 bits.

Codificación de Huffman de la fuente extendida de bloques de dos símbolos:

$$\begin{aligned} P(WW) &= 0{,}49 \qquad C(WW) = 0 \\ P(WB) &= 0{,}21 \qquad C(WB) = 111 \\ P(BW) &= 0{,}21 \qquad C(BW) = 10 \\ P(BB) &= 0{,}09 \qquad C(BB) = 110 \end{aligned} \tag{3.12}$$

Longitud promedio de la palabra código: 1.81 bits (en lugar de los 2 β-bits para la extensión del código).

Codificación de Huffman de la fuente extendida de bloques de tres símbolos:

$$\begin{aligned} P(WWW) &= 0{,}343 \qquad C(WWW) = 11 \\ P(WWB) &= 0{,}147 \qquad C(WWB) = 100 \\ P(WBW) &= 0{,}147 \qquad C(WBW) = 101 \\ P(BWW) &= 0{,}147 \qquad C(BWW) = 00 \\ P(WBB) &= 0{,}063 \qquad C(WBB) = 0101 \\ P(BWB) &= 0{,}063 \qquad C(BWB) = 0110 \\ P(BBW) &= 0{,}063 \qquad C(BBW) = 0111 \\ P(BBB) &= 0{,}027 \qquad C(BBB) = 0100 \end{aligned} \tag{3.13}$$

Longitud promedio de la palabra código: 2.726 bits (en lugar de los 3 β-bits para la extensión del código).

Codificación de Huffman de la fuente extendida de bloques de cuatro símbolos:

$$\begin{aligned} P(WWWW) &= 0{,}2401 \qquad C(WWWW) = 10 \\ P(WWWB) &= 0{,}1029 \qquad C(WWWB) = 010 \\ P(WWBW) &= 0{,}1029 \qquad C(WWBW) = 011 \\ P(WBWW) &= 0{,}1029 \qquad C(WBWW) = 000 \\ P(BWWW) &= 0{,}1029 \qquad C(BWWW) = 001 \\ P(WWBB) &= 0{,}0441 \qquad C(WWBB) = 11001 \\ P(WBWB) &= 0{,}0441 \qquad C(WBWB) = 11010 \\ P(BWWB) &= 0{,}0441 \qquad C(BWWB) = 11110 \\ P(WBBW) &= 0{,}0441 \qquad C(WBBW) = 11011 \\ P(BWBW) &= 0{,}0441 \qquad C(BWBW) = 11101 \\ P(BBWW) &= 0{,}0441 \qquad C(BBWW) = 11100 \\ P(WBBB) &= 0{,}0189 \qquad C(WBBB) = 1111111 \\ P(BWBB) &= 0{,}0189 \qquad C(BWBB) = 110000 \\ P(BBWB) &= 0{,}0189 \qquad C(BBWB) = 110001 \\ P(BBBW) &= 0{,}0189 \qquad C(BBBW) = 111110 \\ P(BBBB) &= 0{,}0081 \qquad C(BBBB) = 1111110 \end{aligned} \tag{3.14}$$

Longitud promedio de la palabra código: 3.56720 bits (en lugar de los 4 β-bits para la extensión del código).

Codificación de Huffman de la fuente extendida de bloques de cinco símbolos:

$$
\begin{aligned}
P(WWWWW) &= 0{,}16807 & C(WWWWW) &= 111\\
P(WWWWB) &= 0{,}07203 & C(WWWWB) &= 1100\\
P(WWWBW) &= 0{,}07203 & C(WWWBW) &= 1101\\
P(WWBWW) &= 0{,}07203 & C(WWBWW) &= 1010\\
P(WBWWW) &= 0{,}07203 & C(WBWWW) &= 1011\\
P(BWWWW) &= 0{,}07203 & C(BWWWW) &= 000\\
P(WWWBB) &= 0{,}03087 & C(WWWBB) &= 01001\\
P(WWBWB) &= 0{,}03087 & C(WWBWB) &= 10000\\
P(WBWWB) &= 0{,}03087 & C(WBWWB) &= 10001\\
P(BWWWB) &= 0{,}03087 & C(BWWWB) &= 01110\\
P(WWBBW) &= 0{,}03087 & C(WWBBW) &= 01111\\
P(WBWBW) &= 0{,}03087 & C(WBWBW) &= 01100\\
P(BWWBW) &= 0{,}03087 & C(BWWBW) &= 01101\\
P(WBBWW) &= 0{,}03087 & C(WBBWW) &= 01010\\
P(BWBWW) &= 0{,}03087 & C(BWBWW) &= 01011\\
P(BBWWW) &= 0{,}03087 & C(BBWWW) &= 10010\\
P(BBBWW) &= 0{,}01323 & C(BBBWW) &= 001001\\
P(BBWBW) &= 0{,}01323 & C(BBWBW) &= 010000\\
P(BWBBW) &= 0{,}01323 & C(BWBBW) &= 010001\\
P(WBBBW) &= 0{,}01323 & C(WBBBW) &= 001110\\
P(BBWWB) &= 0{,}01323 & C(BBWWB) &= 001111\\
P(BWBWB) &= 0{,}01323 & C(BWBWB) &= 001100\\
P(WBBWB) &= 0{,}01323 & C(WBBWB) &= 001101\\
P(BWWBB) &= 0{,}01323 & C(BWWBB) &= 001010\\
P(WBWBB) &= 0{,}01323 & C(WBWBB) &= 001011\\
P(WWBBB) &= 0{,}01323 & C(WWBBB) &= 100110\\
P(WBBBB) &= 0{,}00567 & C(WBBBB) &= 10011101\\
P(BWBBB) &= 0{,}00567 & C(BWBBB) &= 0010000\\
P(BBWBB) &= 0{,}00567 & C(BBWBB) &= 0010001\\
P(BBBWB) &= 0{,}00567 & C(BBBWB) &= 10011110\\
P(BBBBW) &= 0{,}00567 & C(BBBBW) &= 10011111\\
P(BBBBB) &= 0{,}00243 & C(BBBBB) &= 10011100
\end{aligned}
\tag{3.15}
$$

Longitud promedio de la palabra código: 4.44498 bits (en lugar de los 5 β-bits para la extensión del código).

Es posible notar que codificando de manera óptima las extensiones de la fuente X, en lugar de extender el código óptimo de la fuente X, la longitud promedio de la palabra código se reduce. De hecho, se puede ver que conforme más se aumenta la longitud del bloque, más significativa es la diferencia en términos de β-bits. Esto se debe a que el exceso de β-bits por símbolo, que se acumula en la extensión del código óptimo con cada caracter que aumenta el bloque, por el contrario, se reparte a lo largo de cada vez más símbolos en la codificación óptimo de la extensión de la fuente: Se observa que la diferencia entre las longitudes promedio óptimas de la $(k+1)$-ésima y la k-ésima extensión de la fuente es menor que uno. Por otro lado, el cambio observado en la extensión del código óptimo es 1.

Así que, la forma razonable de comprimir un mensaje es extendiendo la fuente en lugar de extender el código, al menos en el caso de una fuente sin memoria (un proceso independiente). Queda la duda de qué sucedería sí el proceso no fuera independiente, sino Markoviano. Esto es lo que se estudia a continuación, comparando las *tasas de cambio de la longitud promedio de la palabra código óptima* respecto a cambios en una unidad de la longitud de la extensión, bajo conocimiento de la distribución de la fuente.

Ahora se extiende la comparación previa al caso de una fuente Markoviana. Además se considera hasta la 10-ésima extensión (extensión de diez símbolos) de las fuentes, para ver lo qué sucede a más largo plazo. Para esto, esta vez, se construyen dos procesos artificiales: un proceso independiente (una fuente sin memoria) y una cadena de Markov (una fuente con memoria de tamaño uno), los cuales serán comparables entre sí por medio de sus probabilidades, y, a su vez, se compararán con la extensión del código óptimo para esas mismas probabilidades. Ahora bien, cada una de las dos fuentes calcula la función de masa de probabilidad conjunta de la extensión del alfabeto de manera distinta: la fuente sin memoria la calcula utilizando las probabilidades marginales de los símbolos, mientras que la fuente con memoria lo hará considerando dependencias de orden uno. Para esto se inicializan las probabilidades marginales y las probabilidades condicionales de orden uno en algún valor por defecto. No en vano, para garantizar que las probabilidades de la cadena de Markov, el procesos independiente y la extensión del código sean comparables entre sí, se lleva a cabo el siguiente procedimiento: primero, se fijan las probabilidades condicionales, $P(W|B)$ y $P(B|W)$, de la fuente con memoria (proceso de Markov) y a partir de estas se calculan las probabilidades marginales, $P(W)$ y $P(B)$, por medio del *teorema de Bayes*: el *teorema de Bayes* establece que:

$$P(W|B) = \frac{P(BW)}{P(B)} = \frac{P(B|W) \cdot P(W)}{P(B)},$$

de donde se sigue que

$$P(W|B)P(B) = P(B|W) \cdot P(W).$$

Sin embargo, dado que $P(B) = 1 - P(W)$, lo anterior puede reescribirse como

$$P(W|B)(1 - P(W)) = P(B|W) \cdot P(W).$$

Así que

$$P(W|B) - P(W|B) \cdot P(W) = P(B|W) \cdot P(W).$$

De donde se sigue que

$$P(W) = \frac{P(W|B)}{P(W|B) + P(B|W)}.$$

De esta manera, a partir de las probabilidades condicionales, $P(W|B)$ y $P(B|W)$, se pueden determinar consistentemente las probabilidades marginales, $P(W)$ y $P(B)$, correspondientes, para construir el proceso independiente y la extensión del código, de modo que sean comparables entre sí. Observe que, en esta comparación, las fuentes son construidas manualmente, así que se tiene completo conocimiento de la distribución de la extensión de la fuente. Por lo tanto, se descarta el exceso de β-bits debido al desconocimiento de la distribución de la fuente (Divergencia Kullback Leibler), es decir, la longitud promedio de la palabra código del código de Huffman de la extensión de la fuente está a lo sumo a un β-bit del mínimo teórico óptimo, la entropía de Shannon de la extensión de la fuente.

Ahora bien, una vez sabiendo cómo definir las fuentes de manera comparable, lo que resta es definir el procedimiento mediante el cuál las tres codificaciones se van a comparar: la extensión del código óptimo, la codificación óptima de la extensión de la fuente sin memoria y la codificación óptima de la extensión de la fuente con memoria (memoria de tamaño uno). Lo que se hará es comparar cómo cambia la longitud promedio de la palabra código de cada una de las codificaciones, al variar el tamaño del bloque.

De manera general, el proceso que se llevará a cabo es el siguiente:

1. Fijar algunos valores de las probabilidades condicionales, $P(A|B)$ y $P(B|A)$.

2. Calcular las cantidades $P(A)$, $P(B)$, $P(A|A)$ y $P(B|B)$, como:

   - $P(A) = P(A|B) / [P(A|B) + P(B|A)]$.
   - $P(B) = 1 - P(A) = P(B|A) / [P(A|B) + P(B|A)]$.
   - $P(A|A) = 1 - P(B|A)$.
   - $P(B|B) = 1 - P(A|B)$.

3. Computar la extensión de alfabeto $\mathcal{A}$ de las fuentes, $\mathcal{A}^k$, para $k = 1,2,...,10$.

4. Calcular las funciones de masa de probabilidad conjunta para los símbolos de la extensión del alfabeto como:

   a) Fuente sin memoria (Proceso independiente):

   $$P(x_1 x_2 \dots x_n) = P(x_n) P(x_{n-1}) \dots P(x_3) P(x_2) P(x_1).$$

   b) Fuente con memoria de tamaño uno (Cadena de Markov):

   $$P(x_1 x_2 \dots x_n) = P(x_n|x_{n-1}) P(x_{n-1}|x_{n-2}) \dots P(x_3|x_2) P(x_2|x_1) P(x_1).$$

5. Codificar la k-ésima extensión de las fuentes, mediante la codificación de Huffman, para $k = 1,2,\dots,10$.

6. Calcular la longitud promedio de la palabra código como:

   $$\sum_{x_1 x_2 \dots x_k} [P(x_1 x_2 \dots x_k) \cdot \ell_{\texttt{huffman}}(x_1 x_2 \dots x_k)],$$

   donde $\ell_{\texttt{huffman}}(x_1 x_2 \dots x_k)$ denota la longitud de la palabra código asignada al bloque $x_1 x_2 \dots x_k$ en el código de Huffman de la extensión de la fuente.

7. Graficar la longitud promedio de la palabra código de la codificación de Huffman de la k-ésima extensión de la fuente, para ambas fuentes, y la k-ésima extensión del código de Huffman para el alfabeto $\mathcal{A}$, para cada uno de los tamaños de los bloques de la extensión: $k = 1,2,\dots,10$.

8. Graficar la *tasa de cambio de la longitud promedio de la palabra código*, respecto al tamaño del bloque, entre la 10-ésima y 9-ésima extensión de las fuente, para ambas fuentes (con memoria y sin memoria), para distintos valores de las probabilidades, $P(A|B)$, $P(B|A)$ y $P(A)$.

Observe que, aquí se lleva a cabo la comparación para un alfabeto $\mathcal{A}$ de tamaño $\alpha = 2$. Este es el caso más simple en el que se puede identificar en qué casos la codificación óptima de la extensión de una fuente (con memoria o sin memoria) se desempeña mejor que la extensión del código óptimo. Si bien, no se puede garantizar que las conclusiones que se obtienen del análisis a continuación se extienden a cualquier alfabeto de tamaño α, sí que se puede generalizar el análisis a cualquier alfabeto, particularmente, se puede generalizar la *tasa de cambio de entropía* y este concepto se puede visualizar de la misma manera como a continuación.

3.4.1. Tasa de cambio de la entropía

A continuación, se presenta la comparación entre la extensión del código y la extensión de la fuente (con memoria y sin memoria), mediante el proceso que se describió antes, 1.

La línea sólida verde, representa, en cada Figura, la longitud de cada bloque mediante su codificación por medio de la extensión del código. Las líneas sólidas de color azul y naranja representan la longitud promedio del bloque en la fuente extendida, para la fuente sin memoria y con memoria de tamaño uno, respectivamente.

En cada una de las Figuras 3.2, 3.3, 3.4 y 3.5, se grafica la entropía de Shannon para la extensión de la fuente como una línea punteada con su respectivo color. Puede observar que en todos los casos la línea punteada se encuentra debajo de la línea sólida, que representa la longitud promedio de la palabra código de la codificación de Huffman de la extensión de la fuente. Sin embargo, también se observa que ambas curvas del mismo color (sólida y punteada) aparecen muy cercanas entre sí, lo cual sugiere que la longitud promedio de la codificación de Huffman es una buena aproximación de la entropía de la fuente. Lo cual es de esperarse ya que no existe exceso de β-bits debido al desconocimiento de la distribución de la extensión de la fuente, pues las fuentes se definieron de manera artificial: solo existe exceso de β-bits debido a la implementación del código (un exceso de a lo sumo un β-bit). Sin embargo, precisamente, este exceso de β-bits decrece conforme la longitud de la extensión de la fuente se hace más grande. Así, se ve que sí al principio existía una diferencia entre las curvas sólidas y punteadas del mismo color, al final, la diferencia tiende a desaparecer, pues se elimina el exceso de β-bits debido a la implementación del código, al extender la fuente para bloques cada vez más grandes.

De ahora en adelante, se denota como S_m la aproximación de la entropía de Shannon, por medio de la longitud promedio de la palabra código de la codificación de Huffman de la fuente extendida de bloques de longitud m, entonces se tiene que

$$\frac{\Delta S_m}{\Delta m} = \frac{S_m - S_{m-1}}{m-(m-1)} = S_m - S_{m-1} = \Delta S_m \tag{3.16}$$

es una aproximación de la *tasa de cambio de la entropía con respecto al cambio en una unidad del tamaño del bloque*, m. [11]

En las Figuras 3.6 y 3.7 se puede ver que el límite de la tasa de cambio de la longitud promedio óptima, ΔS_m es uno. Es decir, que para una fuente sin memoria y una fuente con memoria de tamaño uno, el cambio en la longitud promedio de la palabra código entre dos extensiones de las fuentes consecutivas nunca es mayor que uno, para cualquier distribución de las fuentes.

En la Figura 3.6 se puede comprobar algo que ya se podía notar en las Figuras 3.2, 3.3, 3.4 y 3.5, que era que la línea que correspondía a la extensión de la fuente sin memoria era una línea recta, con una tasa de cambio de la entropía, ΔS_m, siempre menor o igual que la del código extensión. Aquí podemos notar que así es, y que en el peor de los casos, cuando los símbolos son equiprobables, ambas líneas tienen la misma pendiente, i.e son iguales.

En la Figura 3.7, la línea $P(A|B) = P(B|A)$ corresponde al caso de equiprobabilidad para la fuente sin memoria, ya que, sobre esa línea, $P(A) = P(B) = 0{,}5$: $P(A|B) = P(B|A)$, sobre esta línea sucede lo siguiente:

$$P(A) = \frac{P(A|B)}{P(A|B) + P(B|A)} = \frac{P(A|B)}{2P(A|B)} = \frac{1}{2}. \tag{3.17}$$

Mientras que, la línea $P(A|B) = 1 - P(B|A)$ corresponde al caso de independencia para la fuente con memoria, es decir, es la misma fuente sin memoria, ya que, sobre esa línea, $P(A) = P(A|B)$ y $P(B) = P(B|A)$:

$$P(A) = \frac{P(A|B)}{P(A|B) + P(B|A)} = \frac{P(A|B)}{[1 - P(B|A)] + P(B|A)} = \frac{P(A|B)}{1} = P(A|B). \tag{3.18}$$

También se cumple que $P(B) = P(B|A)$, en este caso.

Observe que es posible identificar las tasas de cambio exhibidas en las Figuras 3.2, 3.3, 3.4 y 3.5 en las Figuras 3.6 y 3.7, mediante los valores de las probabilidades condicionales, $P(A|B)$ y $P(B|A)$ (y las probabilidades marginales $P(A)$). De hecho, en las Figura 3.6 y 3.7 puede identificar todas las tasas de cambio que caracterizan todas las gráficas posibles del tipo 3.2 que se pueden lograr. Con esto, usted puede hacerse una idea de cómo se comparan las tasas de cambio para la fuente sin memoria y la fuente con memoria de tamaño uno.

A continuación, se generalizan algunas observaciones: de manera general, la tasa de cambio de la entropía de la fuente con memoria, cuando se tiene completo conocimiento, es máxima sí no se tienen correlaciones. En la figura 3.3, la entropía de la fuente con memoria es exactamente igual a la entropía de la fuente sin memoria. [11]. Y de manera general, la tasa de cambio de la entropía siempre es no negativa [11]:

$$\Delta S_m \geq 0.$$

Tal cómo se observa en las Figuras 3.6 y 3.7 para este caso particular. Se observa que, en general, la extensión de la fuente se desempeña mejor que la extensión del código para codificar bloques de símbolos, debido a que la tasa de cambio de la entropía de la extensión de una fuente siempre es menor que la tasa de cambio de la extensión del código, siempre y cuando se tiene completo conocimiento de la distribución de la fuente. Sin embargo, en el caso de la fuente sin memoria, cuando todos los símbolos son equiprobables, es mejor ópción emplear la extensión del código que la extensión de la fuente, ya que generalmente la extensión de la fuente es más costosa.

Ahora bien, observe que si se tiene conocimiento completo de la función de masa de probabilidad de la fuente, entonces el procedimiento 1 se puede generalizar para cualquier alfabeto $\mathcal{A}$, de tamaño α arbitrario, para determinar cuándo la extensión del código se desempeña mejor que la extensión de la fuente, o incluso para comparar entre sí la extensión de dos fuentes, mediante el concepto de tasa de cambio de la entropía, ΔS_m.

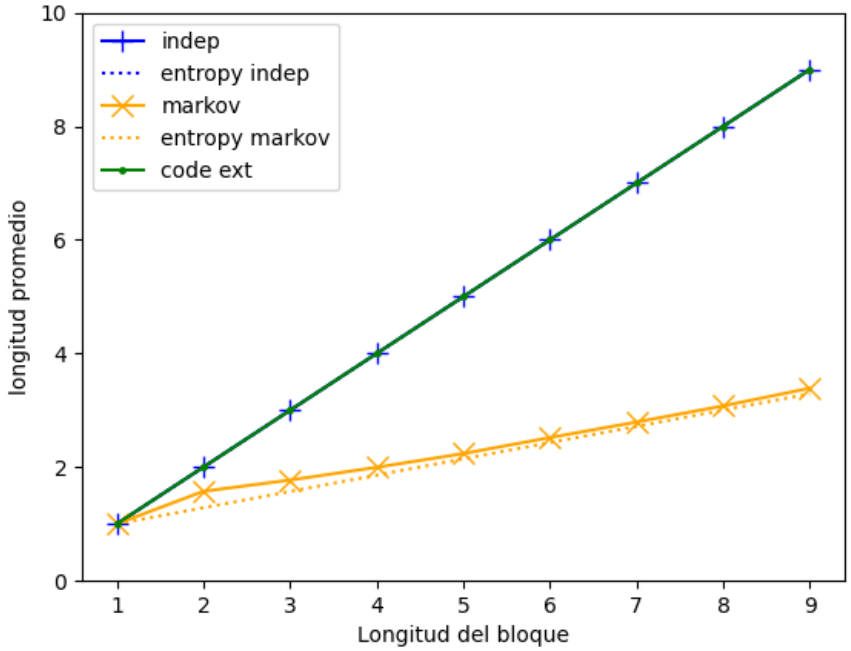

Figura 3.2: $P(A) = 0{,}5, P(B) = 0{,}5,\ P(A|A) = 0{,}05,\ P(B|A) = 0{,}95,\ P(A|B) = 0{,}95,\ P(B|B) = 0{,}05$

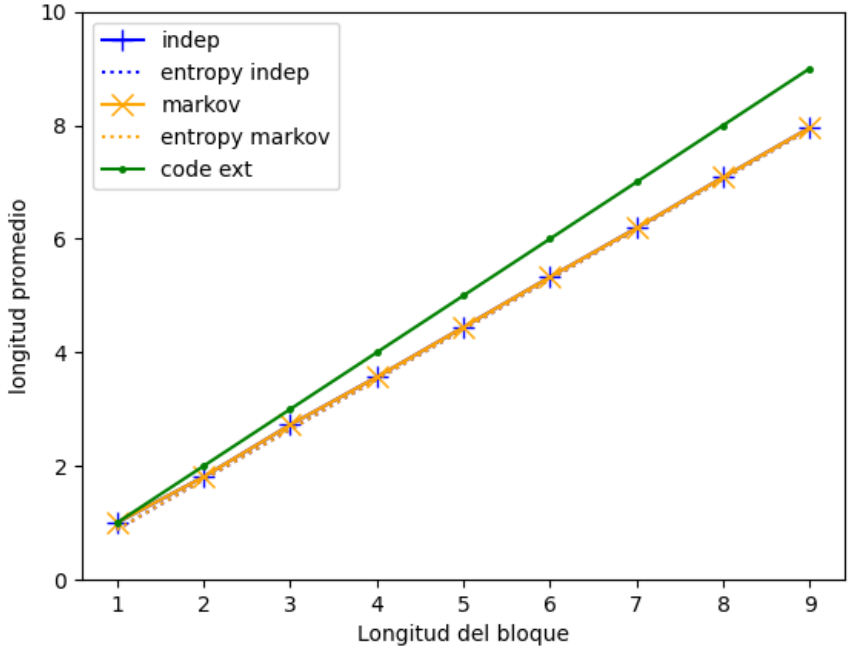

Figura 3.3: $P(A) = 0{,}7, P(B) = 0{,}3,\ P(A|A) = 0{,}7,\ P(B|A) = 0{,}3,\ P(A|B) = 0{,}7,\ P(B|B) = 0{,}3$

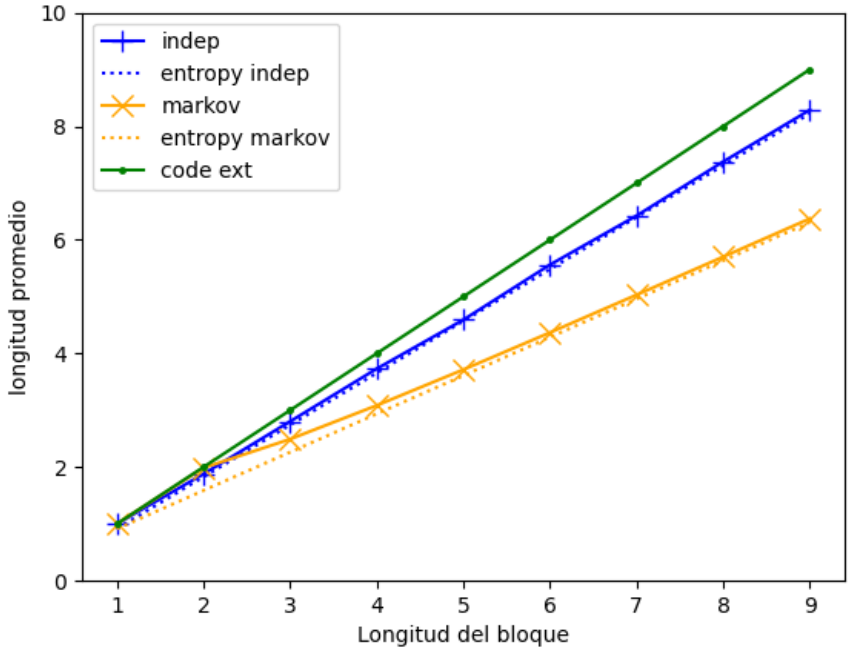

Figura 3.4: $P(A) = 0{,}329, P(B) = 0{,}671$, $P(A|A) = 0{,}001$, $P(B|A) = 0{,}99$, $P(A|B) = 0{,}49$, $P(B|B) = 0{,}51$

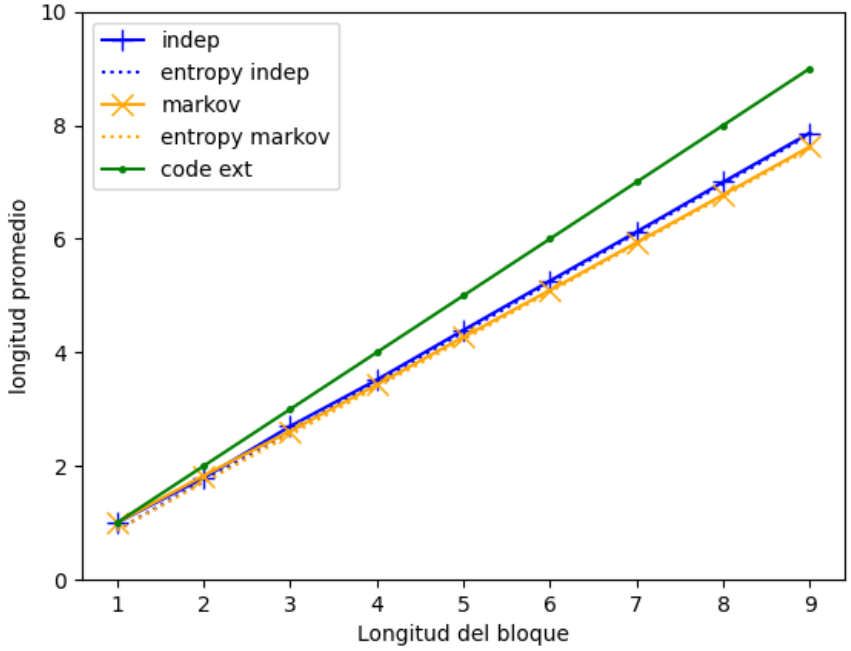

Figura 3.5: $P(A) = 0{,}708, P(B) = 0{,}292$, $P(A|A) = 0{,}65$, $P(B|A) = 0{,}35$, $P(A|B) = 0{,}85$, $P(B|B) = 0{,}15$

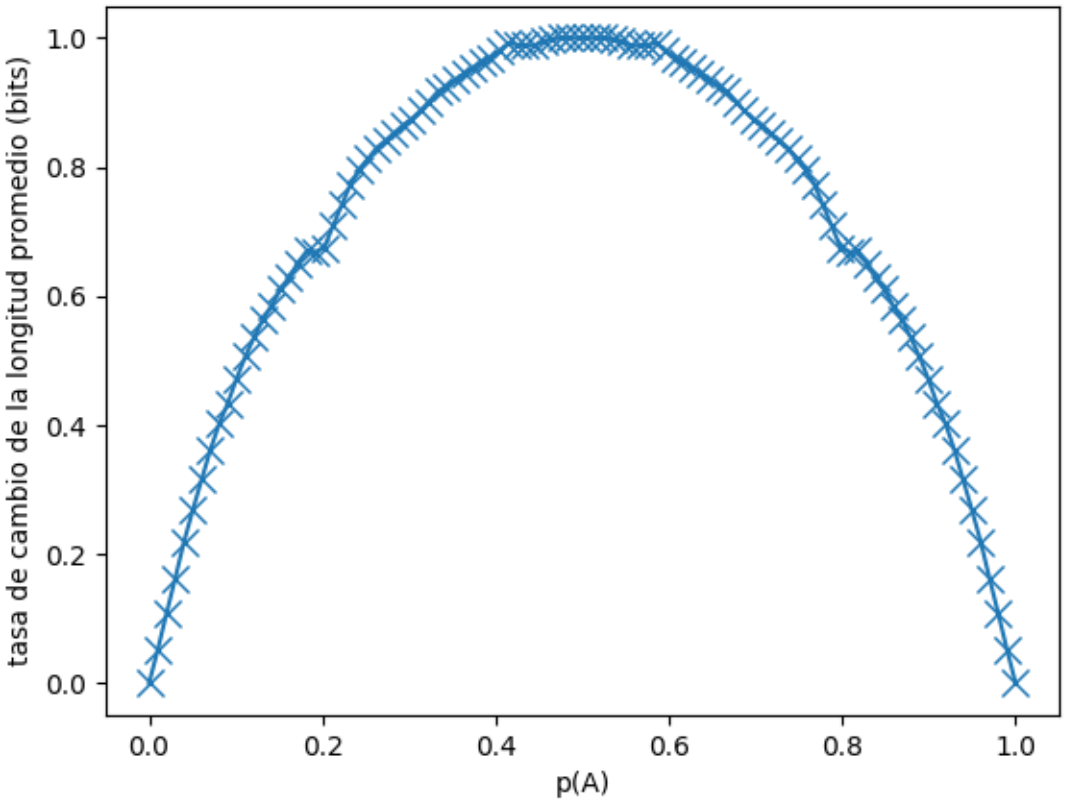

Figura 3.6: Gráfica de la diferencia $\Delta S_9 = S_9 - S_8$, aproximación de la tasa de cambio de la longitud promedio óptima, para la extensión de una fuente sin memoria (proceso independiente) de un alfabeto de dos símbolos $\mathcal{A} = \{A,B\}$, para distintos valores de la probabilidad marginal, $P(A)$. En esta Figura se consideran todas las distribuciones de la extensión de la fuente posibles, ya que solo con variar el parámetro $P(A)$, sobre todo su dominio, se están considerando todas las distribuciones de probabilidad conjunta de las extensiones de la fuente, pues estas solo dependen de las probabilidades marginales, $P(A)$ y $P(B)$.

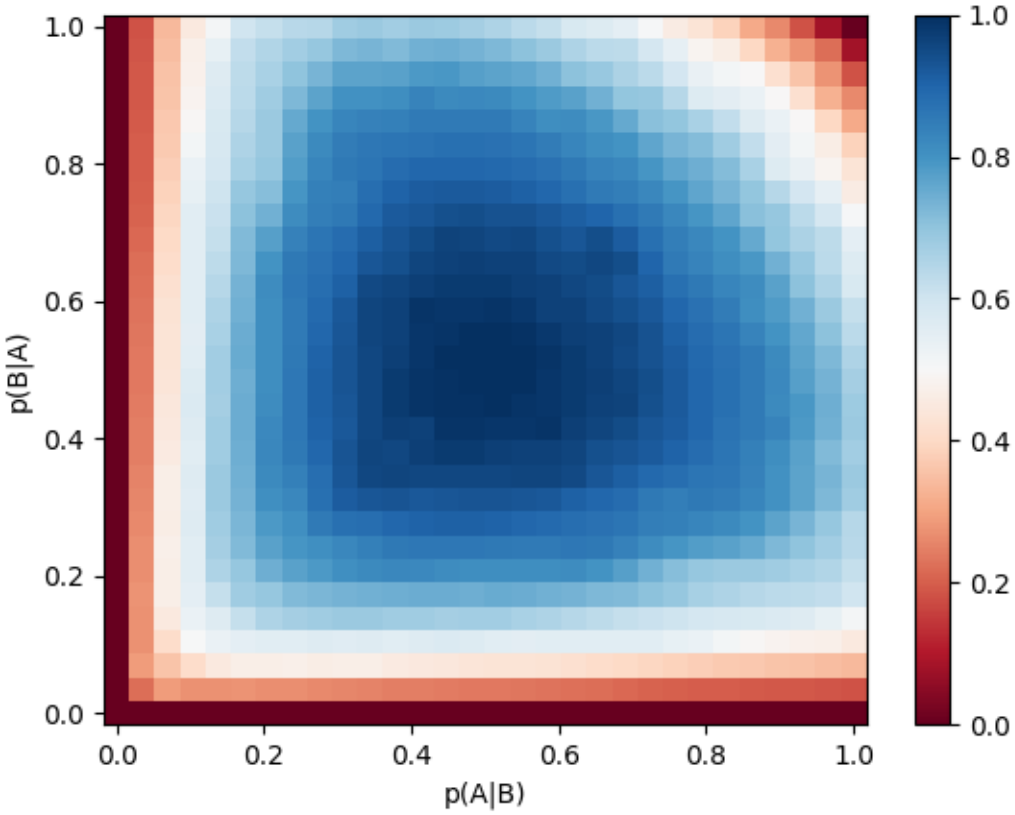

Figura 3.7: Gráfica de la diferencia $\Delta S_9 = S_9 - S_8$, aproximación de la tasa de cambio de la longitud promedio óptima, para la extensión de una fuente con memoria de tamaño uno (cadena de Markov) de un alfabeto de dos símbolos $\mathcal{A} = \{A,B\}$, para distintos valores de las probabilidades condicionales, $P\left(A|B\right)$ y $P\left(B|A\right)$. Esta vez, para considerar todas las distribuciones de la extensiones de la fuente posibles es necesario variar los dos parámetros $P(A|B)$ y $P(B|A)$ en todo su dominio. Los valores de la tasa de cambio de la longitud promedio óptima se visualizan por medio del mapa de color: color más rojos representan valores de la tasa de cambio más cercano a cero y el color más azúl representa valores de la tasa de cambio más cercanas a uno.

Capítulo 4

Aplicaciones

En este capítulo se presentan algunas aplicaciones de la compresión de cadenas al aprendizaje automático de máquina. Para esto, se presenta un nuevo algoritmo de codificación, el cual emplea una aproximación distinta a la de los algoritmos que se han presentado hasta ahora, para comprimir toda cadena de símbolos. Luego, se definen algunas medidas de similaridad entre cadenas de símbolos, en base a algunas medidas de *información*, que están bien definidas en términos de los tamaños de las cadenas comprimidas. Finalmente se ilustra la utilidad de estas medidas de compresión en el aprendizaje automático, generando un Agrupamiento Jerárquico del lenguaje natural. Los resultados finales son comparados con los de sus artículo originales y con los de algunas referencias que utilizaron métodos más tradicionales para resolver el problema de clasificación.

4.1. Codificación de diccionario

La codificación de diccionario consiste en construir un diccionario de patrones, almacenando cada patrón junto con una dirección de referencia: un mapeo que asigna desde el espacio de bloques de caracteres de cierta longitud (que son llamados *patrones*) a un espacio de códigos (que son llamados *referencias*). A este mapeo se le denomina *diccionario*, y constituye un tipo de codificación en sí. Se codifica una cadena reemplazando todas las repeticiones de un patrón por su respectiva referencia en el diccionario. De esta manera, el código consiste en un conjunto de referencias concatenadas, cada una de las cuales direccionan a un patrón en especifico. Para decodificar una cadena codificada por diccionario, se reemplaza de vuelta la referencia por su respectivo patrón de entrada en el diccionario.

4.1.1. Codificación de diccionario estático

Un diccionario puede ser estático, es decir, predefinido y fijo. Este tipo de diccionario comúnmente implementa estándares. Un ejemplo de esto son las codificaciones ASCII (o UNICODE), que codifican cada símbolo del teclado como una combinación diferente de un número fijo bits: cada vez que se presiona una tecla el caracter de la tecla es almacenado en memoria en un segmento de longitud fija de bits.

4.1.2. Codificación de diccionario semi-adaptativo

Un diccionario también puede ser semi-adaptativo, es decir, hecho a mano, especificamente. para la cadena que se codifica. Para eso, primero se procesa la cadena para extraer los patrones y construir el diccionario. Luego, se procesa la cadena para codificarla. No obstante, para decodificar la cadena es necesario el diccionario con el cual se codificó, por lo que, además del código, también se debe transmitir el diccionario.

4.1.3. Codificación de diccionario adaptativo

Un diccionario adaptativo se construye al tiempo que se codifica una cadena. Como tál, es único para cada cadena, sin embargo, no es necesario trasmitirlo junto con el código para su decodificación, ya que se construye con un algoritmo que se aplica sobre la cadena, de modo que el diccionario está implícito en la propia cadena. De esta manera, el decodificador, que consiste en un algoritmo que aplica la función inversa al codificador, operando apropiadamente sobre el código, puede deshacer la codificación para obtener nuevamente la cadena original, sin necesidad de un diccionario explícito, utilizando solo la cadena de símbolos del código.

4.2. *Codificación por bloques*

Una codificación del alfabeto $\mathcal{A}$ posibilita la codificación de palabras, como una codificación de bloques de símbolos del alfabeto $\mathcal{A}$. En una codificación por bloques se divide una cadena de símbolos del alfabeto $\mathcal{A}$ en bloques de una longitud dada, que pueden llegar a sobrelaparse, y en donde a cada uno, dependiendo de la codificación del alfabeto, le corresponde un bloque de símbolos del alfabeto $\mathcal{B}$, compuesto por la concatenación de las palabras código de los símbolos que componen el bloque.

La codificación por bloques puede formularse como un conjunto de pares (X_i, Y_i) con $i = 1,2, ..., M$, donde

- X_i es una secuencia de símbolos del alfabeto $\mathcal{A}$ (palabra).
- Y_i es una secuencia de símbolos del alfabeto $\mathcal{B}$ (palabra codificada).

En realidad, la codificación por bloques consiste en considerar cada bloque como un símbolo. Luego, cada bloque se reemplaza por su respectivo código. Existen diferentes tipos de codificación por bloques, dependiendo del tamaño de los bloques:

1. Código bloque-a-bloque: este tipo de código asigna a bloques de símbolos del mensaje, del mismo tamaño, palabras código del mismo tamaño.
2. Código bloque-a-variable: este tipo de código asigna a bloques de símbolos del mensaje, del mismo tamaño, palabras código de tamaños distintos.
3. Código variable-a-bloque: este tipo de código asigna a bloques de símbolos del mensaje, de diferente tamaño, palabras código del mismo tamaño.

4.2.1. Codificación *bloque-a-bloque*

En la codificación *bloque-a-bloque*, la longitud de los bloques es fija tanto en el alfabeto $\mathcal{A}$ como en el alfabeto $\mathcal{B}$.

Codificación del alfabeto:

$$\ldots \quad (000, AA) \qquad (111, TT) \qquad (001, GG) \qquad (011, UU) \qquad (100, AT) \qquad (110, AG) \quad \ldots$$

Ejemplo de la codificación por bloques:

$$000111001011100110 \equiv \boxed{000}_{AA}\,\boxed{111}_{TT}\,\boxed{001}_{GG}\,\boxed{011}_{UU}\,\boxed{100}_{AT}\boxed{110}_{AG}$$

4.2.2. Codificación *bloque-a-variable*

En la codificación *bloque-a-variable* la longitud de los bloques sobre el alfabeto $\mathcal{A}$ permanece fija, mientras que la longitud de los bloques sobre el alfabeto $\mathcal{B}$ puede variar.

Codificación del alfabeto:

$$\ldots \quad (000, A) \qquad (111, T) \qquad (001, G) \qquad (011, U) \qquad (100, AA) \qquad (110, TT) \quad \ldots$$

Ejemplo de la codificación por bloques:

$$000111001011100110 \equiv \boxed{000}_{A}\,\boxed{111}_{T}\,\boxed{001}_{G}\,\boxed{011}_{U}\,\boxed{100}_{AA}\boxed{110}_{TT}$$

4.2.3. Codificación *variable-a-bloque*

En este caso, la longitud de los bloques pueden ser variables sobre el alfabeto $\mathcal{A}$, mientras que permanecen fijos sobre el alfabeto $\mathcal{B}$.

Codificación del alfabeto:

$$\ldots \quad (00011, AA) \qquad (1, TT) \qquad (001, GG) \qquad (011100, UU) \qquad (110, AT) \quad \ldots$$

Ejemplo de la codificación por bloques:

$$000111001011100110 \equiv \boxed{00011}_{AA}\,\boxed{1}_{TT}\,\boxed{001}_{GG}\,\boxed{011100}_{UU}\boxed{110}_{AT}$$

4.2.4. Codificación *variable-a-variable*

En la codificación *variable-a-variable* la longitud de los bloques pueden ser variables en el alfabeto $\mathcal{A}$ y también pueden ser variables en el alfabeto $\mathcal{B}$. Una codificación *variable-a-variable* se puede implementar como la combinación de una codificación *variable-a-bloque* y luego *bloque-a-variable*.

4.3. Algoritmo de codificación LZ77

El algoritmo de codificación LZ77 es un algoritmo de codificación sin perdida variable-a-bloque formulado originalmente por los autores Abraham Lempel y Jacob Ziv en su artículo *A universal algorithm for sequential data compression*, [25], en 1977. En esta sección se definen la funcionalidad del algoritmo y se presenta un ejemplo de codificación y decodificación.

El algoritmo de codificación LZ77 es un algoritmo de compresión sin perdida variable-to-block. Para la codificación de cadenas el algoritmo LZ77 emplea una codificación de diccionario adaptativo.

4.3.1. Algoritmo de compresión universal

El algoritmo de codificación LZ77 lleva a cabo un procesamiento de una cadena de caracteres para extraer de ello las correlaciones entre símbolos. De esta manera es que el algoritmo de codificación LZ77 logra comprimir una cadena. En el árticulo [19], se demuestra que, al menos para cadenas de Markov ocultas, sí la cadena generada por uno de esos procesos, entonces el tamaño de la cadena comprimida es con alta probabilidad cercana a la entropía del proceso y la probabilidad puede hacerse arbitrariamente más grande para distancias aún más cercanas, aumentando el tamaño de la cadena. Esta propiedad, es la que lo autores del artículo [19] denominan *buena compresión*. En ese sentido, se ha demostrado que el algoritmo de codificación LZ77 es un buen compresor.

El algoritmo de codificación LZ77 es reconocido como un *algoritmo de compresión universal*, [25], debido a que no requiere la distribución de la fuente de símbolos para el mensaje que está codificando. Una de las razones que explican que esto tenga sentido es que al procesar la cadena de símbolos del mensaje, el algoritmo observa la distribución de la fuente y sus correlaciones. Al realizar la observación disminuye la divergencia Kullback-Leibler respecto a la verdadera distribución de la fuente y mientras más precisa se realice la observación, más se reduce la divergencia Kullback-Leibler. Es así que el algoritmo de codificación LZ77 maximiza esta divergencia al procesar la cadena de símbolos.

4.3.2. Extensión reproducible de una cadena

La extensión reproducible de una cadena S en la posición j es la subcadena más larga que empieza en la posición $j+1$ y que ocurre antes de la posición j, en S. Por ejemplo, si tomamos $j = 10$, para la cadena:

$$abbcababcc\underline{a}bbac,$$

entonces la extensión reproducible sería abb, de longitud 3, comenzando en la posición 1, como:

$$abbcababcc\underline{abb}ac.$$

Observe que, la extensión reproducible siempre tiene longitud menor que la longitud de S menos j: $\ell(S) - j$, ya que ninguna cadena que empiece en la posición $j+1$ puede ser más larga que eso. Además, observe que la extensión reproducible puede describirse como el par: posición dónde comienza y longitud.
No obstante, note que el cómputo de la extensión reproducible es perfectamente programable: sabemos que a lo sumo es necesario procesar $\ell(S) - j$ símbolos de la cadena S, para cada una de las j primeras posiciones de la cadena, por definición: un total de $(\ell(S) - j) \times j$ símbolos tienen que ser revisados, en el peor de los casos, antes de terminar el cómputo. El algoritmo 8, en el Apéndice B, ilustra el procedimiento más directo para encontrar la extensión producible de una cadena S, dado un entero j, $j \leq \ell(S)$.

En resumen, la extensión reproducible puede codificarse como un par de números enteros que referencian la posición y el tamaño de la extensión reproducible. Lo anterior constituye una codificación de diccionario adaptativo, donde cada cadena puede expresarse como la posición y la longitud de su extensión reproducible.

4.3.3. Ventana móvil de codificación

Habiendo entendido lo qué es y cómo se codifica la extensión reproducible para cualquier cadena S y posición j, es posible entender en qué consiste la codificación LZ77: basicamente, emplea una ventana móvil de tamaño fijo, n, la cual se divide en dos partes: búfer diccionario y el búfer extensión. El búfer diccionario contiene aquella porción de la cadena, dentro de la ventana, que ya ha sido procesada por el algoritmo, mientras que el búfer extensión contiene la otra porción de la cadena aún sin codificar. Observe la figura 4.1. El algoritmo LZ77 codifica la extensión reproducible de la subcadena que empieza en la posición uno

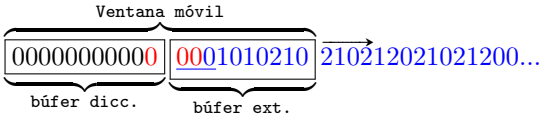

Figura 4.1: Ventana móvil de tamaño fijo y los búferes de diccionario y extensión, como se definió en [24]

del búfer extensión, respecto a la subcadena del búfer diccionario. Es decir, aplica la función de extensión reproducible sobre la cadena que está dentro de la ventana en ese momento, para j igual al tamaño del búfer diccionario. De lo anterior, se obtiene la codificación de la extensión reproducible del búfer extensión, respecto a los patrones del búfer diccionario. Luego, se desplaza la posición de la ventana extensión reproducible símbolos a la derecha y repite el mismo proceso mientras que el tamaño del búfer extensión sea mayor que cero.

El algoritmo LZ77 genera una palabra código, en cada iteración, en la forma de una terna: longitud, posición y siguiente símbolo despues de la extensión reproducible. Las dos primeras entradas de la terna codifican la extensión reproducible. Mientras que, el símbolo despues de la extensión reproducible almacena el estado de la ventana (del búfer de extensión) en el momento especifico en el que se llevó a cabo la codificación. Esto último permite al decodificador reconstruir el estado de la ventana en cada iteración de la codificación: al reconstruir la ventana en cada iteración, se recupera la cadena total. Observe la figura 4.2.

```
while (lookAheadBuffer not empty) {
  get a reference (position, length) to longest match;
  if (length > 0) {
    output (position, length, next symbol);
    shift the window length+1 positions along;
  } else {
    output (0, 0, first symbol in the lookahead buffer);
    shift the window 1 character along;
  }
}
```

Figura 4.2: Idea básica de la codificación LZ77 presentada en [24]

La principal dificultad del algoritmo radíca en gestionar apropiadamente el desplazamiento de la ventana. Para esto es necesario tener en cuenta el tamaño, n, de la ventana, así como el tamaño del búfer extensión, L_s. Pero ademas debemos conocer la posición de la ventana en cada iteración: para esto, consideramos la posición de la ventana como el límite derecho y lo denotamos win_pos. Note que, en todo momento la ventana se encuentra a la izquierda de win_pos, lo cual es conveniente ya que éste alcanzará el último caracter de la cadena antes de que lo haga la ventana, lo cual permitirá definir el criterio de parada para LZ77.

Los detalles técnicos de cómo se desplaza la ventana puede consultarlos en el algoritmo 9, Apéndice B.

4.3.4. Ejemplo de codificación LZ77

El ejemplo que se presenta a continuación es el ejemplo original que Abraham Lempel y Jacob Ziv utilizaron en su artículo [25] para presentar por primera vez el funcionamiento de su algoritmo de codificación.

Dados los siguiente valores de entrada:

$$S = 001010210210212021021200...$$
$$n = 18$$
$$L_s = 9$$

Llamamos la rutina LZ77(S, n, L_s) para los valores anteriores: en las líneas uno y dos del algoritmo 9 se definen $winpos = L_s$ y $B_i = padding(n - L_s)$;

$$B_1 = \boxed{000000000}$$
$$\downarrow$$
$$S[1 : L_s] = \boxed{001010210}\ 210212021021200...$$

En la línea tres del algoritmo 9 se lleva a cabo el comando $B_i += S[1 : L_s]$.

$$B_1 = \boxed{000000000\underline{0}01010210}\ 210212021021200...$$

En la línea seis del algoritmo 9 se lleva a cabo el comando Extension_reproducible(B_1, $n - L_s$): la extensión reproducible es 00, de longitud $\ell = 2$, empezando en la posición $p = 9$, por lo cual, en la línea quince, $winpos += 2$. A continuación, se retorna la terna $(2, 9, 1)$.

Aquí comienza una nueva iteración de LZ77(S, n, L_s):

$$B_2 = 000\boxed{000000001\underline{01}0210210}\ 212021021200...$$

Línea 6: Extension_reproducible(B_2, $n - L_s$) = $(3, 8)$,
Línea 14: $winpos += 3$,
Línea 15: Se retorna la terna $(3, 8, 2)$.
Aquí comienza una nueva iteración de LZ77(S, n, L_s):

$$B_3 = 0000000\boxed{000010102\underline{10}2\underline{10}2120}\ 21021200...$$

Línea 6: Extension_reproducible(B_3, $n - L_s$) = $(7, 7)$,
Línea 14: $winpos += max(1, \ell) = 1$,
Línea 15: Se retorna la terna $(7, 7, 2)$.
Aquí comienza una nueva iteración de LZ77(S, n, L_s):

$$B_4 = 000000000001010\boxed{2102102120\underline{2}1021200}\ ...$$

Línea 6: Extension_reproducible(B_4, $n - L_s$) = $(8, 3)$,
Línea 14: $winpos += max(1, \ell) = 1$,
Línea 15: Se retorna la terna $(8, 3, 0)$.

Cuando queremos codificar cada terna en algún alfabeto de α símbolos, el número total de α-bits necesarios para codificar cada terna es

$$L_c = \lceil log_\alpha(n - L_s) \rceil + \lceil log_\alpha(L_s) \rceil + 1,$$

$\lceil log_\alpha(n - L_s) \rceil$ para codificar la posición; $\lceil log_\alpha(L_s) \rceil$ para codificar la longitud y 1 para codificar el símbolo siguiente de la extensión reproducible. Con esta codificación todos las palabras código tienen la misma longitud y sabemos exactamente cuántas posiciones de la cadena codificada representan una palabra código: cada L_c símbolos de la cadena codificada son una palabra código. Observe el algoritmo 11, Apéndice B.

4.3.5. Ejemplo de decodificación LZ77

Continuando con el ejemplo anterior: el proceso de decodificación comienza con un búfer diccionario, de tamaño $n - L_s = 9$, con una configuración por defecto de ceros en todas sus entradas, al igual que en el ejemplo de codificación. Durante este ejemplo nos saltamos las líneas 6, 7 y 8 del algoritmo 11, Apéndice B.

Dadas la siguientes palabras código:

$$C_1 = (2, 9, 1)$$
$$C_2 = (3, 8, 2)$$
$$C_3 = (7, 7, 2)$$
$$C_4 = (8, 3, 0)$$

El decodificador recuperará la cadena reconstruyendo el contenido del búfer extensión con base en el diccionario, representado por el búfer diccionario, que se tenia en el instante i en el que se generó la palabra código C_i. La palabra código indicará la dirección en el diccionario y también los parámetros que permitirán revertir la extensión reproducible, para extender el búfer extensión. No obstante, como el diccionario es la propia cadena, conforme vayamos decodificando una palabra código se irá construyendo el diccionario para la siguiente.

En un primer instante de la decodificación, se tiene el búfer diccionario con el cual se codifico la primera palabra código, $C_1 = (2, 9, 1)$, el padding de ceros:

$$B_1 = \boxed{000000000},$$

Luego, la terna $(2, 9, 1)$ nos dice que debemos ir a la dirección 9 del búfer diccionario, donde hay un 0, y leer dos posiciones a la derecha, que sería lo mismo que leer 00, ya que lo que está en la posición 9 también está en la posición $j+1 = 10$, en este caso: lo anterior es la extensión reproducible. Ahora, a la cadena que hemos leido, agregar un 1 despues (al final), el resultado es la cadena decodificada: 001.

$$B_1 = \boxed{000000000\_},$$
(diccionario posición: $(2, \textcircled{9}, 1)$) $$B_1 = \boxed{0000000000},$$
(leemos $(\textcircled{2}, 9, 1)$) $$B_1 = \boxed{00000000000},$$
(agregamos $(2, 9, \textcircled{1})$) $$B_1 = \boxed{000000000001}$$

Por último, desplazamos la ventana el número de símbolos que hallamos extendido el búfer extensión, para obtener el nuevo búfer diccionario con el que se codificó la siguiente palabra código.
En la siguiente iteración, repetimos el mismo proceso, esta vez para $C_2 = (3, 8, 2)$, con el nuevo diccionario:

$$B_2 = 000\boxed{000000001\_}$$
(diccionario posición: $(3, \textcircled{8}, 2)$) $$B_2 = 000\boxed{000000010},$$
(leemos $(\textcircled{3}, 8, 2)$) $$B_2 = 000\boxed{000000001010},$$
(agregamos $(3, 8, \textcircled{2})$) $$B_2 = 000\boxed{0000000010102},$$

En la siguiente iteración, tenemos $C_3 = (7, 7, 2)$, con un nuevo diccionario:

$$B_3 = 0000000\boxed{000010102\_},$$
(diccionario posición: $(7, \textcircled{7}, 2)$) $$B_3 = 0000000\boxed{000010102\underline{1}},$$
(leemos $(\textcircled{7}, 7, 2)$) $$B_3 = 0000000\boxed{0000101021021021},$$
(agregamos $(7, 7, \textcircled{2})$) $$B_3 = 0000000\boxed{0000101021021021\underline{2}},$$

En la última iteración que veremos, tenemos $C_4 = (8, 3, 0)$, con un nuevo diccionario:

$$B_4 = 000000000001010\boxed{210210212\_},$$

(diccionario posición: $(8, \textcircled{3}, 0)$) $\quad B_4 = 000000000001010\boxed{2102102120},$

(leemos $(\textcircled{8}, 3, 0)$) $\quad B_4 = 000000000001010\boxed{21021021202102120},$

(agregamos $(8, 3, \textcircled{0})$) $\quad B_4 = 000000000001010\boxed{210210212021021200},$

Así que, el resultado del proceso de codificación, hasta ahora, es la cadena:

$$\cancel{000000000}0010102102102120210212 00$$

Por último, removemos el padding con el que llenamos el primer búfer diccionario, esto es: los primeros $n - L_s = 9$ ceros de la cadena. Así nos queda la cadena: $S = 001010210210212021021200$.

Para conocer los detalles más técnicos del proceso de decodificación observe el algoritmo 11, Apéndice B.

4.4. Árboles filogenéticos del lenguaje natural

Los lenguajes humanos pueden pensarse, naturalmente, como fuentes de información: más allá del alfabeto con el que se realiza la escritura de un idioma, en los textos escritos se pueden identificar ciertas regularidades (o estructuras) que caracterizan cada lenguaje, bien sea gramática o semántica, o cualquier otro mecanismo. Estas regularidades pueden modelarse, hasta cierto punto, mediante su formulación como una fuente de símbolos, tal como lo sugieren en la referencia [1]. Es entonces lógico pensar que para el estudio de un idioma también aplican los conceptos de teoría de la información, vistos en las secciones 2 y 3. Uno podría preguntarse qué lenguajes son más parecidos entre sí, y la pregunta sería equivalente a comparar dos fuentes de símbolos. Tales preguntas pueden entonces ser respondidas mediante el formalismo de la teoría de la información. Para esto, se puede aplicar la teoría de la compresión de cadenas a la aproximación de algunas de las medidas de información vistas anteriormente. Más aún, mediante esta aproximación el proceso para determinar qué lenguajes son más parecidos entre sí se puede automatizar, para construir árboles filogenéticos del lenguaje, por ejemplo.

En esta sección, se presenta una aplicación bien conocida de la compresión de cadenas al aprendizaje automático de máquina no supervisado. Los resultados presentados al final de esta sección sugieren que los métodos de aprendizaje automático explicados aquí pueden servir para construir árboles filogenéticos del lenguaje natural, un problema fundamental en lingüística y antropología, tal como lo mencionan en la referencia [6], el cual tradicionalmente es resuelto mediante métodos exhaustivos y costosos, difícilmente automatizables.

4.4.1. Métodos discriminantes (estadísticos)

Los *métodos discriminatorios* son definidos en la referencia [10] como procesos de análisis de datos exploratorios, usados comúnmente para investigar/ratificar diferencias entre objetos representados dentro de una muestra. Lo que se busca es describir gráficamente o algebráicamente las diferencias entre objetos de distintas poblaciones muestreadas, caracterizadas por distintos parámetros poblacionales.

En la referencia [10], comúnmente, se hace distinción entre las siguientes dos aproximaciones para llevar a cabo el análisis discriminatorio dentro de un conjunto de datos:

1. **Clasificación:** La *clasificación* es un proceso de discriminación no exploratorio. En la clasificación, se fija el número de grupos (o poblaciones muestreadas) y las reglas, de antemano, para clasificar nuevos objetos en cada grupo. El problema de la clasificación consiste en derivar una regla de clasificación óptima para asignar nuevos objetos a cada grupo, de modo que se minimice el costo total (promedio) de cometer un error de clasificación.

2. **Agrupamiento:** El *agrupamiento* es un proceso de discriminación exploratorio. En el agrupamiento, no se asumen suposiciones acerca de los grupos (o poblaciones muestreadas). El agrupamiento consiste en agrupar en un mismo grupo los datos más parecidos entre sí. Dependiendo de la medida de similaridad empleada, se pueden formar diferentes grupos, previamente no conocidos. El problema del agrupamiento consiste en definir una medida de similitud/diferencia para agrupar o desagrupar observaciones.

4.4.2. Clasificación

A continuación, se explica el objetivo fundamental de la clasificación y las ideas de su implementación:

Objetivo: el objetivo de la clasificación es separar datos y posicionarlos en grupos diferentes, previamente definidos.

Métodología:

1. Se usan reglas extraídas a partir de datos de aprendizaje (o de entrenamiento): se examinan observaciones, cuya población es conocida y se estudian sus diferencias.

2. Se definen regiones de clasificación como aquellas que minimizan el costo total (promedio) de clasificar erróneamente.

3. En base a las reglas extraídas o las regiones de clasificación especificadas, se clasifica tal que si una nueva observación cae dentro de una región específica, entonces se clasifica como de la clase a la cual dicha región corresponde.

La idea es: si es muy improbable que una instancia dada de los datos pertenezca a una población, dadas las características observadas de la misma, entonces ésta no debería ser clasificada como parte de esa población.

La clasificación puede llegar a ser bastante buena y tiene la ventaja de ser conceptualmente intuitiva. Además, los métodos que la implementan suelen ser fáciles de programar. Sin embargo, la clasificación depende de que se conozcan las distribuciones de cada una de las poblaciones que se clasifican, los parámetros poblacionales que caracterizan cada población, así como los costos de clasificar erróneamente y las distribuciones *a priori* de cada población (necesario para calcular el costo total -promedio- de clasificar erróneamente).

4.4.3. Agrupamiento

A continuación, se explica el objetivo fundamental del agrupamiento y las ideas de su implementación:

Objetivo: el objetivo del agrupamiento es encontrar grupos (o *clusters*) razonablemente buenos, en el sentido en que los elementos de cada grupo son lo más similares posible entre sí, sin necesidad de considerar todas las combinaciones posibles de grupos que se pueden formar con las instancias del conjunto de datos.

Métodología:

1. **Clustering jerárquico:** el *clustering jerárquico* consiste en iniciar con un único cluster/grupo (o con tantos grupos como instancias de datos) y realizar sucesivas uniones (o divisiones) entre los datos o clusters más cercanos entre sí, hasta que ya no sea posible unir (o dividir) más clusters.

2. **Clustering no jerárquico:** el *clustering no jerárquico* se diferencia del clustering jerárquico en cuanto no realiza divisiones o uniones sucesivas. El clustering no jerárquico se utiliza comúnmente para agrupar datos más que para agrupar variables. Su implementación es menos costosa computacionalmente, que la mayoría de los métodos de clustering jerárquico.

   Quizás, el método de clustering no jerárquico más conocido sea *k-means*, el cual se explica a continuación para ilustrar la idea del clustering no jerárquico:

   a) Fijar k centroides iniciales.

   b) Uno por uno, asignar cada instancia del conjunto de datos al grupo que corresponde al centroide más cercano.

   c) Recalcular el centroide como el nuevo centro de masa del grupo.

   d) Repetir el procedimiento anterior hasta que no haya cambios en los grupos.

Dadas las entradas que se tienen, la clase de datos con el que se trabajará, el tipo de método discriminante que interesará a este escrito es el clustering jerárquico, el cual se explica con más detalle a continuación.

4.4.4. Clustering jerárquico

Un clustering jerárquico es un tipo particular de clustering que no necesita que se especifique el número de clusters necesarios para poder realizar el agrupamiento: basta con asignar una medida de similaridad, en base a la cual comparar las observaciones. A continuación, se realiza el clustering jerárquico utilizando como medida de similaridad una aproximación de la entropía relativa (Divergencia Kullback-Leibler). Luego se repite el proceso para una medida de similitud diferente, la *distancia de compresión normalizada*, y se comparan.

Existen dos aproximaciones para construir un clustering jerárquico:

1. Divisiva: Este es un enfoque de arriba hacia abajo: todas las observaciones comienzan en un grupo y las divisiones se realizan de forma recursiva a medida que se desciende en la jerarquía.

2. Aglomerativa: Este es un enfoque de abajo hacia arriba: cada observación comienza en su propio grupo, y los pares de grupos se fusionan a medida que uno asciende en la jerarquía.

Para la construcción de árboles filogenéticos comunmente se emplea un tipo particular de algoritmo aglomerativo: *Unweighted pair group method with arithmetic mean*, el cual consiste en lo siguiente;

1. Construir una matríz de distancias entre todos los lenguajes.

2. Identificar las filas y columnas de la matríz de las entradas con la distancia mínima: estos constutiyen un nuevo cluster.

3. Se actualiza la matríz: se eliminan las filas y columnas que ya forman parte de algún cluster, en lugar de ellas se incluyen los clusters y las distancia a cualquier otra fila o columna será la distancia promedio respecto a cada uno de los elementos del cluster.

4. Se repiten los pasos 2 y 3 hasta que en la matríz solo existe un único cluster.

El resultado de un agrupamiento (o clustering) jerárquico es lo que comúnmente se denomina **dendograma**. En un dendograma es posible visualizar de manera resumida los momentos en los que suceden las uniones (o divisiones) de los clusters, durante todo el transcurso de la construcción.

4.4.5. Entropía relativa por compresión

El algoritmo de codificación LZ77 posibilita diferenciar las reglas (aún sin conocerlas del todo) que generan algunas secuencias, usando los patrones que puedan presentarse en las propias secuencias para comprimirlas, [1]. Esto se ejemplifica perfectamente para el caso de textos (secuencias) escritos en algún lenguaje natural en diferentes idiomas: midiendo la entropía de sus fuentes mediante la compresión de las cadenas se puede distinguir un lenguaje del otro.

La idea intuitiva de emplear la entropía de las fuentes como métrica de similaridad entre los mensajes que generan dos fuentes distintas es la siguiente: en este contexto, la *entropía relativa* se interpreta como *el número de caracteres desperdiciados para codificar una cadena generada por una fuente $\mathcal{B}$ mediante la codificación óptima para las cadenas generadas por otra fuente $\mathcal{A}$* [1]. No en vano, para aproximar la entropía se pueden emplear algoritmos de compresión como lo demostraron Lempel y Ziv en [25].

En este orden de ideas, la *entropía relativa por caracter* entre las fuentes $\mathcal{A}$ y $\mathcal{B}$, S_{AB}, puede aproximarse como:

$$S_{AB} = \frac{\Delta_{Ab} - \Delta_{Bb}}{|b|}, \tag{4.1}$$

tal cómo se definió en [1]: Siendo $\Delta_{AB} = L_{A+B} - L_A$ con L_X la longitud en bits del archivo comprimido X.

En realidad, es estos casos lo que nos interesará es estimar la *entropía relativa* entre las dos fuentes por medio de algoritmos de compresión, que es lo que se describe intuitivamente en el párrafo anterior: ya que la entropía relativa se puede utilizar como una medida de distancia entre las cadenas generadas por dos fuentes distintas $\mathcal{A}$ y $\mathcal{B}$. Es decir, qué tan bien funcionan las reglas que emplea la fuente $\mathcal{A}$ para generar mensajes propios de la fuente $\mathcal{B}$, lo que nos dirá qué tan distintos son los mensajes que genera cada una.

A continuación, se presenta un procedimiento para hacer un clustering jerárquico del lenguaje natural como el que se puede ver en la figura 4.3 En este ejemplo, podemos ver cómo se agrupan algunas lenguas romances,

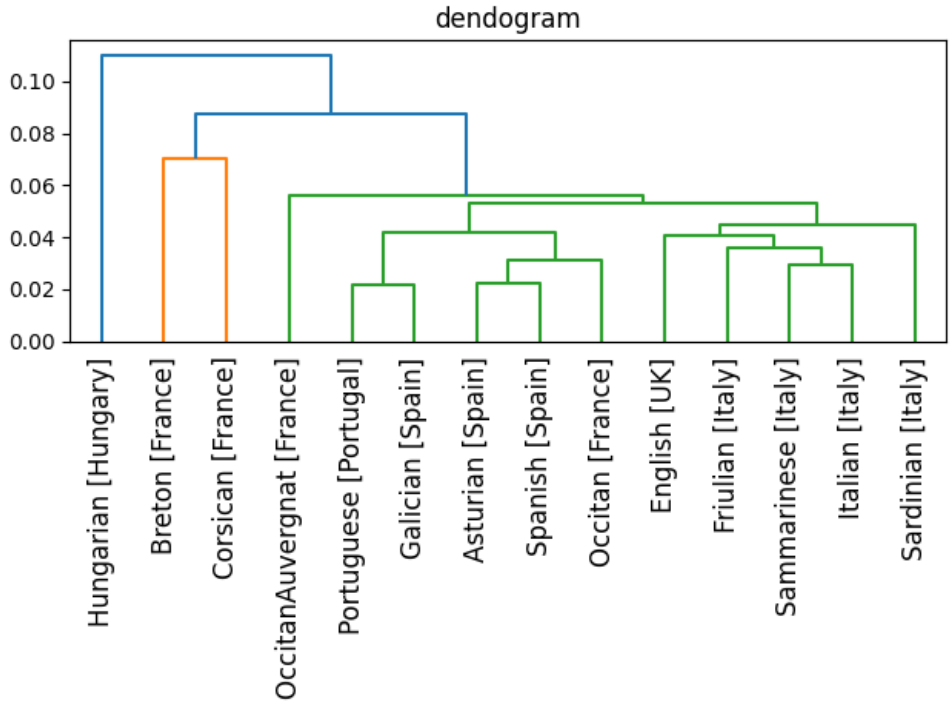

Figura 4.3: Clustering jerárquico de lenguajes romances y algunos no romances. La matríz de distancias se visualiza como un dendograma.

particularmente, cómo se agrupan en clusters diferentes a algunos lenguajes no romances como: OccitanAuvergnat[France], Breton[France], Hungarian[Hungary] o Corsian[France]. También es posible notar la cercanía que existe entre las diferentes lenguas romances.

El procedimiento descrito a continuación está basado en el que presentaron los autores *Dario Bennedetto et.al* en su artículo [1]. A diferencia de ellos, que emplearon la versión comercial del algoritmo GZIP, instancia del algoritmo *Deflate* (una combinación de LZ77 y la codificación de Huffman), aquí se emplea una versión manual del algoritmo LZ77 programada, desde cero, cien porciento en *Python* para el paso cinco a continuación.

El procedimiento es el siguiente:

1. **Construir una base de datos de la declaración de los derechos humanos escrita en más de treinta idiomas de todo el mundo:** el texto de la declaración de los derechos humanos es escogido convenientemente, ya que se ha traducido a todos los idiomas del mundo y es fácil acceder a este por medio de la página principal de la Organización de Naciones Unidas: [16]. El formato original de estos es el "portable document format (PDF): $\mathcal{A}$.pdf. Observe la figura 4.4.

Figura 4.4: Base de datos de las declaraciones de los derechos humanos en formato PDF

2. **Convertimos los archivos PDF al formato de texto .txt:** las declaraciones de los derechos humanos en formato PDF, $\mathcal{A}$.pdf, se convierten al formato TXT, $\mathcal{A}$.txt; es necesario deshacerse del encabezado y las referencias del formato PDF para lograr acceder a cada caracter del mensaje de manera individual, en el cuerpo del archivo. Observe la figura 4.5. Para esto, se empleó el módulo

Figura 4.5: Visualización simplificada de la estructura del formato PDF (en rojo, el mensaje)

slate3k de python, el cual convierte un archivo .pdf al formato .txt convertible a python array.

3. **Construimos las secuencias de símbolos extendidas:** creamos unos arreglos extendidos que equivalen a considerar la secuencia larga de toda la declaración escrita en algún idioma, $\mathcal{A}$.txt -del paso anterior-, y agregarle al final un trozo del comienzo de la declaración escrita en otro idioma, $\mathcal{B}$.txt. En la siguiente sección de código, las primeras cuatro líneas implementan el paso dos del procedimiento, y las últimas cuatro son el paso tres:

```
with open(file_pdf_A, 'rb') as f:
    file_txt_A = slate.PDF(f)
with open(file_pdf_B, 'rb') as f:
    file_txt_B = slate.PDF(f)
txt_A = str(file_txt_A)
txt_B = str(file_txt_B)
txt_Aa = txt_A + txt_A[:m]
txt_Ab = txt_A + txt_B[:m]
```

4. **Convertir todos los arreglos al formato binario:** los arreglos de caracteres se convierten al formato binario, como un arreglo de bytes (bytearray). Cada símbolo se codifica como un paquete de ocho bits (byte) de acuerdo al estándar ASCII. Se emplea el modulo bitarray de python para convertir cada caracter a su codificación binaria ASCII.

5. **Comprimir los arreglos binarios usando el algoritmo LZ77:** pasamos como argumento de entrada al algoritmo LZ77 las secuencias de bytes del paso cuatro del procedimiento. Así se obtienen las cadenas comprimidas y sus longitudes: L_X.

6. **Calcular la entropía relativa en términos del tamaño de los archivos comprimidos:** construimos la matríz de distancias, midiendo la distancia entre cada par de textos $\mathcal{A}$ y $\mathcal{B}$ como

$$S_{AB} = \frac{\Delta_{Ab} - \Delta_{Bb}}{|b|},$$

tal cómo se definió en [1]: $\Delta_{AB} = L_{A+B} - L_A$ con L_X la longitud en bits del archivo comprimido X.

7. **Construir el clustering jerárquico:** para esto se emplean las funciones `dendrogram` y `linkage` del módulo `scipy.cluster.hierarchy` de *Python* [4], el cual construye un árbol filogenético basado en el método del *Unweighted pair group method with arithmetic mean*, descrito en la sección anterior.

4.4.5.1. Resultados

En las Figura 4.6, 4.7 y 4.8, se observa algunos de los resultados que se obtienen siguiendo el procedimiento que se acaba de definir. La clasificación de los lenguajes, que se lee en la descripción de cada una de las Figuras, es la misma que la Organización de Naciones Unidas da a cada lenguaje en su base de datos [16].

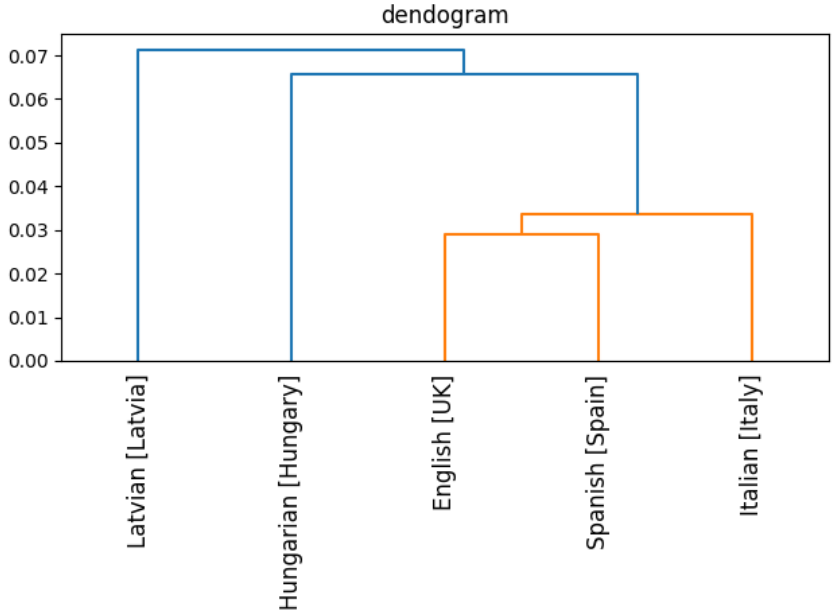

Figura 4.6: Ejemplo de clustering: 1 texto y 5 lenguajes diferentes; 2 lenguajes romances (Español e Italiano), 1 lenguaje báltica (Lituanio), Húngaro e Inglés.

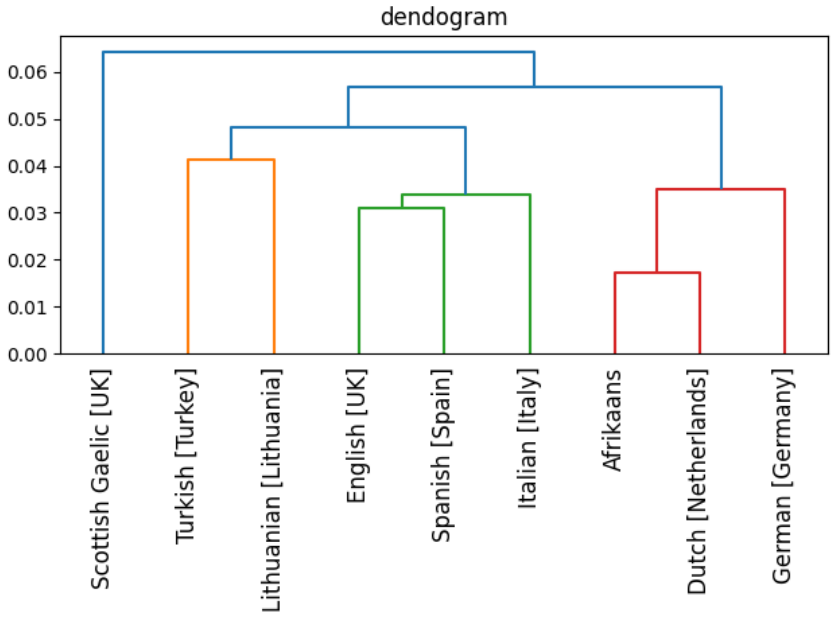

Figura 4.7: Ejemplo de clustering: 1 texto y 9 lenguajes diferentes; 3 lenguajes germánicos (Afrikaans, Holandés y Alemán), 2 lenguajes romances (Español e Italiano), 1 lenguaje altáico (Turco), 1 lenguaje báltico (Lituano), 1 lenguaje celta (Escoces Gaélico) e Inglés.

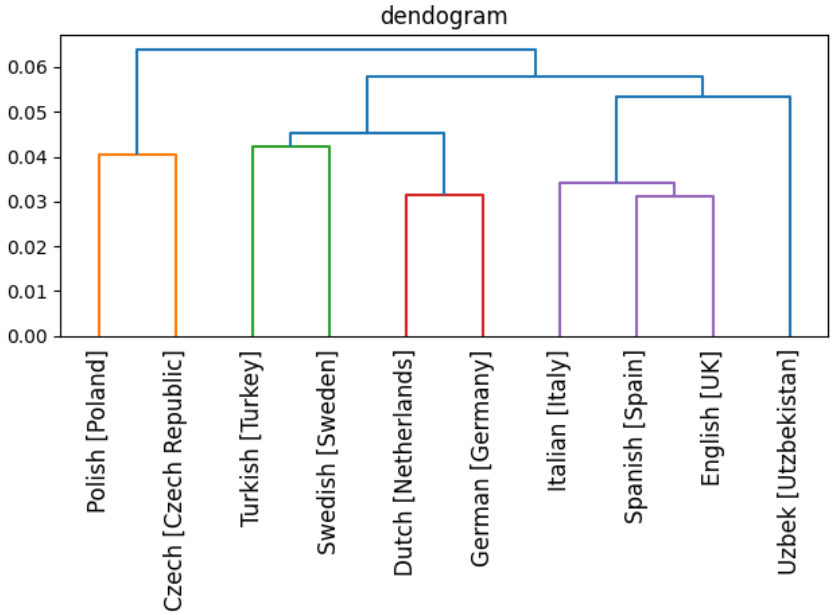

Figura 4.8: Ejemplo de clustering: 1 texto y 10 lenguajes diferentes; 3 lenguajes germánicos (Sueco, Holandés y Alemán), 2 lenguajes romances (Español e Italiano), 2 lenguajes eslávicos, 2 lenguaje altáicos (Turco y Uzbekistani), 1 lenguaje báltico (Lituano) e Inglés.

4.4.6. Distancia de compresión normalizada

La *distancia de compresión normalizada* fue definida originalmente por los autores del artículo [3]. En éste, los autores demostraron que un compresor `C` es capaz de aproximar una *distancia de información*, que ellos definen en base a la *complejidad de Kolmogorov*. Aquí, se emplea el algoritmo de codificación LZ77.

Definición 4.4.1 (*Distancia de compresión normalizada*)**.** Dado un compresor `C`, se define la *Distancia de compresión normalizada* como

$$\texttt{NCD(x,y)} = \frac{\texttt{C(x+y)-min(C(x),C(y))}}{\texttt{max(C(x),C(y))}}, \tag{4.2}$$

donde `C(x+y)` denota el tamaño de la compresión de la concatenación de `x` y `y`, `C(x)` denota el tamaño de la compresión de `x` y `C(y)` denota el tamaño de la compresión de `y`. [3]

La interpretación de la *distancia de compresión normalizada*, `NCD(x,y)`, que los autores proponen es la siguiente: Sí $\texttt{C(y)} \geq \texttt{C(x)}$, entonces la *distancia de compresión normalizada* es

$$\texttt{NCD(x,y)} = \frac{\texttt{C(x+y)} - \texttt{C(x)}}{\texttt{C(y)}},$$

de modo que la distancia `NCD(x,y)`, entre las cadenas de caracteres `x` y `y`, es la mejora, debido a la compresión de `y` usando a `x` como una base de datos previamente comprimida, respecto a comprimir a `y` desde un inicio, expresado como el rádio entre la longitud bit-a-bit de las dos versiones comprimidas.

Ahora, suponga que se lleva a cabo un procedimiento idéntico al de la sección anterior para construir un clustering jerárquico del lenguaje natural, pero a diferencia de éste, esta vez, en lugar de utilizar la aproximación de la *entropía relativa* como métrica en el paso 6, se utiliza la *distancia de compresión normalizada*:

1. Construir una base de datos de la declaración de los derechos humanos escrita en más de treinta idiomas de todo el mundo.

2. Convertir los archivos PDF al formato de texto .txt.

3. Construir las secuencias de símbolos extendidas.

4. Convertir todos los arreglos al formato binario.

5. Comprimir los arreglos binarios usando el algoritmo LZ77.

6. Calcular la entropía relativa en términos del tamaño de los archivos comprimidos: construimos la matriz de distancias, midiendo la distancia entre cada par de textos `A` y `B` como

   $$\texttt{NCD(A, B)}) = \frac{\texttt{C(A+B)-min(C(A),C(B))}}{\texttt{max(C(A),C(B))}}$$

   tal cómo se definió en [3]: donde `C(A+B)` denota el tamaño de la compresión de la concatenación de `A` y `B`, `C(A)` denota el tamaño de la compresión de `A` y `C(B)` denota el tamaño de la compresión de `B`.

7. Construir el clustering jerárquico.

4.4.6.1. Resultados

El resultado de este procedimiento puede verlo en la figura 4.9.

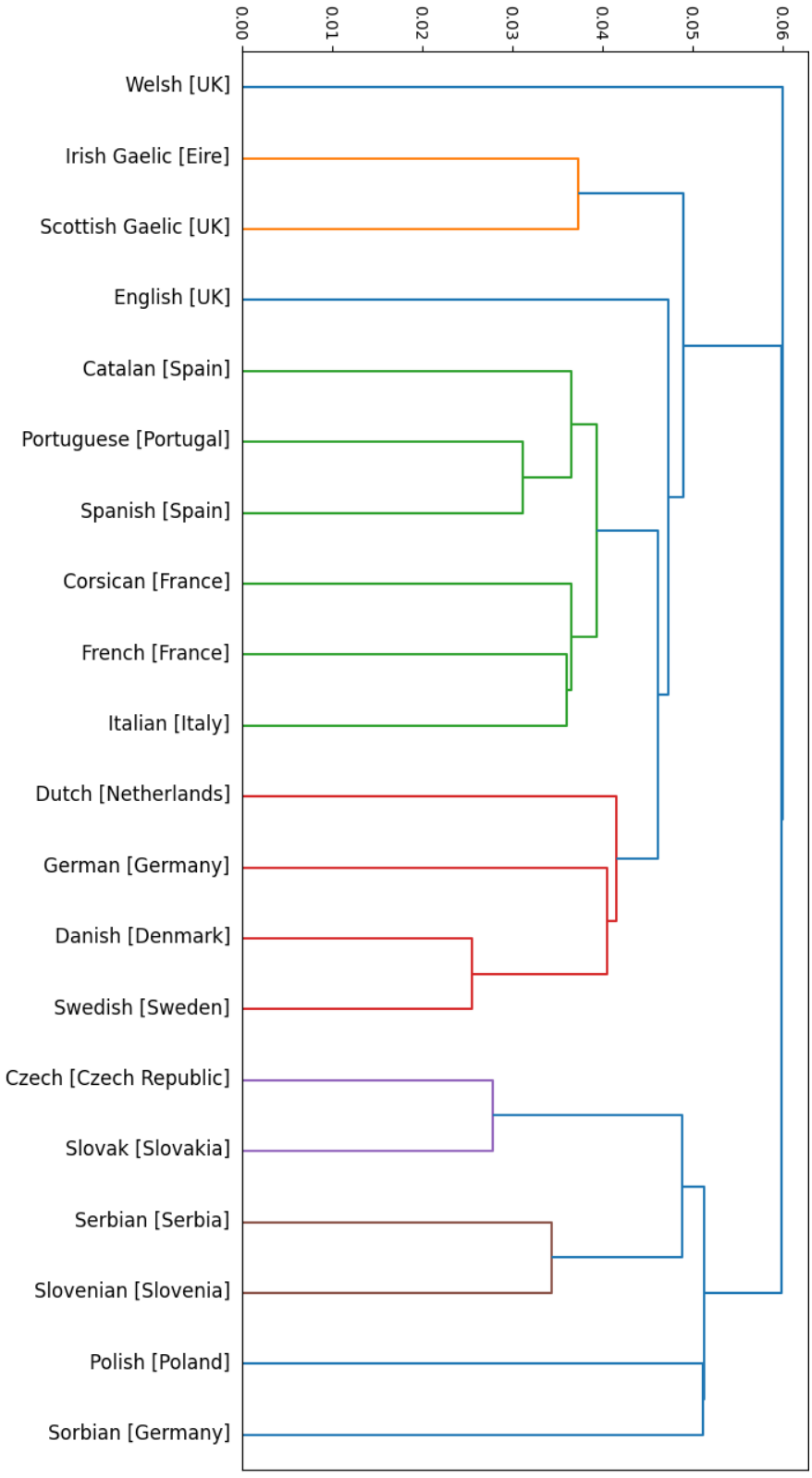

Figura 4.9: Clustering del lenguaje natural mediante la *distancia de compresión normalizada*.

Para esta última Figura, la Figura 4.9, se utilizaron colores para resaltar la cercanía entre los lenguajes: cada uno de los grupos coloreados corresponden a lenguajes de entrada que se encuentran a una cercanía menor a un umbral, que por defecto *Python.Scipy* inicializa en un porcentaje de similitud de 0,7. Es decir, todos los grupos de colores diferentes se encuentran en un mismo nivel de cercanía, los unos de los otros. Entonces, los distintos colores en el diagrama pueden interpretarse como los distintos grupos que se encuentran bajo un mismo umbral de cercanía, lo que los hace lo más parecidos entre sí. También merece la pena notar que la longitud de las ramas índica la similaridad entre las partes agrupadas en un mismo cluster: mientras más pequeñas sean las ramas, más cercanas, bajo la métrica utilizada, se encuentran los elementos agrupados. Esto es debido a que las uniones más tempranas suceden más abajo en el dendograma. Por lo tanto, mientras más pronto suceda la unión, más corta será la rama que los una.

En este orden de ideas, a partir de la Figura 4.9, se pueden resaltar los siguientes hechos:

1. Los dos lenguajes más cercanos entre sí, y más parecidos de todo el *corpus*, son Swedish [Sweden] y Danish [Denmark]. Además de ser los dos lenguajes que más se parecen entre sí, también se puede notar que son los dos lenguajes que más se parecen de todos, pues la longitud de la rama que los une es la más corta que une dos clusters en el dendograma. Por ejemplo, note que Portuguese [Portugal] y Spanish [Spain] son los dos lenguajes más cercanos entre sí, pero también note que la longitud de la rama que los une es mayor que la que une los lenguajes Swedish y Danish, por lo tanto, se concluye que Swedish y Danish son más parecidos entre sí de lo que lo son Portuguese y Spanish.

2. Los dos lenguajes más cercanos entre sí, pero menos parecidos de todo el *corpus*, son Polish [Poland] y Sorbian [Germany]. Observe que aunque estos dos lenguajes son los más parecidos entre sí, su similitud es menor, por ejemplo, que la que hay entre Serbian [Serbia] y Slovenian [Slovenia]. De hecho, su similitud es casi comparable con la que existe entre el cluster de color rojo (lenguas germánicas) y el cluster de color verde (lenguas romances).

3. El lenguaje menos parecido a todos los demás es Welsh [UK], ya que la longitud de la rama que lo une junto al cluster de todos los demás lenguajes es la más larga de todo el dendograma.

4.4.7. Conclusiones del clustering para el lenguaje natural

Los procedimientos presentados aquí permiten construir árboles filogenéticos del lenguaje para cualesquiera lenguajes de entrada. Lo anterior también podría generalizarse a otros tipos de textos como: textos escritos por diferentes autores o cadenas de ADN, tal como lo sugieren los autores del artículo [1].

Los resultados obtenidos son consistentes: los lenguajes conservan su distancia relativa a otro, incluso cuando se introducen nuevos lenguajes en el árbol, tal como se puede notar para el *idioma español* que siempre se agrupó entre los lenguajes *romances* durante las pruebas de agrupamiento para la medida de entropía relativa. Utilizar la medida de compresión normalizada como métrica del agrupamiento demostró un rendimiento ligeramente superior respecto a la entropía relativa, al comparar los resultados con aquellos obtenidos por los autores del artículo [6], para los cuales se empleó el método de *inferencia histórica*.

La implementación de estos procedimientos en *Python* permitieron obtener resultados muy similares al de los artículos [1] y [3], aplicando el mismo método para construir la visualización en forma de árbol filogenético a partir de la matriz de distancias: *Unweighted pair group method*. Los resultados no fueron idénticos, teniendo en cuenta que se utilizó un compresor diferente para estimar las distancias de *entropía relativa* y *medida de compresión normalizada*. Es posible notar la similitud con los resultados del artículo [6], que, aunque no fueron iguales, claramente se observan las mismas islas de agrupamiento: lenguas *Itálicas*, *Célticas*, *Germánicas* y *Eslávicas*, en la Figura 4.9, con gran claridad. Cabe señalar que los autores del artículo [6] obtuvieron sus resultados mediante *el método de inferencia histórica*, una aproximación distinta a la presentada aquí.

A continuación, se listan algunos detalles de la implementación que deben ser tenidos en cuenta:

1. Es recomendable utilizar textos escritos en algún alfabeto codificable por medio de la codificación *ASCII* o *UNICODE*, para facilitar la lectura desde Python.

2. Para la construcción del corpus de lenguajes es recomendable incluir solo textos que tengan aproximadamente la misma longitud, para evitar un sesgo durante el cálculo de las distancias: durante los ensayos, se comprobó que si dos textos dentro del corpus tienen longitudes muy parecidas, pero muy distintas respecto a los demás textos, entonces estos textos serán cercanos, aún cuándo sean informativamente más cercanos a otros: se observó que cuando se toman bloques de la misma longitud dentro del texto, la cercanía desaparece. Además, se pudo comprobar que para textos de longitudes similares las distancias son consistentes al variar el tamaño de los bloques.

3. Es recomendable emplear el método de *Unweighted pair group method* para representar la matríz de distancias como un árbol filogenético. Este método es un método de enlace (linkage) promedio, en el cual la cercanía se calcula con respecto al promedio de la distancia a los elementos del cluster. Otros métodos de enlace promedio podrían funcionar igual de bien. De cualquier modo, no se recomienda emplear métodos de enlace sencillo, ni métodos de enlace completo, ya que, como se vió, algunos lenguajes pueden ser muy distintos, a pesar de pertenecer a un mismo cluster.

Todos los algoritmos implementados durante el desarrollo de este escrito pueden ser consultados en el repositorio de *GitHub*: `https://github.com/estebanhernandezr/Information-theory-Clustering`, en la rama `testing`, la versión más actualizada. También puede encontrarlo en la referencia [8].

Apéndice A

Algoritmo de codificación de Hartley

A.1. Función `generate_combinations`

La función `generate_combinatios` es un algoritmo recursivo que genera todas las combinaciones posibles, de una determinada longitud, `length`, de símbolos de un alfabeto dado, `alphabet`, que tienen como prefijo una cadena, `combination`. Recibe como argumentos una lista de símbolos individuales, `alphabet`, una longitud, `length`, y una cadena de caracteres, `combination`. Retorna como resultado una lista de combinaciones.

Algorithm 1: `generate_combinations`

```
Input: list alphabet, int lenght, string combination.
Output: list combinations.
if len(combination) == lenght then
    return [combination]
else
    combinations=[]
    for s in alphabet do
        combinations += generate_combinations(alphabet, lenght, combination + s)
    return combinations
```

La función `combinations` es una *función envoltorio* (*wrapper*) de la función `generate_combinations`. Recibe como argumentos una lista de símbolos individuales, `alphabet` y una longitud, `length`. Retorna como resultado una lista de tamaño $\texttt{len(alphabet)}^{\texttt{length}}$ con todas las combinaciones posibles, de longitud `length`, de símbolos del alfabeto dado, `alphabet`. Inicializa el valor de la cadena `combination` como una cadena vacia.

Algorithm 2: `combinations`

```
Input: list B, int lenght.
Output: list combinations.
combinations = generate_combinations(B, lenght, )
return (combinations)
```

A.2. Función hartley_coding

La función `hartley_coding` genera un libro de códigos de tamaño `len(A)` $= \alpha$, con palabras código de longitud $\log_\beta(\alpha)$, para $\beta =$ `len(B)`. Para esto emplea la función `combinations(alphabet, length)` para los valores `alphabet=B` y `length=`$log_\beta(\alpha)$. Luego, la correspondencia entre cada símbolo de A y cada palabra código en `combinations(alphabet, length)` se realiza por medio de los índices de las listas. Recibe como argumento las listas de símbolos individuales, `A` y `B`. Retorna una lista de pares ordenados: `(simbolo, codigo)`, correspondiente al mapeo de la codificación.

Algorithm 3: `hatley_coding`

Input: `list A`, `list B`.
Output: `list hartley_codebook`.

```
codeword_length = log(len(A)) / log(len(B))
codeword_length = ceil(codeword_length)
codewords = combinations_book(B, codeword_length)
hartley_codebook = []
for i in 0:len(A) do
    hartley_codebook.append((A[i], codewords[i]))
return (hartley_codebook)
```

Puede acceder al código python que implementa estos pseudocódigos por medio de la referencia [8].

Apéndice B

Algoritmo de codificación de Huffman

La implementación que se presenta a continuación realiza la construcción de un árbol libre de prefijos canónico mediante las operaciones de *extensión* y *fusión*. Para implementar prácticamente las ideas de las demostraciones de la sección de Codificacion de Huffman se programan las dos operaciones que preservan la óptimalidad del código libre de prefijos, `merge` y `expand`, entre los alfabetos de m y $m - \beta$ símbolos. Además de esto, se define una clase `node` que implementa la estructura del árbol y una función `sort_nodes` que ordena los objetos de la clase `node` de acuerdo a uno de sus atributos.

B.1. Clase `node`

La clase `node` formaliza la idea de un nodo dentro de un árbol libre de prefijos: un nodo se caracteriza por el símbolo que almacena (puede almacenar uno sí es un nodo hoja o ninguno sí es un nodo interno), la probabilidad de ese símbolo, el código acumulado hasta ese momento y sus descendientes.

```
class node(self, symbol, probability, code, children):
    self.symbol = symbol
    self.probability = probability
    self.code = code
    self.children = children
```

Los tipos de datos de los atributos de la clase `node` son los siguientes: `symbol:char`, `probability:float`, `code:string` y `children:node_list`.

B.2. Función `sort_nodes`

La función `sort_nodes` implementa un algoritmo de ordenamiento (*selection sort*) sobre objetos de la clase `node`, ordenándolos de acuerdo a su atributo `probability`, de orden mayor a menor (de izquierda a derecha). Recibe como argumento una lista de objetos de la clase `node`. Retorna la lista ordenada de objetos de la clase `node`.

Algorithm 4: `sort_nodes`

Input: `list nodes`.
Output: `list nodes`.

```
1  n=len(nodes)
2  for i in 0:n do
3      k = i
4      j = i-1
5      while k >0 and nodes[k].probability >nodes[j].probability do
6          c = nodes[k]
7          nodes[k] = nodes[j]
8          nodes[j] = c
9          k -= 1
10         j = k-1
11 return nodes
```

B.3. Función `expand`

La función `expand(root, B)` es una función recursiva que implementa la codificación libre de prefijos de β símbolos en cada iteración: cada descendiente del nodo raíz actual es asignado a algúno de β símbolos diferentes y su código se extiende en ese símbolo de más. Recibe como argumentos un objeto de la clase `node`, `root`, y la lista de símbolos individuales que componen las palabras código, `B`. Retorna el objeto de la clase `node`, `root`, con el atributo `children` modificado: el atributo `code` de cada objeto de la clase `node`, en la lista `children` de `root`, es extendido un caracter del alfabeto `B`, en cada iteración.

Algorithm 5: `expand`

Input: `node root`, `list B`.
Output: `node root`.

```
1  for i, child in enumerate(root.children) do
2      child.code = root.code + B[i].symbol;
3      expand(child);
4  return (root)
```

B.4. Función `merge`

La función `merge(P, B)` es una función recursiva que reordena y fusiona los β nodos menos probables de la lista de objetos de la clase `node`, `P`, en cada iteración, hasta que solo queda un único nodo en la lista, el nodo raíz del árbol, en el caso base: en cada recursión, la función realiza un llamado a sí misma para una nueva lista de $m - \beta$ nodos, `P_`. El resultado de esta invocación es lo que la función retorna en cada paso. Al final, la función retornará el nodo raíz del árbol libre de prefijos canónico. En cada paso, la función `merge` construye el alfabeto de $m - \beta$ símbolos, `P_`, de manera canónica: los β símbolos menos probables se fusionan en un mismo símbolo. Como detalle a tener en cuenta: debido a que el número de símbolos en la lista `P` no necesariamente es un múltiplo de β, es posible que cerca al caso base no se puedan fusionar los β símbolos menos probables, debido a que el número de nodos en la lista es menor que β. Este detalle se maneja apropiadamente en los pasos 7 y 8, definiendo unos límites en los pasos 5 y 6 para el número de nodos que se fusionan. Recibe como argumento una lista de objetos de la clase `node`, `P`, y la lista de símbolos individuales que componen las palabras código, `B`. Retorna un objeto de la clase `node`, el nodo raíz del árbol libre de prefijos canónico.

Algorithm 6: merge

```
Input: list P, list B.
Output: node object.
beta = len(B);
if len(P)>1 then
    P = sort_nodes(P);
    merge_node = node(proba=sum([x.proba for x in P[-beta:]]), code=, children=[]);
    liminf = max(0, (len(P)-beta));
    limsup = min(beta, len(P));
    for i in range(0, limsup) do
        merge_node.children.append(P[liminf+i]);
    P_ = P[:-beta]+[merge_node];
    return (merge(P_));
else
    return (P[-1])
```

B.5. Función `huffman_coding`

La función `huffman_coding` implementa la codificación de Hufmman en dos pasos: primero, construye el árbol libre de prefijos canónico mediante la función `merge`, y luego, construye el código canónico recorriendo las aristas del árbol y extendiendo el código de cada nodo conforme va recorriéndolo. Recibe como argumentos una lista de objetos de la clase `node`, `P`, y la lista de símbolos individuales que componen las palabras código, `B`. Retorna, el nodo raíz del árbol libre de prefijos que implementa la codificación canónica del conjunto de nodos `P`.

Algorithm 7: huffman_coding

```
Input: list P, list B.
Output: node root.
root = merge(P, B);
root = expand(root, B);
return (root)
```

Puede acceder al código python que implementa estos pseudocódigos por medio de la referencia [8].

Apéndice C

Algoritmo de codificación LZ77

C.1. Función `reproducible_extension`

La función `reproducible_extension` implementa el concepto de extensión reproducible sobre una cadena de caracteres, que se definió en la sección Extensión reproducible. Recibe como argumentos una cadena de caracteres `S` y un entero `j`, que corresponde a la posición dentro de `S`, en la cual realizar la extensión: la mitad del buffer móvil de codificación.

Algorithm 8: Extension_reproducible

Input: String S, int j
Output: Descripción de la extensión reproducible

```
1  extens = [ ];
2  i = 1;
3  while i ≤ j do
4      ext = 0;
5      while ext ≤ len(S) − j do
6          if S[i : i + ext − 1] == S[j + 1 : j + ext] then
7              ext++;
8          else
9              break;
10     extens.append_pair(ext, i);
11     i++;
12 ext_rep = max(extens, ord = "Dictionary);
13 return ext_rep;
```

C.2. Función `LZ77_coding`

La función `LZ77_coding` implementa una ventana móvil de codificación de tamaño `n` sobre una cadena de caracteres, `S`, y los búferes internos por medio de la función `reproducible_extension`, para un tamaño del búfer extensión `Ls`. Recibe como argumentos una cadena de caracteres, `S`, el entero `n` y el entero `Ls`.

Retorna ternas de números enteros que corresponden a la descripción (`(position, length, next_bit)`) de la extensión reproducible de la cadena dentro de la ventana móvil de codificación en los distintos momentos de las iteraciones.

Algorithm 9: LZ77

Input: String S, int n, int L_s
Output: Codificación de la cadena

```
1  winpos = L_s;
2  B_i = padding(n − L_s);
3  B_i+ = S[1 : L_s];
4  while winpos − L_s < ℓ(S) do
5      if winpos ≤ ℓ(S) then
6          ℓ, p = Extension_reproducible(B_i, n − L_s);
7          ℓ = max(1, ℓ);
8          B_i = B_i[ℓ + 1 : n];
9          B_i+ = S[winpos + 1 : winpos + ℓ];
10     else
11         ℓ, p = Extension_reproducible(B_i[1, ℓ(S) − 1], n − L_s);
12         ℓ = max(1, ℓ);
13         B_i = B_i[ℓ + 1 : ℓ(S)];
14     winpos+ = ℓ;
15     output(ℓ, p, B_i[1]);
```

C.3. Función `decode_codeword`

La función `decode_codeword` recupera la extensión reproducible a partir de la descripción (`(position, length, next_bit)`), sobre una cadena de caracteres que funciona como diccionario `B`. Recibe como argumentos: una cadena de caracteres diccionario, `B`, un número entero `p`, un número entero `l` y un caracteres individual, `s`. Retorna una cadena de caracteres que corresponde a la extensión reproducible que se describío por medio de la terna.

Algorithm 10: LZ77 decoding code word

Input: String B, int p, int ℓ, char s
Output: extensión reproducible

```
1  ext_rep = “ ;
2  for i = 0 to ℓ do
3      B_i+ = B_i[p + i];
4      ext_rep+ = B[p + i];
5  ext_rep+ = s;
6  output(ext_rep);
```

C.4. Función `LZ77_decoding`

La función `LZ77_decoding` implementa una ventana móvil de decodificación en la cual realiza la extensión reproducible tomando como diccionario la cadena de caracteres que se encuentra en el primer búfer de

la ventana móvil, y se desplaza cada vez una posición fija de caracteres. Recibe como argumentos una cadena de caracteres (codificados), `C`, un número entero, tamaño de la ventana, `n`, y el tamaño del búfer extensión, `Ls`. Como pudimos ver el algoritmo 11 hace uso de la subrutina 10.

Algorithm 11: LZ77 decoding

Input: String C, int n, int L_s
Output: Cadena S decodificada

$L_c = L_c = \lceil log_\alpha(n - L_s) \rceil + \lceil log_\alpha(L_s) \rceil + 1$;
$B_i = padding(n - L_s)$;
$i = 1$;
while $i < \ell(C)$ **do**
    $p = C[i : \ i + \lceil log_\alpha(n - L_s) \rceil]$;
    $\ell = C[i + \lceil log_\alpha(n - L_s) \rceil : \ i + L_c - 1]$;
    $s = C[i + L_c - 1 : \ i + L_c]$;
    $p, \ell, s = Radix\_to\_Decimal(p, \ell, s)$;
    $ext\_rep = decode\_word(B_i, p, \ell, s)$;
    $B_i += ext\_rep$;
    $B_i += B_i[\ell :]$;
    $i += L_c$;
$output(S)$;

Puede acceder al código python que implementa estos pseudocódigos por medio de la referencia [8].

Bibliografía

[1] Dario Benedetto, Emanuele Caglioti y Vittorio Loreto. «Language Trees and Zipping». En: *Physical Review Letters* 88.4 (2002). DOI: 10.1103/PhysRevLett.88.048702.

[2] G. J. Chaitin. «A theory of program size formally identical to information theory». En: *Journal of the ACM* 22.3 (1975), págs. 329-340. DOI: 10.1145/321892.321894.

[3] Rudi Cilibrasi y Paul M.B Vitányi. «Clustering by compression». En: *IEEE TRANSACTIONS OF INFORMATION THEORY* VOL 51, NO.4 (2005).

[4] Python community. *scipy.cluster.hierarchy*. 2022. URL: https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html.

[5] Thomas. M. Cover y Joy. A. Thomas. *Elements of information theory*. Wiley-Interscience. A John Wiley & Sons, INC, publication, 2006.

[6] Russell. D Gray, Simon. J Greenhill y Quentin. D Atkinson. «Lenguaje evolution and human history: what a difference a data makes». En: *PHYLOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIERY* VOL 51, NO.4 (2011). DOI: 10.1098/rstb.2010.0378.

[7] Ralph. V. L. Hartley. «Transmission of Information». En: *bell system technical journal* (1928). URL: http://keszei.chem.elte.hu/entropia/Hartley1928text.pdf.

[8] Esteban Hernández Ramírez. *Information-theory-Clustering*. Nov. de 2022. URL: https://github.com/estebanhernandezr/Information-theory-Clustering.

[9] PHILIPP VON HILGERS y AMY N. LANGVILLE. «THE FIVE GREATEST APPLICATIONS OF MARKOV CHAINS». En: *Information and Control 7* (2010). URL: https://langvillea.people.cofc.edu/MCapps7.pdf.

[10] Richard A. Johnson y Dean W. Wichern. *Applied multivariate statistical analysis*. Pearson, Prentice hall, 2007.

[11] K.Lindgren. *Information theory for complex systems*. Complex systems group. Department of Energy y Environment, 2014. ISBN: 9780198520115.

[12] A. N. Kolmogorov. «Three approaches to the quantitative definition of information». En: *Problems in Information Transmission* 1.1 (1965), págs. 1-7.

[13] Stanislav Krajci et al. «Performance Analysis of Fano coding». En: *Signal and Information Processing Lab* (2012).

[14] Ming Li y Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Texts in computer science. Springer, 2019. ISBN: 978-3-030-11297-4.

[15] Toshiko Matsumoto. «Biological Sequence Compression Algorithms». En: *Genome Informatics* 11 (2000), págs. 43-52. DOI: 10.11234/GI1990.11.43.

[16] Organización de Naciones Unidas. *Base de datos de la Declaración Universal de los Derechos Humanos*. 2022. URL: https://www.ohchr.org/es/universal-declaration-of-human-rights.

[17] C. E. Shannon. «A mathematical theory of communication». En: *The Bell System Technical Journal* 27.3 (1948), págs. 379-423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[18] R. J. Solomonoff. «A formal theory of inductive inference». En: *Information and Control 7* Parts 1 and 2.1-22 (1964), págs. 224-254. DOI: https://doi.org/10.1016/S0019-9958(64)90223-2.

[19] Madhu Sudan y Xiang David. «A Self-Contained Analysis of the Lempel-Ziv Compression Algorithm». En: *Harvard John A. Paulson School of Engineering and Applied Sciences* (2019).

[20] Andreia Teixeira et al. «Entropy Measures vs. Kolmogorov Complexity». En: *Entropy* 13.3 (2011), págs. 595-611. ISSN: 1099-4300. DOI: 10.3390/e13030595. URL: https://www.mdpi.com/1099-4300/13/3/595.

[21] Benedetta Tondi y Mauro Barni. *Lectures notes on Information Theory and Coding*. Universita degli Studi di Siena Facolta di Ingegneria, 2012.

[22] Paul M.B. Vitányi. «How Incomputable Is Kolmogorov Complexity?» En: *Entropy* 22.4 (2020). ISSN: 1099-4300. DOI: 10.3390/e22040408. URL: https://www.mdpi.com/1099-4300/22/4/408.

[23] Mark M. Wilde. *Quantum Information Theory*. Cambridge University press, 2013. ISBN: 978-1-107-03425-9.

[24] Christina Zeeh. *The Lempel Ziv Algorithm*. Seminar "Famous Algorithms". 2003.

[25] Jacob Ziv y Abraham Lempel. «A Universal Algorithm for Sequential Data Compression». En: *IEEE TRANSACTIONS ON INFORMATION THEORY* 23.3 (1977), págs. 337-343. DOI: 10.1109/TIT.1977.1055714.

[26] Peter Zörnig. *Non-linear programming*. De Gruyter Textbook. 2014. DOI: https://doi.org/10.1515/9783110315288.