# PRÁCTICA 1: CÓDIGO HUFFMAN

**Autor** *Martín Fernández de Diego*
**Compañera de código** *Belén Sánchez Centeno*

## 1. INTRODUCCIÓN

A partir de una pequeña muestra de cada población de las variables aleatorias $S_{Esp}$ y $S_{Eng}$, se pretenden resolver las tres siguientes cuestiones:

***Apartado i)*** **Hallar el código Huffman binario de $S_{Eng}$ y $S_{Esp}$, sus longitudes medias $L(S_{Eng})$ y $L(S_{Esp})$, y comprueba que se satisface el Primer Teorema de Shannon.**

***Apartado ii)*** **Codificar con dicho código la palabra cognada X = "medieval" para ambas lenguas, y comprobar la eficiencia de longitud comparada con el código binario usual.**

***Apartado iii)*** **Decodifica la siguiente palabra del inglés `10111101101110110111011111`.**

## 2. MATERIAL USADO

Cada muestra de población se proporciona en un fichero que incluye un texto breve de los dos idiomas. Se analiza este texto para construir un modelo que responda a la probabilidad de que cierto caracter aparezca.

### 2.1. Apartado i)

Se han utilizado las funciones:

- `huffman_tree(dataframe)`
- `huffman_branch(dataframe)`
- `longitud_media(dataframe,diccionario)`
- `entropia(dataframe)`

Para hallar el **código Huffman**, cada muestra es dispuesta en dos diccionarios que asocian los caracteres a su número de apariciones en el lenguaje. A continuación, se redisponen en un *dataframe* que asocia los caracteres a su probabilidad de aparición. Se construye el árbol de Huffman, haciendo uso de las funciones dadas `huffman_tree(dataframe)` y `huffman_branch(dataframe)`, y así se minimiza la longitud media del binario que identifica a cada elemento. Este código resulta de recorrer de raíz a hojas el árbol de Huffman para cada elemento del lenguaje.

La **longitud media** de la codificación binaria de los caracteres del lenguaje se calcula como $L(C) = \frac{1}{W}\sum_{i=1}^{N} w_i|c_i|$ donde $|c_i|$ es la longitud de cada cadena binaria y $W = \sum_{i=1}^{N} w_i = 1$ por ser $w_i$ frecuencias relativas. La función `longitud_media(dataframe,diccionario)` recibe las frecuencias por el parámetro `dataframe` y los códigos asociados a cada elemento por el `diccionario`. Simplemente multiplicamos la probabilidad de cada elemento por la longitud de su código.

Comprobamos finalmente que se satisface el **Primer Teorema de Shannon**, $H(C) \leq L(C) < H(C) + 1$, a través de las funciones $H$ de `entropia(dataframe)` y $L$ de `longitud_media(dataframe,diccionario)`.

### 2.2. Apartado ii)

Se ha utilizado la función:

- `codifica(palabra,diccionario)`
- `codifica_usual(palabra,dataframe)`

Para **codificar** una palabra haciendo uso de la función `codifica(palabra,diccionario)`, basta tomar uno a uno los caracteres de la palabra y buscar su traducción binaria en el `diccionario` de entrada. El resultado será la concatenación de estas cadenas binarias.

Su **eficiencia** vendrá determinada por *eficiencia* $= \frac{L(C)}{L(C_{usual})}$ donde $L(C)$ corresponde a la longitud media de la codificación construida vía Huffman y $L(C_{usual})$ a la longitud media del código binario que resulta de la codificación natural y que se calcula como $L(C_{usual}) = \lceil \log_2(\textit{cardinal del lenguaje}) \rceil$ en `codifica_usual(palabra,dataframe)`.

### 2.3. Apartado iii)

Se ha utilizado la función:

- `decodifica(binario,diccionario)`

Para **decodificar** una cadena binaria haciendo uso de la función `decodifica(binario,diccionario)`, basta tomar la primera subcadena que represente un caracter del lenguaje usado y, una vez traducido, repetir la operación hasta llegar al final de la cadena. Para ello, se invertirá el `diccionario` de entrada de modo que la *clave* sea el código binario y el *valor* sea el símbolo del lenguaje.

**Ejemplo**: Dado el código `010`, el algoritmo toma el primer dígito binario `0`. Si no coincide con ninguna clave del diccionario, repite la operación, toma un segundo dígito —en este caso el `1`— y lo concatena con los previos que no han conformado aún un código recogido en el diccionario —`01`—.

## 3. RESULTADOS

### 3.1. Apartado i)

Primero, se calculan los diccionarios de los códigos Huffman binarios de $S_{Eng}$ y $S_{Esp}$. Aunque se mostrará la ejecución con el diccionario completo en el Anexo, véase aquí que el símbolo ' ' en $S_{Eng}$ se ha codificado con `00` y el mismo símbolo en $S_{Esp}$ se ha codificado con `111`.

Las longitudes medias han resultado,

$$L(S_{Eng}) = 4{,}158163265306123$$

$$L(S_{Esp}) = 4{,}431924882629108$$

y además, como las entropías son

$$H(S_{Eng}) = 4{,}117499394903037$$

$$H(S_{Esp}) = 4{,}3943938614799665$$

se verifica el Primer Teorema de Shannon

$$S_{Eng} : 4{,}117 \leq 4{,}158 \leq 5{,}117$$

$$S_{Esp} : 4{,}394 \leq 4{,}432 \leq 5{,}394$$

### 3.2. Apartado ii)

Codificación de *medieval* en $S_{Eng}$:

```
1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1
1 1 1 0 0 0 1 1 1 0 1 1 1 1 0 1 0 0 1 1
0 1 0 1 1 0 1 1 1 0
```

Codificación de *medieval* en $S_{Esp}$:

```
1 1 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 1 0
1 0 0 1 1 1 0 1 0 1 0 0 1 1 0 1 0 1
```

Conocida la longitud de la palabra para la codificación usual —56—, para la codificación de $S_{Eng}$ —50— y de $S_{Esp}$ —38— se pueden calcular las eficiencias.

$$\textit{eficiencia}(S_{Eng}(\textit{medieval})) = 112\,\%$$

$$\textit{eficiencia}(S_{Esp}(\textit{medieval})) = 147,368\,\%$$

### 3.3. Apartado iii)

La palabra codificada con $S_{Eng}$ en:

```
1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1
0 1 1 1 1 1
```

es *hello*.

## 4. CONCLUSIÓN

Véase que no es necesario utilizar símbolos especiales de separación entre los dígitos de los códigos binarios para indicar el fin de una letra. El algoritmo implementado traduce automáticamente en cuanto la subcadena correspondiente está asociada con un símbolo. Algo similar ocurre con otras lenguas de nuestro mundo. El japonés no incluye espacios entre palabras, sino entre oraciones.

Luego, cabe destacar la ventaja de eficiencia del español, respecto del inglés, como lenguaje utilizado para codificación binaria. Por supuesto, esta ventaja es consecuencia directa del algoritmo de Huffman.

También es necesario aclarar que lo que en este reporte se entiende por codificación usual se basa en la cantidad mínima de dígitos necesarios para identificar de manera unívoca los símbolos de cierto lenguaje. En la codificación usual no se toma la longitud de los códigos como una variable con carácter semántico.

# 5. ANEXO CON EL SCRIPT Y CÓDIGO UTILIZADO

## 5.1. Código

```
"""
PRÁCTICA 2: CÓDIGO HUFFMAN
Belén Sánchez Centeno
Martín Fernández de Diego
"""

import os
import numpy as np
import math
import pandas as pd
from collections import Counter

"""
Dado un dataframe
devuelve una rama del arbol de Huffman
"""
def huffman_branch(distr):
    states = np.array(distr['states'])
    probab = np.array(distr['probab'])
    state_new = np.array(['' .join(states[[0,1]])])
    probab_new = np.array([np.sum(probab[[0,1]])])
    codigo = np.array([{states[0]: 0, states[1]: 1}])
    states =  np.concatenate((states[np.arange(2,len(states))], state_new), axis=0)
    probab =  np.concatenate((probab[np.arange(2,len(probab))], probab_new), axis=0)
    distr = pd.DataFrame({'states': states, 'probab': probab, })
    distr = distr.sort_values(by='probab', ascending=True)
    distr.index=np.arange(0,len(states))
    branch = {'distr':distr, 'codigo':codigo}
    return(branch)

"""
Dado un dataframe
devuelve el arbol de Huffman
"""
def huffman_tree(distr):
    tree = np.array([])
    while len(distr) > 1:
        branch = huffman_branch(distr)
        distr = branch['distr']
        code = np.array([branch['codigo']])
        tree = np.concatenate((tree, code), axis=None)
    return(tree)

"""
Dado un arbol de Huffman
devuelve un diccionario con el codigo de cada estado
"""
def extraer_cadena_caracter(tree):
    d = dict() # diccionario {carácter : código}
    # Se recorre el árbol desde la raíz, que se encuentra en la última posición
    for i in range(tree.size-1,-1,-1):
        # Se accede a ambas hojas
        for j in range(2):
            estado = list(tree[i].items())[j][0]
            # Se sabe por construcción del árbol que en la primera hoja hay un 0 y en la
     segunda un 1
            codigo = str(j)
            # Se guardan o actualizan los caracteres en el diccionario
            for caracter in estado:
                if caracter in d:
                    d[caracter] += codigo # codigo (0 o 1)
                else:
                    d[caracter] = codigo # codigo (0 o 1)
    return d

"""
```

```
Dado un dataframe y un diccionario de un código de Huffman
devuelve la longitud media, es decir, la suma de las longitudes de los elementos por sus probabilidades
"""
def longitud_media(distr,d):
    lm = 0
    for i in range(len(d)):
        lm += len(d[distr.at[i,'states']])*distr.at[i,'probab']
    return lm

"""
Dado un dataframe
devuelve la entropía total del sistema
"""
def entropia(distr):
    h = 0
    for p in distr['probab']:
        h -= p*math.log(p,2)
    return h

"""
Dada una palabra y un diccionario de un código de Huffman
devuelve su codificación en binario
"""
def codifica(palabra, d):
    binario = ""
    for c in palabra:
        binario += d[c]
    return binario

"""
Dada una palabra en binario y un diccionario de un código de Huffman
devuelve su decodificación
"""
def decodifica(binario, d):
    palabra = ""
    codigo = ''
    # Se separan keys y values en diferentes listas para poder buscar key por value
    list_values = list(d.values())
    list_keys = list(d.keys())
    for bit in binario:
        # Se buscan los tramos mínimos del binario que constituyen un codigo asociado a un caracter
        codigo += bit
        if codigo in d.values():
            palabra += list_keys[list_values.index(codigo)]
            codigo = ''
    return palabra

"""
Dada una palabra y un dataframe con los caracteres disponibles
devuelve la longitud de la codificacion binaria usual de esa palabra
"""
def codifica_usual(palabra, distr):
    count = math.log(len(distr),2)
    return len(palabra)*math.ceil(count)


# FORMATO
class Formato:
    BOLD = "\033[1m"
    RESET = "\033[0m"

#### Vamos al directorio de trabajo ####
os.getcwd()
#os.chdir(ubica)
#files = os.listdir(ruta)

with open('/Users/martin/Documents/Estudios/Matematicas e Ingenieria Informatica/2021-2022/GCom/Git/GCom/Laboratorio/P2/GCOM2022_pract2_auxiliar_eng.txt', 'r',
```

```
        encoding="utf8") as file:
133       en = file.read()
134
135 with open('/Users/martin/Documents/Estudios/Matematicas e Ingenieria Informatica
        /2021-2022/GCom/Git/GCom/Laboratorio/P2/GCOM2022_pract2_auxiliar_esp.txt', 'r',
        encoding="utf8") as file:
136       es = file.read()
137
138 # APARTADO i)
139 print("\n" + Formato.BOLD + "Apartado i)" + Formato.RESET)
140
141 # eng
142 print("\n" + Formato.BOLD + "S_eng:" + Formato.RESET)
143 tab_en = Counter(en)
144
145 ##### Transformamos en formato array de los carácteres (states) y su frecuencia
146 ##### Finalmente realizamos un DataFrame con Pandas y ordenamos con 'sort'
147 tab_en_states = np.array(list(tab_en))
148 tab_en_weights = np.array(list(tab_en.values()))
149 tab_en_probab = tab_en_weights/float(np.sum(tab_en_weights))
150 distr_en = pd.DataFrame({'states': tab_en_states, 'probab': tab_en_probab})
151 distr_en = distr_en.sort_values(by='probab', ascending=True)
152 distr_en.index = np.arange(0,len(tab_en_states))
153
154 tree_en = huffman_tree(distr_en)
155
156 # Código de Huffman binario
157 d_en = extraer_cadena_caracter(tree_en)
158 print("Código de Huffman binario: " + str(d_en))
159 # Longitud media
160 lm_en = longitud_media(distr_en, d_en)
161 print("Longitud media: " + str(lm_en))
162 # Entropía
163 e_en = entropia(distr_en)
164 # Teorema de Shannon
165 print("Teorema de Shannon: " + str(e_en) + " <= " + str(lm_en) + " < " + str(e_en+1))
166
167 #esp
168 print("\n" + Formato.BOLD + "S_esp:" + Formato.RESET)
169 tab_es = Counter(es)
170
171 ##### Transformamos en formato array de los carácteres (states) y su frecuencia
172 ##### Finalmente realizamos un DataFrame con Pandas y ordenamos con 'sort'
173 tab_es_states = np.array(list(tab_es))
174 tab_es_weights = np.array(list(tab_es.values()))
175 tab_es_probab = tab_es_weights/float(np.sum(tab_es_weights))
176 distr_es = pd.DataFrame({'states': tab_es_states, 'probab': tab_es_probab })
177 distr_es = distr_es.sort_values(by='probab', ascending=True)
178 distr_es.index=np.arange(0,len(tab_es_states))
179
180 tree_es = huffman_tree(distr_es)
181
182 # Código de Huffman binario
183 d_es = extraer_cadena_caracter(tree_es)
184 print("Código de Huffman binario: " + str(d_es))
185 # Longitud media
186 lm_es = longitud_media(distr_es, d_es)
187 print("Longitud media: " + str(lm_es))
188 # Entropía
189 e_es = entropia(distr_es)
190 # Teorema de Shannon
191 print("Teorema de Shannon: " + str(e_es) + " <= " + str(lm_es) + " < " + str(e_es+1))
192
193 # APARTADO ii)
194 print("\n" + Formato.BOLD + "Apartado ii)" + Formato.RESET)
195
196 palabra = 'medieval'
197 codifica_huffman_en = codifica(palabra, d_en)
198 codifica_huffman_es = codifica(palabra, d_es)
199
```

```
200 print("Codificacion de \"" + palabra + "\" en inglés: " + codifica_huffman_en)
201 print("   Eficiencia del " + str(codifica_usual(palabra,distr_en)/len(
        codifica_huffman_en)*100) + "%")
202 print("Codificacion de \"" + palabra + "\" en español: " + codifica_huffman_es)
203 print("   Eficiencia del " + str(codifica_usual(palabra,distr_es)/len(
        codifica_huffman_es)*100) + "%")
204
205 # APARTADO iii)
206 print("\n" + Formato.BOLD + "Apartado iii)" + Formato.RESET)
207
208 binario = '10111101101110110111011111'
209 print("La palabra '" + binario + "' decodificada del inglés es '" + decodifica(binario,
        d_en) + '\'')
```