PRÁCTICA 2: Código Huffman

Geometría Computacional

Belén Sánchez Centeno
belsan05@ucm.es

29 de marzo de 2022

Date Performed: 9 y 16 de febrero de 2022
Code Partner: Martín Fernández de Diego

1. Introducción

Sean $S_{Eng} = \{a, b, c, ..., Z, !, ?, -, /n, , 0, .., 9\}$ y $S_{Esp} = \{a, á, b, c, ..., ñ, ..., Z, ¡, !, ¿, ?, -, /n, , 0, .., 9\}$, dos variables aleatorias, hemos tomado una pequeña muestra de cada población y las disponemos en los archivos "GCOM2022_pract2_auxiliar_eng.txt" y "GCOM2022_pract2_auxiliar_esp.txt", respectivamente. Si suponemos que las mayúsculas y minúsculas son distintos estados de las variables, e incluso con o sin tilde, se pide:

Primer Objetivo
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.

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

Tercer Objetivo
Decodificar la palabra "10111101101110110111011111" del inglés.

2. Material utilizado

Se resuelve el problema con un script de Python, en el que se usan funciones de implementación propia, de la plantilla proporcionada y de las siguientes bibliotecas: `os`, `numpy`, `math` y `pandas`. Quedan detalladas a continuación:

- `huffman_tree`, que, dado un Dataframe con una lista de estados ordenados según sus frecuencias, devuelve el árbol de Huffman, en el que cada rama de la izquierda tiene un bit a 0 y la de la derecha a 1, y en cuyas hojas están todos los estados de forma individual. Para construir cada rama, es decir, para realizar cada iteración del *Paso 2 del algoritmo de Huffman visto en los apuntes de la asignatura, se usa la función* ***`huffman_branch`***.

- ***`extraer_cadena_caracter`****, que, dado un árbol de Huffman, devuelve un diccionario cuyas claves son los estados y los valores las cadenas de bits correspondientes para esa codificación. Para ello se recorre el árbol de la raíz a las hojas. En cada nodo, se recorre la cadena de caracteres de la rama de la izquierda y la de la derecha, añadiendo al código de cada estado un 0 o un 1 en función de la rama por la que se pasó para encontrarlo.*

- ***`longitud_media`****, que, dado un Dataframe como el de anteriores funciones y un diccionario con su codificación de Huffman, devuelve la longitud media, es decir, la suma de las longitudes de los códigos de los estados por sus frecuencias de aparición.*

- ***`entropía`****, que, dado un Dataframe con una lista de N estados independientes con sus probabilidades $\{Pj\}_{j=1}^{N}$ en la muestra, devuelve la entropía total del sistema*

$$H = -\sum_{j=1}^{N} P_j log_2 P_j \tag{1}$$

- ***`codifica`****, que devuelve la codificación en binario de la palabra dada, en el código de Huffman del diccionario dado, concatenando los códigos de cada estado.*

- ***`decodifica`****, que devuelve la decodificación de una palabra en binario dada, según el código de Huffman del diccionario dado, buscan los tramos mínimos del binario que constituyen un código asociado a un caracter y concatenando dichos códigos. Puesto que un diccionario no permite buscar una clave a partir de un valor, se separan claves y valores en dos listas diferentes y, cuando se encuentra un código que corresponde a un estado, se accede a él sabiendo la posición en la que estaba su valor.*

- ***`codifica_usual`****, que, dados una palabra y un Dataframe como el de anteriores funciones, calcula la longitud de la codificacion binaria usual por estado para la muestra de la población dada y, multiplicándola por la longitud de la palabra sin codificar, devuelve su longitud si estuviera en el binario usual.*

Adicionalmente, se han utilizado las funciones `getcwd`, de `os`, `open`, `read` para extraer todos los caracteres de las muestras proporcionadas "GCOM2022_pract2_auxiliar_eng.txt" y "GCOM2022_pract2_auxiliar_esp.txt"; y la herramienta `Counter`, de `collections`, para contar cuántas veces se repite cada uno.

3. Resultados

3.1. Primer objetivo

En primer lugar se hace este apartado para la lengua inglesa. Como se indica en el *Paso 1* del algoritmo de Huffman, se crea una lista ordenada por frecuencia de aparición de los estados de la muestra S_{Eng}, que se guarda en un Dataframe que se utillizará durante toda la práctica. A continuación, se obtiene la codificación binaria de Huffman, su longitud media y se comprueba que se verifica el Primer Teorema de Shannon, es decir, que la longitud media queda acotada por la entropía del sistema y la entropía del sistema más 1. Se realizan los mismos pasos para la lengua española. Se han obtenido los siguientes resultados:

S_eng:
Código de Huffman binario: {' ': '00', 'I': '01000', "'": '0100100', '/n': '0100101', '-': '0100110', 'A': '010011100', 'k': '010011101', 'y': '01001111', 's': '0101', 't': '011', 'c': '10000', 'g': '10001', 'f': '10010000', 'q': '100100010', ';': '100100011', 'b': '10010010', '?': '10010011', ',': '1001010', 'S': '1001011', 'w': '10011', 'h': '101', 'i': '11000', 'u': '11001', 'a': '11010', 'r': '1101100', 'W': '11011010', 'B': '11011011', 'l': '1101110', 'd': '1101111', 'e': '1110', 'n': '111100', 'v': '11110100', 'T': '1111010100', 'p': '1111010101', 'C': '1111010110', 'm': '1111010111', '.': '1111011', 'o': '11111'}
Longitud media: 4.158163265306123
Teorema de Shannon: 4.117499394903037 $\leq$ 4.158163265306123 $<$ 5.117499394903037

S_esp:
Código de Huffman binario: {'t': '0000', 'd': '00010', 'é': '000110', 'T': '000111000', 'D': '000111001', 'C': '000111010', 'B': '000111011', 'Q': '00011110', 'w': '00011111', 'a': '001', 'e': '010', 'o': '0110', 'h': '011100', 'P': '01110100', 'v': '01110101', 'A': '011101100', 'á': '011101101', 'Y': '01110111', 'S': '0111100', 'E': '01111010', '?': '01111011', 'c': '011111', 's': '1000', 'n': '1001', 'p': '10100', 'l': '10101', 'z': '1011000', 'q': '1011001', '-': '10110100', '/n': '10110101', 'g': '10110110', 'í': '10110111', 'u': '10111', 'b': '110000', 'm': '110001', 'i': '11001', 'r': '11010', ',': '1101100', '¿': '11011010', 'ñ': '110110110', 'y': '1101101110', ';': '1101101111', '.': '1101110', 'j': '1101111', ' ': '111'}
Longitud media: 4.431924882629108
Teorema de Shannon: 4.3943938614799665 $\leq$ 4.431924882629108 $<$ 5.3943938614799665

3.2. Segundo objetivo

Codificacion de "medieval"en inglés: 11110101111111011011111100011101111010011010110111 0
Eficiencia del 96.0 %
Codificacion de "medieval"en español: 1100010100001011001010011101010011 0101
Eficiencia del 126.3157894736842 %

3.3. Tercer objetivo

La palabra '10111101101110110111011111' decodificada del inglés es 'hello'

4. Conclusión

Se ha comprobado que efectivamente se verifica el Primer Teorema de Shannon en ambos casos. La longitud media de ambas codificaciones es menor que la de la usual, sin embargo, la codificación en inglés de la palabra "medieval"es menos eficiente.

5. Anexo: Código utilizado

```
"""
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('GCOM2022_pract2_auxiliar_eng.txt', 'r',encoding="utf8") as
    file:
      en = file.read()

with open('GCOM2022_pract2_auxiliar_esp.txt', 'r',encoding="utf8") as
    file:
      es = file.read()

# APARTADO i)
print("\n" + Formato.BOLD + "Apartado i)" + Formato.RESET)

# eng
print("\n" + Formato.BOLD + "S_eng:" + Formato.RESET)
tab_en = Counter(en)

##### Transformamos en formato array de los carácteres (states) y su
    frecuencia
##### Finalmente realizamos un DataFrame con Pandas y ordenamos con '
    sort'
tab_en_states = np.array(list(tab_en))
tab_en_weights = np.array(list(tab_en.values()))
tab_en_probab = tab_en_weights/float(np.sum(tab_en_weights))
distr_en = pd.DataFrame({'states': tab_en_states, 'probab':
    tab_en_probab})
distr_en = distr_en.sort_values(by='probab', ascending=True)
distr_en.index = np.arange(0,len(tab_en_states))

tree_en = huffman_tree(distr_en)

# Código de Huffman binario
d_en = extraer_cadena_caracter(tree_en)
print("Código de Huffman binario: " + str(d_en))
# Longitud media
lm_en = longitud_media(distr_en, d_en)
```

```
print("Longitud media: " + str(lm_en))
# Entropía
e_en = entropia(distr_en)
# Teorema de Shannon
print("Teorema de Shannon: " + str(e_en) + " <= " + str(lm_en) + " < "
    + str(e_en+1))

#esp
print("\n" + Formato.BOLD + "S_esp:" + Formato.RESET)
tab_es = Counter(es)

##### Transformamos en formato array de los carácteres (states) y su
    frecuencia
##### Finalmente realizamos un DataFrame con Pandas y ordenamos con '
    sort'
tab_es_states = np.array(list(tab_es))
tab_es_weights = np.array(list(tab_es.values()))
tab_es_probab = tab_es_weights/float(np.sum(tab_es_weights))
distr_es = pd.DataFrame({'states': tab_es_states, 'probab':
    tab_es_probab })
distr_es = distr_es.sort_values(by='probab', ascending=True)
distr_es.index=np.arange(0,len(tab_es_states))

tree_es = huffman_tree(distr_es)

# Código de Huffman binario
d_es = extraer_cadena_caracter(tree_es)
print("Código de Huffman binario: " + str(d_es))
# Longitud media
lm_es = longitud_media(distr_es, d_es)
print("Longitud media: " + str(lm_es))
# Entropía
e_es = entropia(distr_es)
# Teorema de Shannon
print("Teorema de Shannon: " + str(e_es) + " <= " + str(lm_es) + " < "
    + str(e_es+1))

# APARTADO ii)
print("\n" + Formato.BOLD + "Apartado ii)" + Formato.RESET)

palabra = 'medieval'
codifica_huffman_en = codifica(palabra, d_en)
codifica_huffman_es = codifica(palabra, d_es)

print("Codificacion de \"" + palabra + "\" en inglés: " +
    codifica_huffman_en)
print("   Eficiencia del " + str(codifica_usual(palabra,distr_en)/len(
    codifica_huffman_en)*100) + "%")
print("Codificacion de \"" + palabra + "\" en español: " +
    codifica_huffman_es)
print("   Eficiencia del " + str(codifica_usual(palabra,distr_es)/len(
    codifica_huffman_es)*100) + "%")

# APARTADO iii)
print("\n" + Formato.BOLD + "Apartado iii)" + Formato.RESET)

binario = '10111101101110110111011111'
```

```
print("La palabra '" + binario + "' decodificada del inglés es '" +
    decodifica(binario,d_en) + '\'')
```