Práctica 1: Código de Huffmann
y
Primer Teorema de Shannon

Geometría Computacional

Óscar Mesa Martín

1 Introducción

Los objetivos de esta primera práctica serán dos:

I. Poder codificar y descodificar un texto (archivo *.txt*) a través de la **Codificación de Huffman**.

II. Aplicar el Primer Teorema de Shannon al texto mencionado.

Para ello implementaremos el Algoritmo de Huffman visto en las *clases teóricas* de tal manera que, a partir del texto empleado, consigamos codificarlo. Y, asimismo, que a partir de un código, seamos capaces de descodificarlo para saber de qué texto se trata. Todo ello será programado y ejecutado a través de *python*.

2 Método

a) **Árbol de Huffman**:

Comenzamos la *codificación de Huffman* por la creación del **Árbol de Huffman** del texto, es decir, la secuencia que va a asociar a cada *caracter* el *código* que le corresponda. Lo haremos apoyándonos en la librería *pandas* ya que, mediante la creación de una tabla (*DataFrame*), podremos manejar con más facilidad tanto los *caracteres* (**'states'**) como sus *pesos/frecuencias* (**'probab'**).

Mediante la función ***huffman_branch*** conseguimos realizar el primer paso del Árbol:

Fusionar los dos caracteres (*'states'*) que menos peso (*'probab'*) tengan, quedándonos así con uno menos.

Y, a través de la función ***huffman_tree***, obtendremos:

El árbol de Huffman desglosado en un array. Para ello iteraremos el proceso anterior hasta que sólo queden dos estados (que serán la raíz del árbol)

Caracter	**Frecuencia Relativa**
C	0.001200
“	0.001200
V	0.001200
H	0.001200
)	0.001200
(	0.001200
\n	0.001200
j	0.001200
...	...

b) **Código de Huffman**:

Una vez obtenido el *Árbol de Huffman*, necesitaremos saber cuál es el *código* asociado a cada *caracter*. La idea que seguiremos será la de recorrer el árbol de izquierda a derecha una vez por cada caracter, preocupándonos de ir *“pegando”* los 0's ó 1's en los que esté presente nuestro caracter (con cuidado de *“pegar por la izquierda”*, ya que lo recorreremos desde la última hoja del árbol hasta la raíz). De esto se encargará nuestra función ***leyenda***, que:

Recibe: Árbol de Huffman en formato array.

Devuelve: DataFrame donde asocia el código de Huffman a cada caracter.

Caracter	**Código**
\n	1110100110
	110
(	1001111111
)	1001111110
,	0010010
.	0111111
A	01111101
B	00100110
...	...

c) **Codificación del texto**:

Teniendo ya el *código de Huffman*, la tarea de codificar se nos simplifica. Tan sólo tendremos que recorrer de izquierda a derecha el texto que nos den e ir escribiendo, por cada caracter, el código que le corresponda. Nuestra función ***codificador*** será la responsable:

Recibe: Texto (string)

Devuelve: Cadena de 0's y 1's (el texto codificado)

Input	codificador("gcom")
Output	100110100100110001110

'g'	100110	**'c'**	10010
'o'	0110	**'m'**	01110

d) **Descodificación del código**:

La idea que recoge la función ***descodificador*** es la de ir recorriendo un código dígito a dígito identificando cada subcadena con su respectivo caracter:

Recibe: Cadena de 0's y 1's (Cadena de dígitos)

Devuelve: Texto (string que resulta de descodificar la cadena de dígitos)

Input	descodificador("10011010010011001110")
Output	gcom

'100110'	g	**'10010'**	c
'0110'	o	**'01110'**	m

3 Resultados

i) Para resolver el primer apartado, aplicamos la función ***leyenda*** a S_{Eng} y a S_{Esp}. Esto nos dará el ***Código de Huffman*** y la ***Longitud Media*** de cada texto. También sabremos que se cumple el ***Primer Teorema de Shannon***, ya que:

$$H(C) \leq L(C) < H(C) + 1 \Longrightarrow \begin{cases} 4.41218 \leq 4.43697 < 5.41218 & (\text{para } S_{Eng}) \\ 4.3383 \leq 4.3713 < 5.3383 & (\text{para } S_{Esp}) \end{cases}$$

Observación: El error cometido es $\epsilon = \pm\ 0.000005$

ii) Ahora codificaremos la palabra ***"dimension"*** en inglés y en español usando la función ***codificador***:

Input	codificador("dimension") (para S_{Eng})
Output	00111010101110000100001000101011010000

Input	codificador("dimension") (para S_{Esp})
Output	11110001101100000100110100011100010001

Para resaltar la **eficiencia** del Código de Huffman *vs* el Código Binario usual, compararemos la longitud media de sus cadenas:

$$\begin{aligned} \bar{l}_{CH} = L(C) = 4.43697 \leq 6 = [\log_2 N_{Eng}]^{\in N} = \bar{l}_{CB} \qquad (\text{para } S_{Eng}) \\ \bar{l}_{CH} = L(C) = 4.3713 \leq 6 = [\log_2 N_{Esp}]^{\in N} = \bar{l}_{CB} \qquad (\text{para } S_{Esp}) \end{aligned}$$

iii) Para concluir los apartados, descodificaremos una palabra del inglés haciendo uso de la función ***descodificador*** y así encontrarnos con:

Input	descodificador("0101010001100111001101111000101111110101010001110") (para S_{Eng})
Output	***"isomorphism"***

4 Conclusión

Después de realizar esta práctica nos damos cuenta de la ***eficiencia*** de la Codificación de Huffman, pues su valor reside en la capacidad que tiene este algoritmo de minimizar la cantidad de dígitos en la que se puede codificar cualquier texto.
Además, al habernos apoyado en el ***módulo pandas*** para facilitarnos el trabajo de lectura del código, hemos conseguido adquirir cierto manejo con los *DataFrames* de esta librería. Esto nos proporciona una herramienta muy útil para futuras prácticas en las que trabajemos con cantidades de elementos *suficientemente grandes*.

5 Anexo (código *.py*)

Código del archivo ***Práctica1_MesaMartínÓscar.py***:

```
# -*- coding: utf-8 -*-

##############################################################################
########################## IMPORTACIÓN DE LIBRERÍAS ##########################
##############################################################################

import os
import numpy as np
import pandas as pd
import math as mt

# Obtenemos el directorio actual de trabajo:

os.getcwd()



##############################################################################
############################## ELEGIR EL ARCHIVO #############################
##############################################################################


elegirTexto = input('¿Qué texto quiere elegir?:\n   a) Eng   b) Esp   \n ')

# Para el texto en Inglés (SEng):

if elegirTexto == ('a' or 'A'):
    # Abrimos el archivo en Inglés,  'r' modo lectura, y lo nombramos "pru":
    with open('GCOM2023_pract1_auxiliar_eng.txt',
              'r',
              encoding="utf8") as file:

        pru = file.read()
        idi = "Inglés"

# Para el texto en Español (SEsp):

if elegirTexto == ('b' or 'B'):
    # Abrimos el archivo en Español, 'r' modo lectura, y lo nombramos "pru":
    with open('GCOM2023_pract1_auxiliar_esp.txt',
              'r',
              encoding="utf8") as file:

        pru = file.read()
        idi = "Español"



##############################################################################
########################## TABLA INICIAL (DATAFRAME) #########################
##############################################################################


# Importamos de una librería la función Counter, que nos va a
# permitir contar, de cada caracter, cuántos hay en la prueba.
# Los va a contar por orden de aparición pero no van a estar
```

```
# ordenados por frecuencia de aparición.

from collections import Counter

tab_pru = Counter(pru)

# Distinguimos entre:
#   1) states:    Cada uno de los caracteres.
#   2) weights:   Frecuencia con que aparece cada caracter
#   3) probab:    Veces que aparece entre número de caracteres

tab_pru_states  = np.array(list(tab_pru))
tab_pru_weights = np.array(list(tab_pru.values()))
tab_pru_probab  = tab_pru_weights/float(np.sum(tab_pru_weights))

# Creamos una tabla con 'states' y 'probab' ordenada según
# había ordenado la función Counter() (por aparición):

distr_pru = pd.DataFrame({'states': tab_pru_states,
                          'probab': tab_pru_probab})

# Los ordenamos según 'probab' (frecuencia de aparición) pero,
# entonces los índices se nos descolocan:

distr_pru = distr_pru.sort_values(by = 'probab', ascending = True)

# Ordeno los índices:

distr_pru.index = np.arange(0, len(tab_pru_states))

# Renombro:

distr = distr_pru



##############################################################################
#################### PASO INICIAL DEL ÁRBOL DE HUFFMAN ######################
##############################################################################


def huffman_branch(distr):

    # Separamos "caracteres" y "frecuencias relativas":
    states = np.array(distr['states'])
    probab = np.array(distr['probab'])

    # Fusionamos los 2 estados que correspondan:
    #   1) Juntamos los 2 primeros 'states' con menos 'probab':
    state_new = np.array([''.join(states[[0,1]])])
    #   2) Su 'probab' va a ser la suma de los 2:
    probab_new = np.array([np.sum(probab[[0,1]])])

    # Creamos nuestro primera parte del Código de Huffman
    # con esos dos 'states':
    codigo = np.array([{states[0]: 0, states[1]: 1}])

    # Actualizamos:
    #   1) Juntamos el nuevo 'state' con el resto de 'states':
    states = np.concatenate((states[np.arange(2,len(states))],
```

```
                                    state_new),
                                   axis=0)
    #   2) Así como su nueva 'probab':
    probab = np.concatenate((probab[np.arange(2,len(probab))],
                                    probab_new),
                                   axis=0)

    # Creaciónde la tabla:
    #   1) La creamos:
    distr = pd.DataFrame({'states': states, 'probab': probab})
    #   2) Los ordenamos crecientemente en función de su 'probab':
    distr = distr.sort_values(by = 'probab', ascending = True)
    #   3) Ordenamos los nuevos índices:
    distr.index = np.arange(0,len(states))

    # Nuestra tabla y código (Rama del Árbol de Huffman):
    branch = {'distr': distr, 'codigo': codigo}
    return (branch)



##############################################################################
############################### ÁRBOL 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)



##############################################################################
############################## CÓDIGO DE HUFFMAN ##############################
##############################################################################


# Inicializamos variables y renombramos (por comodidad):
    # Lista de Caracteres:
caracteres = list(distr_pru['states'])
    # Ordenamos los Caracteres:
caracteres.sort()
    # Renombramos al Árbol de Huffman
t1 = huffman_tree(distr)


# IDEA:
#     Recorreremos el Árbol de Huffman de abajo a arriba,
#     una vez por cada caracter.
#     En estas "vueltas" veremos dónde está presente nuestro
#     caracter, e iremos anotando (por la izquierda) si le
#     corresponde un 0 ó un 1 en la rama en la que se encuentre.
```

```
def leyenda(tree):

    # Inicializamos variables y renombramos (por comodidad):
        # Lista de Caracteres:
    caracteres = list(distr_pru['states'])
        # Ordenamos los Caracteres:
    caracteres.sort()
        # Renombramos al Árbol de Huffman
    t1 = huffman_tree(distr)

    # Tabla Inicial de Caracteres con su respectivo Código:
    codiF = pd.DataFrame({'caracteres': caracteres, 'codigo': ''})

    for k in range(0, len(caracteres)):
        for i in range(0, len(t1)):
            # Si el caracter 'k' está en el NODO y es 0:
            if (caracteres[k] in list(t1[i].items())[0][0]):
                # Añadir a la 'C' un 0:
                codiF['codigo'][k] = '0' + codiF['codigo'][k]
            # Si el caracter 'k' está en el NODO y es 1:
            if (caracteres[k] in list(t1[i].items())[1][0]):
                # Añadir a la 'C' un 0:
                codiF['codigo'][k] = '1' + codiF['codigo'][k]
            # Si el caracter 'k' no está en el NODO:
                # Seguimos iterando.
    return (codiF)
```

```
##############################################################################
########################## CODIFICACIÓN DE HUFFMAN ###########################
##############################################################################
```

```
def codificador(texto):
    ley = leyenda(t1) # Sacamos la leyenda a partir del texto dado
    codificacion = ""
    # Recorre todos los caracteres del texto dado:
    for i in range (0, len(texto)):
        # Busca en la Leyenda cuál es el código del caracter i:
        for j in range(0, len(caracteres)):
            # Encuentra el caracter en la Leyenda y lo escribe:
            if texto[i] == ley['caracteres'][j]:
                codificacion = codificacion + ley['codigo'][j]

    return (codificacion)
```

```
##############################################################################
######################## DESCODIFICACIÓN DE HUFFMAN ##########################
##############################################################################
```

```
def descodificador(binario):
    # Cadena (de caracteres) de dígitos:
    cb = list(binario)
    # Leyenda:
    ley = leyenda(pru)
    # Para obviar cadenas vacías:
```

```
    if (len(cb) > 0):
        # Descifrado:
        descifre = ''
        # "Cachito" de CB (Código Binario):
        cacho = ''
        # Para que recorra todo el CB:
        for i in range (0, len(cb)):
            # Actualizamos el cacho código:
            cacho = cacho + cb[i]
            # Recorremos toda la Leyenda (pa ver si coindide):
            for j in range (0, len(ley)):
                # Si coincide el cacho con una letra:
                if (cacho == ley['codigo'][j]):
                    # Lo añadimos a nuestro descifrado:
                    descifre = descifre + ley['caracteres'][j]
                    # Y comenzamos con cacho nuevo:
                    cacho = ''
                # Si no coincide:
                    # Seguimos para bingo.

    return (descifre)



################################################################################
############################## FUNCIÓN PRINCIPAL ###############################
################################################################################


def main():

    # Apartado i)

    print ("\nPrimer apartado i): \n\nCódigo de Huffman binario del texto "
           + idi + ": \n")
    print (leyenda(pru), "\n")

    print ("\nRedondearemos al quinto decimal " +
           "con la función:\n\n"+
           "    round (número, 5)" +
           "\n\nPor tanto, nuestro error será:\n\n" +
           "    e = +- 0.00001")

    print ("\nLongitud media del texto " + idi + ":\n")
    # Lista Longitud Media del código de cada caracter:
    #           (ci):
    lmc     = list(map(len, leyenda(t1)['codigo']))
    # Lista de las 'probab' de cada caracter:
    #        (Hay que ordenadarlos para que pueda operar con ellos
    #         de una manera lógica cuando estén en sus listas)
    #           (wi):
    lp0     = distr.sort_values(by='states',ascending=True)
    lpc     = list(lp0['probab'])

    nrlongmed = sum (np.multiply(lmc, lpc))
    longmed = round(nrlongmed, 5)
    print ("    L(C) =", longmed)

    print ("\n¿Se Satisface el primer Teorema de Shannon?")
    print ("\nPrimer Teorema de Shannon:\n\n" +
```

```
        "     H(C)    <=    L(C)    <    H(C) + 1 \n\n" +
        "H(C): Entropía del Código " + idi)

    #           - sum (       Pj        *       [log(base2)Pj]       )
    badentropi = - sum ( distr['probab'] * np.log2(distr['probab']) )
    entropi    = round(badentropi, 5)

    print ("\n    ", entropi)

    print ("\nSe cumple el Primer Teorema de Shannon, ya que:\n\n",
           "   ", entropi, " <= ", longmed, " < ", (entropi + 1), "\n")


    input ("\n\n\n" +
           "Presione 'Enter' para continuar con el 2o apartado" +
           "\n\n\n")


    # Apartado ii)

    print ("\nSegundo apartado ii):" +
           "\n\nCodificar la palabra 'dimension':\n")
    print ("     Para el texto en " + idi + " sería:\n")
    print ("        ", codificador("dimension"), "\n")

    print ("La eficiencia del Código de Huffman" +
           " vs " +
           "el Código Binario Usual:\n\n")

    print ("CB => Longitud de las cadenas = ",
           mt.ceil(np.log2(len(distr))))
    print ("CH => Longitud de las cadenas = L(C) = ",
           longmed, "\n")
    print ("Es notoria.\n")


    input ("\n\n\n" +
           "Presione 'Enter' para continuar con el 3er apartado" +
           "\n\n\n")


    # Apartado iii)

    print ("\nTercer apartado iii):")
    print ("\nDescodificar una palabra del texto inglés:")
    print ("\n    ¡Asegúrese de haber escogido el texto inglés!\n")
    print ("Su palabra es:    ", descodificador
           ("0101010001100111001101111000101111110101010001110"))



##############################################################################
########################## EJECUCIÓN DE LA PRÁCTICA ##########################
##############################################################################

main()
```