Práctica 2 - Geometría Computacional

Marcos Herrero Agustín

1. Introducción

Esta práctica consiste calcular el código de Huffman del inglés y del español (basándonos en dos muestras dadas) y utilizarlos para codificar y decodificar algunos términos. Asimismo, se comparará la eficiencia de codificar las palabras utilizando estos códigos frente a si se utiliza el código binario usual.

2. Datos y condiciones iniciales

Para la realización de la práctica se han utilizado los siguientes datos y condiciones iniciales:

- Una muestra del inglés almacenada en el fichero GCOM2022_pract2_auxiliar_eng.txt
- Una muestra del español almacenada en el fichero GCOM2022_pract2_auxiliar_esp.txt
- Para el apartado *ii*), la palabra *medieval* a codificar
- Para el apartado *iii*), el código 10111101101110110111011111 a decodificar. También se han añadido las palabras *dragon* (para S_{Eng}) y *queso* (para S_{Esp}) para tener más ejemplos de codificación y decodificación.

3. Metodología

Para cada uno de los lenguajes, S_{Eng} y S_{Esp}, se ha utilizado el algoritmo visto en la teoría para calcular el código de Huffman del lenguaje. Los pasos seguidos son:

1) Calcular la frecuencia de aparición de cada carácter en la muestra.
2) Ordenar los caracteres de menor a mayor frecuencia.
3) Mientras quede más de un carácter, fusionar los dos de menor frecuencia, actualizar el conjunto de nodos y reordenarlo. Se obtiene una lista con los nodos fusionados en el orden en que lo han hecho, donde las posiciones pares corresponden al dígito binario '0′ y las impares al '1′.
4) Recorriendo la lista anterior, se obtiene un diccionario que traduce cada carácter del lenguaje a su código de Huffman.

Este diccionario permite traducir palabras del lenguaje a su codificación de Huffman. La relación es biyectiva, por lo que puede invertirse fácilmente para realizar traducciones inversas.

4. Resultados y discusión

4.1. Apartado *i*)

4.1.1. S_{Eng}

El código de Huffman asociado a S_{Eng} es:

' ' 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				

Su longitud media es $L(S_{Eng}) = 4.1582$
Su entropía es $H(S_{Eng}) = 4.1175$
Observamos que, como establece el Primer Teorema de Shannon, se verifica:

$$H(S_{Eng}) \leq L(S_{Eng}) < H(S_{Eng}) + 1$$

$$4.1175 \leq 4.1582 < 5.1175$$

4.1.2. S_{Esp}

El código de Huffman asociado a S_{Esp} es:

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				

Su longitud media es $L(S_{Esp}) = 4.4319$
Su entropía es $H(S_{Esp}) = 4.3944$
Observamos que, como establece el Primer Teorema de Shannon, se verifica:

$$H(S_{Esp}) \leq L(S_{Esp}) < H(S_{Esp}) + 1$$

$$4.3944 \leq 4.4319 < 5.3944$$

4.2. Apartado *ii*)

La codificación de la palabra *medieval* en binario con código UTF-8 es:

$$1101101110010111001001101001110010111101101100001110110 0$$

que tiene longitud 64. Se utilizan 8 dígitos binarios por carácter. No obstante, si restringimos el número total de caracteres que se pueden codificar a únicamente los 38 de S_{Eng} o los 45 de S_{Esp}, podríamos construir una codificación binaria usual que utilice 6 dígitos binarios por carácter, dado que:

$$\lceil \log_2 38 \rceil = \lceil \log_2 45 \rceil = 6$$

La codificación de *medieval* usando el código de Huffman de S_{Eng} es:

$$11110101111110110111111000111011110100110101101110$$

que tiene longitud 50. Se utilizan, pues, 6.25 dígitos binarios por cáracter (frente a la longitud media de S_{Eng}, que es 4.1582). La eficiencia respecto al código binario usual (de 6 dígitos binarios por carácter) es del 96 %. Por tanto, si bien de media el código de Huffman es mucho más eficiente que el binario usual para codificar, en este caso es algo peor.

La codificación de *medieval* usando el código de Huffman de S_{Esp} es:

$$11000101000010110010100111010100110101$$

que tiene longitud 38. Se utilizan, pues, 4.75 dígitos binarios por cáracter (frente a la longitud media de S_{Esp}, que es 4.4319). La eficiencia respecto al código binario usual (de 6 dígitos binarios por carácter) es del 126 % Por tanto, en este caso, aunque la longitud de la codificación de Huffman sea algo mayor que la media, sigue siendo más eficiente que la binaria usual.

4.3. Apartado *iii*)

La decodificación de la palabra 10111101101110110111011111 con el lenguaje S_{Eng} da como resultado la palabra *hello*. También se ha comprobado que se puede codificar una palabra y recuperarla decodificando, en cualquiera de los idiomas. Para este propósito, se han escogido las palabras *dragon* para S_{Eng} y *queso* para S_{Esp}.

5. Conclusiones

Hemos comprobado la sencillez de uso del código de Huffman. Asimismo, hemos observado que, de media, las palabras codificadas usando codificación de Huffman quedan considerablemente más cortas que con binario usual. Sin embargo, para palabras que utilicen caracteres no muy frecuentes en la muestra, como *medieval* en S_{Eng}, puede darse que la longitud de su código de Huffman sea mayor que la de su código binario usual.

Apéndice A Código utilizado

```
"""
Práctica 2

Autor : Marcos Herrero
"""

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


#Fusiona los dos estados con menor frecuencia para obtener una nueva rama
def huffmanBranch(distrib):
    #Extraemos los estados con menor frecuencia, que vamos a fusionar
    toMerge = distrib.head(2)
    distrib.drop(index = [0,1], inplace=True)

    #Anotamos el codigo de los estados fusionados
    codigo = np.array([toMerge['states'][0], toMerge['states'][1]])

    #Fusionamos los estados, calculando la probabilidad del nuevo
    newState = ''.join(np.array(toMerge['states']))
    newProb = np.sum(toMerge['probab'])

    #Añadimos el nuevo estado a la distribucion
    df = pd.DataFrame({'states' : [newState], 'probab' : [newProb]})
    distrib = pd.concat([distrib, df])

    #Reordenamos de menor a mayor frecuencia
    distrib = distrib.sort_values(by='probab', ascending=True)
    distrib.index=np.arange(0,len(distrib.index))

    return distrib, codigo

#Devuelve el codigo de Huffman del lenguaje y su longitud media
def huffmanCode (filename):

    with open(filename, 'r',encoding="utf8") as file:
        text = file.read()

    # Calculamos las frecuencias de aparición de cada carácter en el texto
    counter = Counter(text)
    states = np.array(list(counter))
    weights = np.array(list(counter.values()))
    probs = weights/float(np.sum(weights))
    distrib = pd.DataFrame({'states': states, 'probab': probs})

    # Calculamos, a partir de las frecuencias, la entropía del lenguaje
    H= 0
    for frec in probs:
        H += (-frec*math.log2(frec))

    #Ordenamos los caracteres de menor a mayor frecuencia
    distrib = distrib.sort_values(by='probab',ascending=True)
    distrib.index = np.arange(0,len(states))

    #Calculamos el orden en que los nodos se van fusionando, junto
    #con los dígitos que va adquiriendo cada uno
    #(que es una forma de representar el árbol de Huffman)

    ordenFusion = np.array([])
```

```
    while len(distrib) > 1:
        distrib, codigo = huffmanBranch(distrib)
        #Los elementos de las posiciones pares adquieren un 0 en esa rama,
        #y los de las posiciones impares un 1
        ordenFusion = np.concatenate((ordenFusion, codigo), axis=None)


    #Calculamos el código de Huffman del lenguaje

    codigoHuffman = {}


    for caract in states:
        codigoHuffman[caract] = []

    for i in range(len(ordenFusion)//2):

        for caract in ordenFusion[2*i]:

            codigoHuffman[caract].append('0')

        for caract in ordenFusion[2*i+1]:

            codigoHuffman[caract].append('1')

    #Obsérvese que los códigos se construyen al reves, asi que
    # hay que darles la vuelta. Aprovechamos para escribirlos como string

    for caract in states:
        codigoHuffman[caract] = ''.join(reversed(codigoHuffman[caract]))

    #Calculamos la longitud media de los codigos de Huffman
    #(ponderada por frecuencia)
    L = 0
    for i in range(len(states)):
        L += (len(codigoHuffman[states[i]])*probs[i])


    return codigoHuffman,L,H

# Devuelve la codificación de la palabra X en binario usual UTF-8
def codificarBinarioUsual (X):
    CodXBin = bytes(X,'utf8')
    CodXBin = [str(bin(x))[0]+str(bin(x))[2:] for x in CodXBin]
    CodXBin = ''.join(CodXBin)

    return CodXBin

# Devuelve la codificación de la palabra X con el código que se pasa
# como parámetro
def codificar (X,code):
    CodX = []

    for caract in X:
        CodX.append(code[caract])

    CodX = ''.join(CodX)

    return CodX

# Devuelve la codificación de la palabra CodX con el código inverso que se pasa
# como parámetro
def decodificar (CodX,revcode):
```

```
    X = []
    i , j = 0, 1

    while i < len(CodX):
        while j <= len(CodX) and CodX[i:j] not in revcode:
            j+=1

        if j > len(CodX):
            print("La palabra no forma parte del lenguaje")
            break

        X.append(revcode[CodX[i:j]])

        i = j
        j = i+1

    X = ''.join(X)

    return X

'''
Apartado i) : hallar el código de Huffman de Seng y Sesp y sus longitudes medias
y comprobar que se cumple el teorema de Shannon
'''

print('Apartado i)')
print()
#Obtenemos el códigos de huffman de las muestras, la longitud media
#y la entropía del inglés


huffCodeEn, LEn, HEn = huffmanCode('GCOM2022_pract2_auxiliar_eng.txt')

print("Lenguaje Seng:")
print("==============")
print("Codigo Huffman:")

for key,value in sorted(huffCodeEn.items(),key=lambda x:x[1]):
    print(" {} : {}".format(key,value))
print()

print("Longitud media: {}".format(LEn))
print("Entropía : {}".format(HEn))
print()
print()

#Obtenemos el códigos de huffman de las muestras, la longitud media
#y la entropía del español
huffCodeEsp, LEsp, HEsp = huffmanCode('GCOM2022_pract2_auxiliar_esp.txt')

print("Lenguaje Sesp:")
print("==============")
print("Codigo Huffman:")

for key,value in sorted(huffCodeEsp.items(),key=lambda x:x[1]):
    print(" {} : {}".format(key,value))
print()

print("Longitud media: {}".format(LEsp))
print("Entropía : {}".format(HEsp))
print()
print()
```

```
'''
Apartado ii) : codificar la palabra medieval y comparar la longitud del
código obtenido con la longitud que habría sido necesaria en binario usual.
Comparar tb la eficiencia en general.
'''

print('Apartado ii)')
print()

X = 'medieval'

print("Palabra a tratar : {}".format(X))
print()

#Con binario usual (UTF-8)

CodXBin = codificarBinarioUsual(X)

print("Codificación de la palabra con binario usual (UTF-8): {}".format(CodXBin))
print("Longitud: {}".format(len(CodXBin)))
print()

#Con Seng
CodXEng = codificar(X,huffCodeEn)

print("Codificación de la palabra con Seng: {}".format(CodXEng))
print("Longitud: {}".format(len(CodXEng)))
print()

#Con Sesp
CodXEsp = codificar(X,huffCodeEsp)

print("Codificación de la palabra con Sesp: {}".format(CodXEsp))
print("Longitud: {}".format(len(CodXEsp)))
print()

'''
Apartado iii) : decodificar la palabra 101111011011101101110111111 con Seng
'''

print('Apartado iii)')
print()

CodYEng = '101111011011101101110111111'

print('Codificación a tratar : {}'.format(CodYEng))


#Canstruimos el hashmap inverso de huffCodeEng (la relación es biyectiva)
revHuffCodeEng = dict(reversed(item) for item in huffCodeEn.items())

#Decodificamos con Seng
Y = decodificar(CodYEng,revHuffCodeEng)

print("Decodificación: {}".format(Y))
print()
print()

'''
Codificar y decodificar una misma palabra con ambos lenguajes
'''

print('Codificar y decodificar una misma palabra:')
```

```
print()

#Con SEng
palabra = 'dragon'
print("Con SEng")
print("Palabra elegida: {}".format(palabra))

cod = codificar(palabra,huffCodeEn)
print("Codificada: {}".format(cod))

redecod = decodificar(cod,revHuffCodeEng)
print("Redecodificada: {}".format(redecod))
print()

#Con SEsp
palabra = 'queso'
print("Con SEsp")
print("Palabra elegida: {}".format(palabra))

cod = codificar(palabra,huffCodeEsp)
print("Codificada: {}".format(cod))

revHuffCodeEsp = dict(reversed(item) for item in huffCodeEsp.items())
redecod = decodificar(cod,revHuffCodeEsp)
print("Redecodificada: {}".format(redecod))
```

Apéndice B Resultado de la ejecución

```
Apartado i)

Lenguaje Seng:
==============
Codigo Huffman:
   : 00
 I : 01000
 ' : 0100100

 : 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: 4.158163265306121
Entropía : 4.117499394903036


Lenguaje Sesp:
==============
Codigo Huffman:
 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

  : 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: 4.431924882629109
Entropía : 4.394393861479968

Apartado ii)

Palabra a tratar : medieval

Codificación de la palabra con binario usual (UTF-8):
01101101011001010110010001101001011001010111011001100000101101100
Longitud: 64

Codificación de la palabra con Seng: 1111010111111011011111100011101111010011010110111 0
Longitud: 50

Codificación de la palabra con Sesp: 11000101000010110010100111010100110101
Longitud: 38

Apartado iii)

Codificación a tratar : 10111101101110110111011111
Decodificación: hello

Codificar y decodificar una misma palabra:

Con SEng
Palabra elegida: dragon
Codificada: 1101111110110011010100011111111111100
Redecodificada: dragon

Con SEsp
Palabra elegida: queso
Codificada: 101100110111010100000110
Redecodificada: queso