MEMORIA PRÁCTICA 1 - Código Huffman y Primer Teorema de Shannon

Gabriel Alba Serrano

1. Introducción

En ciencias de la computación y teoría de la información, la codificación Huffman es un algoritmo usado para compresión de datos, es decir, la reducción del volumen de datos tratables para representar una determinada información empleando una menor cantidad de espacio.

El concepto de codificación está fuertemente relacionado con el de entropía de la información, una variable aleatoria que mide la incertidumbre de una fuente de información. El objetivo de la práctica es programar y comprender la codificación Huffman, sus aplicaciones, y además ver que se satisface el Primer Teorema de Shannon que establece una cota inferior y superior de la longitud mínima posible de bits de información como función de la entropía.

Cabe destacar que la deflación y el códec multimedia como JPEG y MP3 tienen una cuantificación digital basada en la codificación de Huffman.

2. Material usado

Método: Utilizando el software de python3 he programado el algoritmo de Huffman para codificar sistemas de estados.

Datos:

- Los archivos de texto GCOM2023_pract1_auxiliar_eng y GCOM2023_pract1_auxiliar_esp, que representan los sistemas SEng y SEsp, respectivamente, y así mismo los estados y sus probabilidades.
- La plantilla de Python del campus virtual.
- La codificación en código binario usual de la palabra “dimension”. https://www.convertbinary.com/text-to-binary/

3. Resultados

(i) Las codificaciones de los sistemas SEng y SEsp al aplicar el algoritmo de Huffman se pueden encontrar en el documento Tabla.pdf

Sean H la entropía de un sistema (en este caso en base 2) y L la longitud media de un sistema y su codificación Huffman. A través del código en Python he obtenido que:

$H(SEng) = 4.4121 \pm \varepsilon_1$

$L(SEng) = 4.4369 \pm \varepsilon_1$

$H(SEsp) = 4.3383 \pm \varepsilon_2$

$L(SEsp) = 4.37130 \pm \varepsilon_2$

Siendo: $\varepsilon_1 = 2.2727E\text{-}02$, $\varepsilon_2 = 2.2222E\text{-}02$, los errores calculados por propagación cuadrática de errores, siendo en particular 1/N, con N el tamaño muestral de cada sistema de estados (SEng y SEsp).

Para los sistemas de estados SEng y SEsp se satisface el Primer Teorema de Shannon tal que:

$$H(S) \leq L(S) < H(S) + 1 \text{ , } S = \{SEng, SEsp\}$$

(ii) He obtenido que la codificación de “dimension” para

- SEng es: 00111010101110000100001000101011010000
- SEsp es: 11110001101100000100110100011100001001

La codificación en código binario usual es:
01100100 01101001 01101101 01100101 01101110 01110011 01101001 01101111 01101110
Se observa que la codificación en código binario usual es más extensa que el código binario Huffman, tanto de SEng como de SEsp. En particular es 2.1621621621621623 veces más extensa.

(iii) La decodificación de
“010101000110011100110111100010111111010101000111 0” es: isomorphism

(EXTRA) Para comprobar que las funciones codificar y decodificar que he definido en Python funcionan correctamente, he diseñado un test para SEng y SEsp, donde se codifican unas cadenas de caracteres y se decodifican. En todos los casos:
decodificar(codificar(‘string’)) = ‘string’

```
 ----- EXTRA -----

Para comprobar el correcto funcionamiento de la codificación Huffman, vamos a testear si
funcionan correctamente las funciones codificar y decodificar.
Test para SEng:
La cadena de caracteres "computational geometry" se ha codificado correctamente.
La cadena de caracteres "logistic attractor" se ha codificado correctamente.
La cadena de caracteres "convex hull" se ha codificado correctamente.
La cadena de caracteres "nonlinear dynamic system" se ha codificado correctamente.
La cadena de caracteres "chaos" se ha codificado correctamente.
La cadena de caracteres "devaney" se ha codificado correctamente.
La cadena de caracteres "exponente" se ha codificado correctamente.
La cadena de caracteres "lyapunov" se ha codificado correctamente.
La cadena de caracteres "fractal dimension" se ha codificado correctamente.
La cadena de caracteres "hausdorff" se ha codificado correctamente.
No hay errores al aplicar el test. Éxito!!!

Test para SEsp:
La cadena de caracteres "geometria computacional" se ha codificado correctamente.
La cadena de caracteres "atractor logistico" se ha codificado correctamente.
La cadena de caracteres "envolvente convexa" se ha codificado correctamente.
La cadena de caracteres "sistema dinamico no lineal" se ha codificado correctamente.
La cadena de caracteres "caos" se ha codificado correctamente.
La cadena de caracteres "devaney" se ha codificado correctamente.
La cadena de caracteres "exponente" se ha codificado correctamente.
La cadena de caracteres "lyapunov" se ha codificado correctamente.
La cadena de caracteres "medidas fractales" se ha codificado correctamente.
La cadena de caracteres "hausdorff" se ha codificado correctamente.
No hay errores al aplicar el test. Éxito!!!

Observaciones:
(1) Hay que tener en cuenta que SEng y SEsp no corresponden a los alfabetos completos de
sus respectivos idiomas, sino a la muestra de los archivos proporcionados en el campus
virtual, por lo tanto no podremos codificar todas las palabras del inglés y el español.
(2) Si codificamos un string en SEsp, al decodificarlo en SEng, obtendremos un error, y
viceversa.
```

4. Conclusión

Hemos podido comprobar que la Codificación Huffman funciona, y, es un óptimo sistema de compresión de datos, ya que reduce entorno a la mitad las longitudes de bits frente a otras codificaciones, como la binaria usual, y esto es una consecuencia directa del Primer Teorema de Shannon, que relaciona dichas longitudes (en concreto la media ponderada de todas ellas) con la entropía, la cuál puede interpretarse como el número de variables independientes de un sistema S que se usan para construir las partes de dicho sistema P(S).

5. Anexo (código en Python)

""" Voy a usar la plantilla del campus virtual como base para desarrollar funciones que resuelvan los apartados de la práctica. """

""" Voy a crear una función que represente el código binario de cada estado de un sistema, que se obtiene al aplicar el algoritmo de Huffman. Dicha función aceptará una lista de estados del sistema y un árbol de Huffman, y devolverá un diccionario, cuyas keys serán los estados del sistema y sus respectivos values sus expresiones en código binario Huffman.
La idea de la función es ver para cada estado, cuando aparece en los elementos del árbol y añadir un 0 ó un 1, dependiendo del respectivo value de la llave en la que aparece el estado. """

```
def codigo_huffman(estados,arbol):
    result = {}
    for i in range(len(estados)):
        a = estados[i]
        codigo = ''
        for j in range(len(arbol)):
            if a in list(arbol[j])[0]:
                codigo = '0' + codigo
            if a in list(arbol[j])[1]:
                codigo = '1' + codigo
        result[a] = codigo
    return result
```

""" Sean a[i] los diferentes estados de un sistema S, w[i] las probabilidades asociadas a cada estado a[i], y |c[i]| la longitud de cada cadena binaria c[i] obtenida aplicando el algoritmo de Huffman .
Sea W = suma(w[i]), en particular si w[i] son frecuencias relativas como en los sistemas SEng y SEsp, entonces W = 1.
Definimos la longitud media de un sistema S:
    L(S) = (1/W) * suma(w[i] * |c[i]|)
"""

```
def longitud_media(w,codigo):
    cadenas = list(codigo.values())
    result = 0
    for i in range(len(w)):
        result += w[i]*len(cadenas[i])
    return result

""" Definimos la entropía de un sistema. """

import math
def entropia_sistema(p): # p es una lista de las probabilidades de cada estado
    H = 0
    for i in range(len(p)):
        H -= p[i] * math.log2(p[i])
    return H

""" Para codificar una palabra, ó, varias palabras, concatenamos la
representación binaria de cada caracter (estado). La función codificar acepta
como argumentos de entrada una cadena de caracteres (estados) x, y un
diccionario cod, cuyos índices son los estados del sistema, y sus respectivos
valores sus representaciones en binario. """

def codificar(x,cod):
    y = list(x)
    result = ''
    for i in range(len(y)):
        result += cod[y[i]]
    return result

""" Para decodificar una representación en binario de la codificación Huffman,
se busca, empezando por la izquierda, la representación de un estado (caracter
de SEng o SEsp) y se añade a result, hasta que no haya ningún 0, ni ningún 1. """

def decodificar(x,cod):
    estados = list(cod.keys())
    cadenas = list(cod.values())
    y = list(str(x))
    result = ''
    i = 0
    while i<len(y):
        cad = ''
        # Para buscar la representación binaria de un estado vamos añadiendo
        # elementos de x, hasta que coincide con alguna cadena del diccionario cod
```

```
        while cad not in cadenas:
            cad += y[i]
            i += 1
        indice = cadenas.index(cad)
        result += estados[indice]
    return result

""" Finalmente este script va a imprimir por pantalla, las soluciones a las
preguntas de la práctica. Voy a declarar unas variables globales utilizando el
comando iloc para DataFrame. Las cuáles son listas que representan los estados
y las probabilidades del sistema SEng (alfabeto inglés), ordenadas en orden
ascendente según la probabilidad de cada estado. Además voy a declarar una
variable que representa el árbol de Huffman que se obtiene con la función de la
plantilla. Análogamente con SEsp (alfabeto español). """

estados_en = list(distr_en.iloc[:, 0])
prob_en = list(distr_en.iloc[:, 1])
arb_en = huffman_tree(distr_en)
estados_es = list(distr_es.iloc[:, 0])
prob_es = list(distr_es.iloc[:, 1])
arb_es = huffman_tree(distr_es)

huffman_en = codigo_huffman( estados_en, arb_en )
cod_en = np.array(list(huffman_en.values()))
huffman_es = codigo_huffman( estados_es, arb_es )
cod_es = np.array(list(huffman_es.values()))
long_en = longitud_media( prob_en, huffman_en)
long_es = longitud_media( prob_es, huffman_es)
entropia_en = entropia_sistema(prob_en)
entropia_es = entropia_sistema(prob_es)
e1 = format( 1/len(estados_en), '.4E')
e2 = format( 1/len(estados_es), '.4E')
dim_en = codificar('dimension', huffman_en)
dim_es = codificar('dimension', huffman_es)
dim_bin = '01100100 01101001 01101101 01100101 01101110 01110011 01101001
01101111 01101110'
x = '0101010001100111001101111000101111110101010001110'
decod_x = decodificar(x,huffman_en)

print('\n ----- APARTADO 1 ----- \n')
print('El código binario Huffman del sistema SEng es: \n')
print(pd.DataFrame({'States': np.array(estados_en), 'Coding': cod_en}))
print('\n')
print('El código binario Huffman del sistema SEsp es: \n')
```

```
print(pd.DataFrame({'Estados': np.array(estados_es), 'Codificación': cod_es}))
print('\n')
print('La longitud media de SEng es: ' + str(long_en) + ' ± ' + str(e1))
print('La longitud media de SEsp es: ' + str(long_en) + ' ± ' + str(e2))
print('La entropia H de SEng es: ' + str(entropia_en) + ' ± ' + str(e1))
print('La entropia H de SEsp es: ' + str(entropia_es) + ' ± ' + str(e2) + '\n')
print('Observamos que se satisface el Primer Teorema de Shannon porque:')
print('   H(S) <= L(S) < H(S)+1 , para S = SEng, SEsp \n')

print('\n ----- APARTADO 2 ----- \n')
print('La codificación de la palabra "dimension" en SEng es: ' + dim_en)
print('La codificación de la palabra "dimension" en SEsp es: ' + dim_es)
print('La codificación en código binario usual es: ' + dim_bin + '\n')
print('Observamos que la codificación en código binario usual es más extensa que el
código binario Huffman, tanto de SEng como de SEsp. En particular es ' + str(
format(len(dim_bin)/len(dim_en),'.4E') ) + ' veces más extensa.\n')

print('\n ----- APARTADO 3 ----- \n')
print('La decodificación de ' + x + ' es: ' + decod_x)



print('\n ----- EXTRA ----- \n')
print('Para comprobar el correcto funcionamiento de la codificación Huffman, vamos
a testear si funcionan correctamente las funciones codificar y decodificar. ')

print('Test para SEng:')
test_en = ['computational geometry', 'logistic attractor', 'convex hull', 'nonlinear
dynamic system', 'chaos', 'devaney', 'exponente', 'lyapunov', 'fractal dimension',
'hausdorff']
errores_en = 0
for k in test_en:
    codificacion = codificar(k, huffman_es)
    if k == decodificar( codificacion, huffman_es ):
        print('La cadena de caracteres "' + k + '" se ha codificado correctamente.')
    else:
        print('La cadena de caracteres ' + k + ' no se ha codificado correctamente.')
        errores_en += 1
if errores_en == 0:
    print('No hay errores al aplicar el test. Éxito!!! \n')
else:
    print('En total, al aplicar el test, hemos encontrado ' + errores_en + ' errores. \n')



print('Test para SEsp:')
```

```
test_es = ['geometria computacional', 'atractor logistico', 'envolvente convexa',
'sistema dinamico no lineal', 'caos', 'devaney', 'exponente', 'lyapunov', 'medidas
fractales', 'hausdorff']
errores_es = 0
for k in test_es:
    codificacion = codificar(k, huffman_es)
    if k == decodificar( codificacion, huffman_es ):
        print('La cadena de caracteres "' + k + '" se ha codificado correctamente.')
    else:
        print('La cadena de caracteres ' + k + ' no se ha codificado correctamente.')
        errores_es += 1
if errores_es == 0:
    print('No hay errores al aplicar el test. Éxito!!! \n')
else:
    print('En total, al aplicar el test, hemos encontrado ' + errores_es + ' errores. \n')

print('Observaciones: ')
print('(1) Hay que tener en cuenta que SEng y SEsp no corresponden a los alfabetos
completos de sus respectivos idiomas, sino a la muestra de los archivos
proporcionados en el campus virtual, por lo tanto no podremos codificar todas las
palabras del inglés y el español.')
print('(2) Si codificamos un string en SEsp, al decodificarlo en SEng, obtendremos un
error, y viceversa.')
```