Íñigo Iriondo Delgado
Geometría Computacional

Memoria Práctica 1

(1) Introducción

La práctica consiste en, a partir de las muestras dadas, hallar el código Huffman binario en inglés y español y sus longitudes medias, comprobando que se satisface el Primer Teorema de Shannon. Después, codificamos la palabra "dimension" en ambas lenguas y lo comparamos con el código binario usual. Finalmente, descodificamos en inglés "0101010001100111001101111000101111110101010001110".

(2) Material usado

Programamos en python utilizando la plantilla *GCOM2023-practica1_plantilla.py* que hace lo siguiente: 1) Abre los archivos *GCOM2023_pract1_auxiliar_eng.txt* y *GCOM2023_pract1_auxiliar_esp.txt* que serán nuestras muestras para el inglés y el español respectivamente. 2) Crea una distribución de estados (caracteres) que le asignamos una probabilidad dependiendo de las apariciones en el texto y los ordenamos con un Dataframe, guardándolo en un diccionario. 3) Define la función *huffman_branch(distr)* que hace lo mismo y *huffman_tree(distr)* que crea el árbol del algoritmo.
Creamos las siguientes funciones:

- *codificar1(tree,distr)*: devuelve una lista de strings de los códigos de Huffman de cada letra del árbol *tree* en el orden de la distribución *distr*.
- *error(distr,idioma)*: devuelve el error de la entropía.
- *shannon(L,H)*: devuelve un string que nos dice si se cumple el teorema de Shannon para longitud L y entropía H.
- *eng(palabra)*: codifica *palabra* en el código de Huffman inglés.
- *esp(palabra)*: codifica *palabra* en el código de Huffman español.
- *codificar2(tree,states)*: como *codificar1* pero añade el string de cada estado intercalando en la lista de forma que quedan junto a su código.
- *binario(palabra)*: codifica *palabra* en binario usual.
- *descodificar(binario,codificacion)*: descodifica la string binaria *binario* dado el codigo Huffman *codificacion*.

También completamos las distribuciones *distr* y *distr2* con sus correspondientes códigos, longitudes, pesos y entropías.

(3) Resultados

Si ejecutamos el programa obtenemos:

El codigo de Huffman de S_eng es ['V', '1001111100', 'H', '1001111101', ')', '1001111110', '(', '1001111111', '\n', '1110100110', 'j', '1110100111', '”', '1001111010', 'q', '1001111011', 'D', '100111011', 'z', '100111010', 'x', '111010010', 'I', '100111100', 'M', '100111001', 'b', '01111100', 'A', '01111101', 'B', '00100110', 'S', '00100111', 'k', '0010000', 'v', '0010001', 'T', '11101000', ',', '0010010', '.', '0111111', 'y', '1110101', 'w', '011110', 'g', '100110', 'f', '111011', 'p', '00101', 'u', '00110', 'd', '00111', 'm', '01110', 'c', '10010', 'l', '11100', 'r', '11110', 'h', '11111', 's', '0100', 'i', '0101', 'o', '0110', 'n', '1000', 't', '1010', 'a', '1011', 'e', '000', ' ', '110', 'C“', '100111000']
El codigo de Huffman de S_esp es ['V', '1011001101', 'H', '', ')', '1011011010', '(', '1011011011', '\n', '1011011000', 'j', '0110100', '”', '', 'q', '11101101', 'D', '011111100', 'z', '01101101', 'x', '1011001111', 'I', '111011000', 'M', '011111101', 'b', '0111100', 'A', '', 'B', '111011100', 'S', '01101100', 'k', '', 'v', '111011001', 'T', '01111111', ',', '11101111', '.', '0111110', 'y', '10110111', 'w', '', 'g', '1011000', 'f', '0110111', 'p', '111010', 'u', '10111', 'd', '11110', 'm', '01100', 'c', '11100', 'l', '0010', 'r', '11111', 'h', '1011010', 's', '1010', 'i', '0011', 'o', '1000', 'n', '1001', 't', '01110', 'a', '010', 'e', '000', ' ', '110', 'C“', '']
L(S_eng) es igual a 4.44 con un error de 0.05
L(S_esp) es igual a 4.69 con un error de 0.04
S_eng cumple el primer teorema de Shannon
S_esp cumple el primer teorema de Shannon
La codificacion en Huffman en inglés de dimension es 00111 0101 01110 000 1000 0100 0101 0110 1000 con longitud 46
La codificacion en Huffman en español de dimension es 11110 0011 01100 000 1001 1010 0011 1000 1001 con longitud 46
La codificacion en binario de dimension es 1100100 1101001 1101101 1100101 1101110 1110011 1101001 1101111 1101110 con longitud 72
La descodificacion en S_eng de 0101010001100111001101111000101111111010101000111 0 es isomorphism

(4) Conclusión

Aproximadamente, la longitud de S_eng es 4’44 y de S_esp es 4’69, se cumple el teorema de Shannon y la codificación en binario usual es más de un 56% más larga que la de Huffman de los dos casos.

(5) Anexo con el script

Código compartido con David Diez Roshan.

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

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

with open('GCOM2023_pract1_auxiliar_eng.txt', 'r',encoding="utf8") as file:
    en = file.read()

with open('GCOM2023_pract1_auxiliar_esp.txt', 'r',encoding="utf8") as file:
    es = file.read()

#### Contamos cuantos caracteres hay en cada texto
from collections import Counter
tab_en = Counter(en)
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_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))

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))

##### Para obtener una rama, fusionamos los dos states con menor frecuencia
distr = distr_en
```

```
".join(distr['states'][[0,1]])

### Es decir:
states = np.array(distr['states'])
probab = np.array(distr['probab'])
state_new = np.array([".join(states[[0,1]])])   #Ojo con: state_new.ndim
probab_new = np.array([np.sum(probab[[0,1]])])   #Ojo con: probab_new.ndim
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))

#Creamos un diccionario
branch = {'distr':distr, 'codigo':codigo}

## Ahora definimos una función que haga exáctamente lo mismo
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)

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)

distr = distr_en
tree = huffman_tree(distr)
tree[0].items()
```

```
tree[0].values()

#Buscar cada estado dentro de cada uno de los dos items
list(tree[0].items())[0][1] ## Esto proporciona un '0'
list(tree[0].items())[1][1] ## Esto proporciona un '1'

distr2 = distr_es
tree2 = huffman_tree(distr2)
tree2[0].items()
tree2[0].values()

#Buscar cada estado dentro de cada uno de los dos items
list(tree2[0].items())[0][1] ## Esto proporciona un '0'
list(tree2[0].items())[1][1] ## Esto proporciona un '1'



#Apartado 1

def codificar1(tree, distr):
    l = []
    for i in distr.states:
        cod = ""
        for j in range(len(tree)):
            if i in list(tree[j].items())[0][0]:
                cod = "0" + cod
            if i in list(tree[j].items())[1][0]:
                cod = "1"+ cod
        l.append(cod)
    return l

def codificar2(tree, states):
    l=[]
    for i in states:
        l.append(i)
        cod = ""
        for j in range(len(tree)):
            if i in list(tree[j].items())[0][0]:
                cod = "0" + cod
            if i in list(tree[j].items())[1][0]:
                cod = "1" + cod
        l.append(cod)
    return l

print('El codigo de Huffman de S_eng es',codificar2(tree,states))
```

```
print('El codigo de Huffman de S_esp es',codificar2(tree2,states))

distr['huffman'] = codificar1(tree,distr)
distr['long'] = distr.huffman.apply(lambda x: len(x))
distr['pesos'] = distr.long*distr.probab
distr['entropia'] = distr.probab.apply(lambda x: -x*math.log2(x))

distr2['huffman'] = codificar1(tree2,distr2)
distr2['long'] = distr2.huffman.apply(lambda x: len(x))
distr2['pesos'] = distr2.long*distr.probab
distr2['entropia'] = distr2.probab.apply(lambda x: -x*math.log2(x))

def error(distr,idioma):
    sum = 0
    for i in range(len(distr)):
        sum += (abs(math.log2(distr.probab[i]) + 1/math.log(2)))**2
    return ((1/len(idioma)**2)*sum)**(1/2)

print('L(S_eng) es igual a', round(distr.pesos.sum(),2), 'con un error de',
round(error(distr,en),2))
print('L(S_esp) es igual a', round(distr2.pesos.sum(),2), 'con un error de',
round(error(distr2,es),2))

def shannon(L,H):
    if H <= L and L <= H + 1:
        return 'cumple el primer teorema de Shannon'
    else:
        return 'no cumple el primer teorema de Shannon'

print('S_eng', shannon(distr.pesos.sum(),distr.entropia.sum()))
print('S_esp', shannon(distr2.pesos.sum(),distr2.entropia.sum()))

#Apartado 2

def eng(palabra):
    l = codificar2(tree,states)
    r = ""
    for letra in palabra:
        r = r + l[l.index(letra) + 1] + " "
    return r

def esp(palabra):
    l = codificar2(tree2,states)
    r = ""
```

```
    for letra in palabra:
        r = r + l[l.index(letra) + 1] + " "
    return r

def binario(palabra):
    f = ""
    for letra in palabra:
        b = bin(ord(letra))[2:]
        f += b + " "
    return f


print('La codificacion en Huffman en inglés de dimension es',eng("dimension"),'con
longitud',len(eng("dimension")))
print('La codificacion en Huffman en español de dimension es',esp("dimension"),'con
longitud',len(esp("dimension")))
print('La codificacion en binario de dimension es',binario("dimension"),'con
longitud',len(binario("dimension")))

#Apartado 3

def descodificar(binario, codificacion):
    result = ""
    cod = ""
    for i in binario:
        cod += i
        for j in range(0, len(codificacion), 2):
            if codificacion[j+1] == cod:
                result += codificacion[j]
                cod = ""
                break
    return result

print('La descodificacion en S_eng de
01010100011001110011011110001011111101010100011 10
es',descodificar("01010100011001110011011110001011111101010100011 10",
codificar2(tree, states)))
```