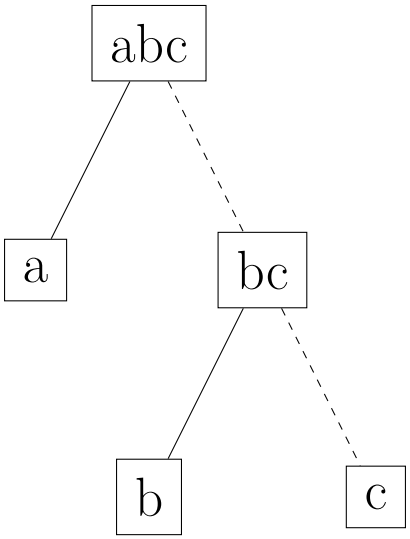
Práctica 2 - GCOMP

Celia Rubio Madrigal

23 de febrero de 2022

Índice



1. Introducción

En esta práctica queremos obtener un método de codificación y decodificación basado en árboles de Huffman para los lenguajes escritos del Español y del Inglés.

Tendremos en cuenta la probabilidad de cada estado, es decir, la frecuencia de aparición de cada letra en un texto. Así, los estados o letras más frecuentes se codificarán de manera más compacta, produciendo una codificación, en general, más breve.

En concreto, queremos minimizar la media de la longitud del código asociado a cada estado, ponderada por su frecuencia de aparición. A esta cantidad la llamaremos $L(C)$.

2. Material usado y metodología

Como no podemos acceder a la frecuencia absoluta de cada letra en cada idioma, disponemos de dos muestras de texto almacenadas en los archivos "`GCOM2022_pract2_auxiliar_eng.txt`" y "`GCOM2022_pract2_auxiliar_esp.txt`". Sirven como muestras representativas de las poblaciones totales de los idiomas, y tomaremos la frecuencia de cada estado en dichos textos como estimadores de la frecuencia real.

2.1. Apartado *i*)

Vamos a construir los dos árboles de Huffman a partir de las frecuencias de aparición de cada letra en sendos textos muestrales. Una vez obtenidas las frecuencias, mediante la función `distribution_from_file`, usamos un algoritmo voraz en la función `huffman_tree`. Agrupa los dos nodos de menor frecuencia en su suma, y se vuelve a insertar en la lista de nodos pendientes hasta que solo queda uno.

Para ayudar en las tareas de decodificación de apartados posteriores, y al estar creando el árbol de abajo hacia arriba, asociaremos a cada nodo el índice de sus dos hijos en el árbol. Además, crearemos un diccionario que asocie cada estado a su codificación con la función `codif_table`.

Después comprobaremos que nuestras dos codificaciones cumplen el 1º Teorema de Shannon. La entropía de nuestros sistemas, $H(C)$, es una medida de la cantidad de información contenida en ellos, y se calcula como:

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

donde P_j es la probabilidad de cada estado del sistema. El teorema dice que la media ponderada de la longitud de los códigos, $L(C)$, cumple:

$$H(C) \leq L(C) < H(C) + 1$$

2.2. Apartado *ii*)

Tras obtener los árboles de codificación, vamos a calcular el código asociado a la secuencia de estados dada por la palabra "`medieval`" en ambos idiomas. Además la codificaremos en binario, tomando como alfabeto la unión de todos los caracteres disponibles y asociando un código de longitud fija a cada letra. Así veremos cuál de los códigos es más eficiente en este caso concreto.

La función `codification` une las codificaciones de cada estado, que cumplen que ninguna de ellas es prefijo de otra. Eso asegura no tener ambigüedad al decodificarlas después.

2.3. Apartado *iii*)

Por último tenemos un texto codificado que proviene del inglés: "`10111101101110110111011111`". Lo decodificaremos con la función `decodification`, que usa las anotaciones previas de nodos hijo en el árbol.

Al crear los árboles de codificación, el lenguaje de Python decide resolver los empates en frecuencia de maneras arbitrarias. Para construir el mismo árbol que quien ha codificado el texto, también disponemos de su orden de estados, que impondremos en nuestro código al ordenar. Esta secuencia, en orden creciente, es: "`TpCmAkq;bfy?WBv'\ n-S,lr.dnIcwgiuaoseth` ".

3. Resultados y conclusiones

3.1. Apartado *i*)

A continuación, mostramos para ambos sistemas su medida de entropía, $H(C)$, el valor de la longitud media de su codificación hallada, $L(C)$, y el valor $1+H(C)$ para comprobar que $H(C) \leq L(C) < H(C) + 1$. Tomaré un margen de $\varepsilon = 0.001$ para englobar errores de computación.

Sistema C	**Entropía $H(C)$**	**Longitud media $L(C)$**	**$1 + H(C)$**
Inglés	4.117 $\pm$0.001	4.158 $\pm$0.001	5.117 $\pm$0.001
Español	4.394 $\pm$0.001	4.432 $\pm$0.001	5.394 $\pm$0.001

Con esto hemos atestado que el 1º Teorema de Shannon se satisface para nuestros sistemas.

3.2. Apartado *ii*)

Para codificar la palabra "`medieval`" en binario, se ha considerado un alfabeto de 50 letras, por lo que se codifica del 0 al 49, y se necesitan 6 bits por cada uno. Por tanto, la longitud de su código será $8 \cdot 6 = 48$. Mostramos la codificación para el binario, el inglés, y el español.

Sistema	**Longitud**	**Codificación**
Binario	48	010100101000100011010011101000011001110001100111
Inglés	50	11110101111110110111111000111011110100110101101110
Español	38	11000101000010110010100001111000110101

Aquí, el sistema de codificación español es más eficiente que el binario, y que a su vez, el inglés.

Hay que destacar que, para otro texto a codificar, podría haber salido un resultado distinto. Por ejemplo, hemos comprobado que la palabra `test` tiene codificaciones de longitud 24, 14 y 15, respectivamente. Una de las causas es que la letra `t` es más común en inglés que en español, y su codificación se reduce de tamaño 4 a 3.

3.3. Apartado *iii*)

Por último, la palabra secreta a decodificar es "`hello`".

Así, con las funciones programadas en esta práctica, somos capaces de codificar y decodificar textos en inglés y en español, haciendo uso del conocimiento de probabilidades que tenemos sobre ambos sistemas, y consiguiendo comprimir la información transmitida en muchos casos.

4. Código

```
'''
Práctica 2
'''
import numpy as np
import pandas as pd
from collections import Counter

# Me permite hacer el apartado iii) en las mismas condiciones que la plantilla.
# Columna extra para desambiguar el orden de los elems de igual probabilidad.
col = 'TpCmAkq;bfy?'WBv\n-S,lr.dnIcwgiuaoseth '
col = {l: i for i, l in enumerate(col)}

# Dado el nombre de un fichero, devuelve un Dataframe con columnas
# 'state' (letras) y 'probabs' (frecuencias normalizadas) ordenado
# de menor a mayor frecuencia. Se desambigua por col.
def distribution_from_file(path):
    contents = open(path, 'r', encoding='utf8').read()
    frecs = Counter(contents) # cuento la frecuencia de cada letra
    distr = pd.DataFrame(frecs.most_common(), columns=['state', 'probab'])
    distr.probab = distr.probab / float(sum(distr.probab)) # normalizo probabs
    distr['s'] = [col.get(l, 0) for l in distr.state] # iii)
    distr.sort_values(['probab', 's'], ascending=True, ignore_index=True,
        inplace=True)
    del distr['s'] # iii)
    return distr


# Realiza una iteración del algoritmo. Fusiona filas 0 y 1,
# introduce el nuevo elemento, y reordena de nuevo.
def huffman_branch(distr):
    code = {
        'value0': distr.state[0],
        'value1': distr.state[1],
        'child0': distr.child[0], # para recorrer el árbol luego
        'child1': distr.child[1], # para recorrer el árbol luego
    }
    distr.loc[len(distr.index)] = {
        'state': distr.state[0] + distr.state[1], # concat states
        'probab': distr.probab[0] + distr.probab[1], # sum probabs
        'child': len(distr.index) - 2, # para recorrer el árbol luego
    }
    distr = distr.drop([0, 1])
    distr.sort_values('probab', ascending=True, ignore_index=True, inplace=True)
    return distr, code


# Devuelve el árbol de Huffman de una tabla de letras y probabilidades.
```

```python
# Una lista que empieza en el 0 y se recorre con cada índice childx.
def huffman_tree(distr):
    distribution = distr.copy()
    distribution['child'] = None # si son hojas se quedarán así
    tree = []
    while len(distribution) > 1:
        distribution, code = huffman_branch(distribution)
        tree = [code] + tree
    return tree


# Dado un árbol de Huffman, devuelve un diccionario letra->código.
def codif_table(tree):
    codification = {}
    for code in tree:
        for l in code['value0']:
            codification[l] = codification.get(l, '') + '0'
        for l in code['value1']:
            codification[l] = codification.get(l, '') + '1'
    return codification


# Para el apartado ii)
# Dado un diccionario letra->código, devuelve la codificación de una palabra.
def codification(codif, word):
    return ''.join([codif[l] for l in word])


# Para el apartado iii)
# Dado un árbol de Huffman y una palabra codificada, devuelve su original.
# El árbol es una lista cuya raíz es el primer elemento.
# Cada elem: Valor (0), Valor (1), Índice del hijo (0) e Índice del hijo (1).
def decodification(tree, code):
    word = ''
    i = 0
    for c in code:
        if not tree[i]['child' + c]:
            word += tree[i]['value' + c]
            i = 0
        else:
            i = tree[i]['child' + c]
    return word


# Longitud media: sum( wi |ci| ) con wi la probabilidad normalizada y ci el
    código de cada letra
def get_L(distr, codif):
    return sum(distr.probab * [len(codif[s]) for s in distr.state])
```

```python
# Entropía
def get_H(distr):
    return -sum(distr.probab * np.log2(distr.probab))


''' Apartado i) '''
print('-' * 10)

distr_eng = distribution_from_file('GCOM2022_pract2_auxiliar_eng.txt')
tree_eng = huffman_tree(distr_eng)
codif_eng = codif_table(tree_eng)
L_eng = get_L(distr_eng, codif_eng)
H_eng = get_H(distr_eng)

print('Inglés:\t\tH =', H_eng, '  L =', L_eng, '< H+1 =', H_eng + 1)

distr_esp = distribution_from_file('GCOM2022_pract2_auxiliar_esp.txt')
tree_esp = huffman_tree(distr_esp)
codif_esp = codif_table(tree_esp)
L_esp = get_L(distr_esp, codif_esp)
H_esp = get_H(distr_esp)

print('Español:\tH =', H_esp, '  L =', L_esp, '< H+1 =', H_esp + 1)


''' Apartado ii) '''
print('-' * 10)

word = 'medieval'

# Creamos la tabla de codificación binaria para el conjunto de todas las letras
     de los textos.
# La tabla no hace falta para saber la longitud de la palabra codificada, pues
    será
# su longitud sin codificar (8) * la longitud en binario de (len(Alfabeto)-1)
all_letters = set(distr_eng.state).union(set(distr_esp.state))
size_bin = len(bin(len(all_letters) - 1)[2:])
codif_bin = {l: bin(i)[2:].zfill(size_bin) for i, l in enumerate(all_letters)}

# Resultados:
result_bin = codification(codif_bin, word)
result_eng = codification(codif_eng, word)
result_esp = codification(codif_esp, word)

print('Codificación de la palabra:', word)
print('En binario (hay', len(all_letters), 'letras):', len(word), '*', size_bin
    , '=', len(word) * size_bin, result_bin)
print('Huffman inglés:\t\t\t', len(result_eng), result_eng)
```

```
print('Huffman español:\t\t', len(result_esp), result_esp)

print("Comprobación:",decodification(tree_eng,result_eng),decodification(
    tree_esp,result_esp))

''' Apartado iii) '''
print('-' * 10)

code = '10111101101110110111011111'
word = decodification(tree_eng, code)
print('Decodificación del código:', code)
print(word)
```