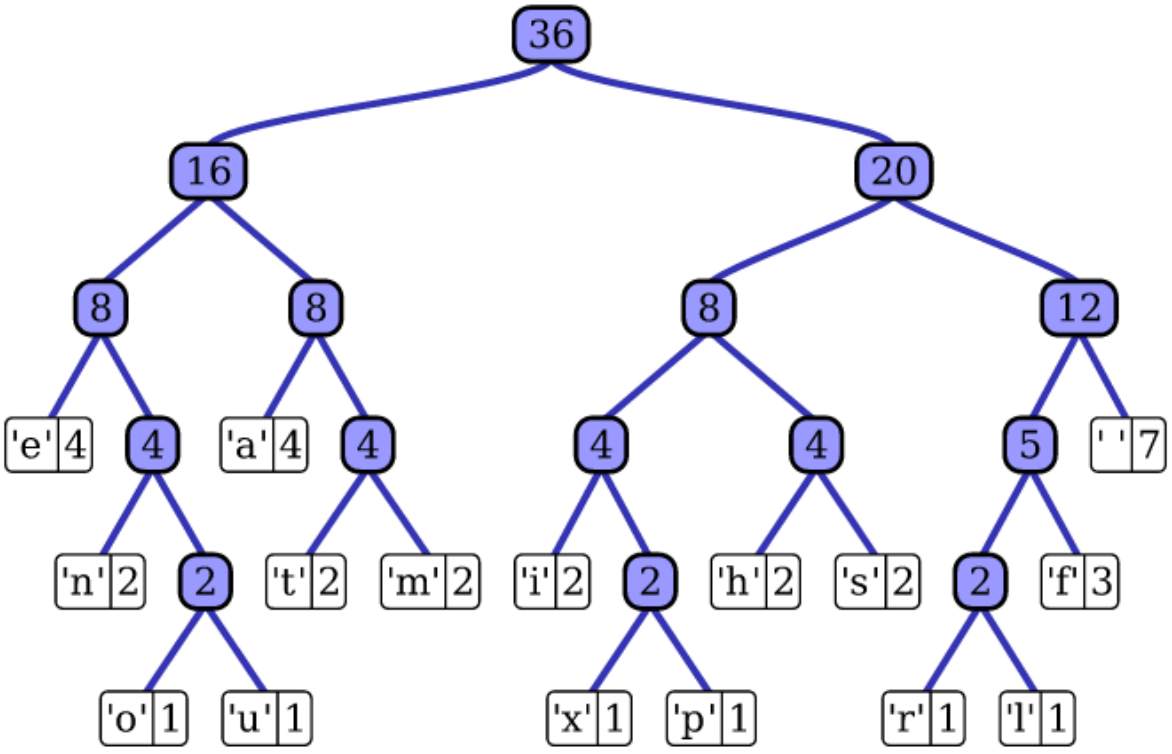
GEOMETRÍA COMPUTACIONAL

PRÁCTICA 2



DAVID SEIJAS PÉREZ

1. Introducción

En esta práctica vamos a trabajar con muestras de dos variables aleatorias y aprender y familiarizarnos con el algoritmo de Huffman. Para esto, nuestro objetivo será ver cómo se utiliza este algoritmo y conseguir codificar y decodificar distintas palabras gracias al código binario hallado de las muestras.

2. Material Usado

Para el desarrollo de esta práctica he utilizado *GCOM2022_pract2_auxiliar_eng.txt* y *GCOM2022_pract2_auxiliar_esp.txt* donde teníamos las muestras de cada población. Además, he usado el archivo *GCOM2022-Practica2_plantilla.py* como plantilla, de donde he adaptado algunas funciones para hallar los árboles de Huffman utilizando listas y diccionarios. Por último, he requerido del empleo de las librerías *math* y *pandas* para la realización de algún cálculo matemático y el uso de DataFrames respectivamente.

3. Metodología

Para realizar esta práctica, primero obtuve y analicé la plantilla dada para entender y asimilar el manejo de diccionarios y los árboles de Huffman representados con este tipo de objetos. Una vez tuve esto claro, desarrollé varias funciones para conseguir los objetivos.

Estas funciones son *Huffman_dict*, que la he usado para traducir el árbol de Huffman en un diccionario carácter:código, *codificar* y *decodificar*, necesarias para los apartados 2 y 3. Primero, para crear el diccionario miro las dos ramas del nodo raíz inicializando así cada caracter de los textos con su valor 0 o 1 según la rama a la que corresponda. Posteriormente, recorro todas las ramas del árbol mirando para cada una todos los caracteres que hay y añadiendo a sus códigos los 0s o 1s correspondientes según la rama a la que pertenezcan. Para codificar, por cada caracter de la palabra dada obtengo y concateno a la salida su código asociado en el diccionario. Para decodificar es necesario invertir el diccionario, obteniendo un diccionario código:caracter, y con este ir viendo a qué letra se traducen la acumulación de los dígitos del código, reseteando esta cuando se corresponde con la traducción de un caracter.

Para el primer apartado, he traducido los árboles de Huffman en los diccionarios nombrados anteriormente y, con estos, he podido traducir una a una las letras aparecidas en los textos. Para las longitudes medias y entropías he plasmado en código las fórmulas mostradas en los apuntes.

Para el segundo y tercer apartado, solo he tenido que usar las funciones *codificar* y *decodificar* eligiendo el idioma en que quería realizar estas acciones.

4. Resultados

1. En el primer apartado he obtenido que se satisfacía el *Primer Tª de Shannon* en ambos casos con los resultados siguientes:

   S_{en} : **Longitud Media:** 4.158163265306123
   Entropía: 4.117499394903037
   Código Binario del texto: $111101010010111011001110111000100111100001110$...
   ...$1001001011100011011101101001111101111011010 0101$

   S_{es} : **Longitud Media:** 4.431924882629108
   Entropía: 4.3943938614799665
   Código Binario del texto:$000111000110100101000111110000110101011111011$...
   ...$1011010111010100011100000011101000010010110 1110$

2. En el segundo apartado he obtenido la codificación siguiente para la palabra **"medieval"**:

   S_{en} : $11110101111110110111111000111011110100110101101110$

   S_{es} : $11000101000010110010100111010100110101$

   La longitud de la codificación de esta palabra con el código binario usual sería la siguiente: tenemos 38 y 45 elementos en los diccionarios inglés y español respectivamente; por tanto, necesitaríamos 6 bits para codificar cada caracter pues $2^5 = 32 < 38 < 45 < 2^6 = 64$. Como "medieval" tiene 8 letras y cada letra serían 6 bits, tendríamos que medieval se codificaría con 48 bits. Por tanto, saldrían menos bits que en la codificación Huffman con S_{en} (50 bits) y más que con S_{es} (38 bits).

3. En el tercer apartado he visto que la palabra en inglés decodificada de $10111101101110110111011111$ es ***hello***.

5. Conclusión

Con esta práctica he aprendido, especialmente, varias cosas. La primera, aunque no muy relacionada con la asignatura en realidad, es la utilización y manejo de diccionarios en Python. Además, he aprendido cómo se traduce el algoritmo de Huffman y los distintos usos que se le puede dar. Como vimos, *"la entropía de un sistema es asimilable al número de bits que se requiere para representar el sistema"*; por tanto, esta codificación de Huffman está claramente relacionada con este concepto pues es una codificación binaria que usa de bits para representar un sistema, o muestras de una variable aleatoria, con el fin de hallar probabilidades, frecuencias o apariciones de los distintos estados en el sistema. En definitiva, hemos aprendido en profundidad una codificación que, según los resultados vistos en el apartado 2, depende de la variable tomada, pero puede llegar a ser más eficaz que la codificación binaria usual.

6. Anexo: Código

```
"""
DAVID SEIJAS PEREZ
Practica 2
"""

import math
import pandas as pd


with open('GCOM2022_pract2_auxiliar_eng.txt', 'r', encoding="utf8") as
    file: en = file.read()

with open('GCOM2022_pract2_auxiliar_esp.txt', 'r', encoding="utf8") as
    file: es = file.read()

from collections import Counter
tab_en = Counter(en)
tab_es = Counter(es)

tab_en_states = list(tab_en)
tab_en_weights = list(tab_en.values())
tab_en_probab = [x/float(sum(tab_en_weights)) for x in 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 = [i for i in range(len(tab_en_states))]

tab_es_states = list(tab_es)
tab_es_weights = list(tab_es.values())
tab_es_probab = [x/float(sum(tab_es_weights)) for x in 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 = [i for i in range(len(tab_es_states))]


def huffman_branch(distr):
    states = list(distr['states'])
    probab = list(distr['probab'])
    state_new = [''.join(states[0:2])]
    probab_new = [probab[0] + probab[1]]
    codigo = list([{states[0]: 0, states[1]: 1}])
    states = states[2:] + state_new
    probab = probab[2:] + probab_new
    distr = pd.DataFrame({'states': states, 'probab': probab, })
    distr = distr.sort_values(by='probab', ascending=True)
    distr.index = [i for i in range(len(states))]
    branch = {'distr':distr, 'codigo':codigo}
    return(branch)


```

```
48  def huffman_tree(distr):
49      tree = []
50      while len(distr) > 1:
51          branch = huffman_branch(distr)
52          distr = branch['distr']
53          code = branch['codigo']
54          tree += code
55      return(tree)
56
57  tree_en = huffman_tree(distr_en)
58  tree_es = huffman_tree(distr_es)
59
60
61
62  '''
63  Funcion que crea un diccionario recorriendo las ramas y asignando 0 o
        1 a cada letra segun la rama a la que pertenezca
64  '''
65  def huffman_dict(tree):
66          dicc = {}
67          for c in list(tree[len(tree)-1].items())[0][0]:
68              dicc[c] = "0"
69          for c in list(tree[len(tree)-1].items())[1][0]:
70              dicc[c] = "1"
71
72          i = len(tree) - 2
73          while i >= 0:
74              for j in list(tree[i].items())[0][0]:
75                  dicc[j] += "0"
76              for k in list(tree[i].items())[1][0]:
77                  dicc[k] += "1"
78              i -= 1
79
80          return(dicc)
81
82
83  dicc_en = huffman_dict(tree_en)
84  dicc_es = huffman_dict(tree_es)
85
86
87  '''
88  Funcion que codifica una palabra dada en un idioma dado
89  '''
90  def codificar(palabra, idioma):
91      codificacion = ""
92      if(idioma == "I"):
93          dicc = dicc_en
94      if(idioma == "E"):
95          dicc = dicc_es
96
97      for c in palabra:
98          codificacion += dicc[c]
99
100     return(codificacion)
```

```
101
102
103 '''
104 Funcion que decodifica una palabra a partir de un codigo y un idioma
        dado
105 '''
106 def decodificar(codigo, idioma):
107     decodificacion = ""
108     if(idioma == "I"):
109         dicc = {v:k for k, v in dicc_en.items()}
110     if(idioma == "E"):
111         dicc = {v:k for k, v in dicc_es.items()}
112
113     aux = ""
114     for c in codigo:
115          aux += c
116          if aux in dicc:
117              decodificacion += dicc[aux]
118              aux = ""
119
120     return(decodificacion)
121
122
123
124 def apartado1():
125     codif_en = codif_es = ""
126     for c in en:
127         codif_en += dicc_en[c]
128     for c in es:
129         codif_es += dicc_es[c]
130
131     print("Codigo Huffman binario de S_en:")
132     print(codif_en)
133     print("Codigo Huffman binario de S_es:")
134     print(codif_es)
135     print("\n")
136
137     long_en = long_es = 0
138     for i in range(len(distr_en)):
139         long_en += distr_en['probab'][i]*len(dicc_en[distr_en['states'
                ][i]])
140     for i in range(len(distr_es)):
141         long_es += distr_es['probab'][i]*len(dicc_es[distr_es['states'
                ][i]])
142
143     print("Longitud media de S_en:")
144     print(long_en)
145     print("Longitud media de S_es:")
146     print(long_es)
147     print("\n")
148
149     entr_en = entr_es = 0
150     for i in range(len(distr_en)):
```

```
151             entr_en -= distr_en['probab'][i]*math.log(distr_en['probab'][i
                    ], 2)
152         for i in range(len(distr_es)):
153             entr_es -= distr_es['probab'][i]*math.log(distr_es['probab'][i
                    ], 2)
154 
155         print("Entropia de S_en:")
156         print(entr_en)
157         print("Entropia de S_es:")
158         print(entr_es)
159         print("\n")
160 
161         if(entr_en <= long_en < entr_en + 1):
162             print("Se verifica el 1er Teorema de Shannon para S_en")
163         else:
164             print("NO se verifica el 1er Teorema de Shannon para S_en")
165         if(entr_es <= long_es < entr_es + 1):
166             print("Se verifica el 1er Teorema de Shannon para S_es")
167         else:
168             print("NO se verifica el 1er Teorema de Shannon para S_es")
169 
170 
171     def apartado2(palabra):
172         codif_en = codificar(palabra, "I")
173         codif_es = codificar(palabra, "E")
174         print("Codificacion de " + palabra + " en Ingles:")
175         print(codif_en)
176         print("Codificacion de " + palabra + " en Espa ol:")
177         print(codif_es)
178 
179 
180     def apartado3(codigo):
181         decodif_en = decodificar(codigo, "I")
182         print("Decodificacion de " + codigo + " en Ingles:")
183         print(decodif_en)
184 
185 
186     apartado1()
187     print("\n")
188     print("----------------------")
189     print("\n")
190     apartado2("medieval")
191     print("\n")
192     print("----------------------")
193     print("\n")
194     apartado3("10111101101110110111011111")
```