Código Huffman y primer teorema de Shannon

Sergio González Montero

12 de febrero de 2024

1. Objetivo

A partir de los textos dados, se obtendrá el código Huffman binario de cada caracter con al menos una aparición en el documento. Además, se calcularán las longitudes medias las cuales se usarán para comprobar que se satisface el Primer Teorema de Shannon. Lo siguiente será, con los códigos realizados, codificar la palabra *Lorentz* en ambas lenguas y comparar las longitudes de las cadenas frente a la codificación binaria usual; calcular la entropía y los errores asociados a ambas características. Para terminar, se crea una función para decodificar cualquier palabra (cadena de Huffman) del texto proporcionado en ambos idiomas.

2. Material y datos

Se usaron los siguientes apartados de las notas de Robert Monjo de la asignatura de Geometría Computacional, a día 07/02/2024:

1. Definición 1.1.12 : Entropía de un sistema
2. Algoritmo 1.2.2 : Huffman
3. Teorema 1.2.1 : Primer Teorema de Shannon
4. Ejercicio 2.1 : Métrica de errores

Se usaron los archivos *GCOM2024_pract1_auxiliar_eng.txt* y *GCOM2024_pract1_auxiliar_esp.txt* proporcionados por el profesor, que contienen un texto sobre la transformada de Lorentz en inglés y en español, respectivamente.
En cuanto al código, es original. En él, se han utilizado las librerías siguientes:

1. collections.Counter: subclase de diccionario para contar objetos hashables
2. heapq: proporciona una implementación del algoritmo de cola de pila, también conocido como el algoritmo de cola de prioridad. heapq.nsmallest devuelve los n elementos más pequeños
3. math: proporciona acceso a las funciones matemáticas

3. Resultados

Los resultados obtenidos tras ejecutar el archivo .py han sido:
Código Huffman de S_{eng}: [('x', '100111100'), ('S', '100111101'), (É', '100111110'), ('R', '100111111'), ('4', '00101110'), ('-', '00101111'), (Á', '10011010'), ('T', '10011011'), ('L', '10011100'), ('M', '10011101'), ('z', '11110100'), ('q', '11110101'), ('g', '11110110'), (',', '11110111'), ('k', '0010110'), ('.', '1001100'), ('w', '1111000'), ('b', '1111001'), ('y', '1111110'), ('v', '1111111'), (ú', '001010'), ('p', '111110'),

('c', '00100'), ('f', '00110'), ('l', '00111'), ('h', '01010'), ('d', '01011'), ('m', '10010'), ('r', '0100'), (ó', '0110'), (á', '0111'), ('s', '1000'), ('t', '1010'), (ñ', '1011'), (í', '1110'), (é', '000'), (", '110')]
Código Huffman de S_{esp}: [('x', '001111000'), ('P', '001111001'), (É', '001111010'), ('j', '001111011'), ('R', '001111100'), ('í', '001111101'), ('4', '001111110'), ('C', '001111111'), ('M', '111111100'), ('w', '111111101'), ('g', '111111110'), ('y', '111111111'), ('h', '00111000'), ('é', '00111001'), ('k', '00111010'), ('-', '00111011'), ('q', '11011110'), ('L', '11011111'), (',', '11111100'), ('.', '11111101'), ('b', '1101100'), ('z', '1101101'), ('ó', '1101110'), ('f', '1111100'), ('v', '1111101'), ('p', '00110'), ('m', '01110'), (ú', '01111'), ('l', '11010'), ('d', '11100'), ('t', '11101'), ('c', '11110'), ('s', '0010'), ('r', '0110'), (ñ', '1010'), (ó', '1011'), (í', '1100'), (á', '000'), (é', '010'), (", '100')]
Las longitudes medias son: $L_{en} = 4.305$ con error de entropía 0.056 y error de longitud 0.068
$L_{esp} = 4.304$ con error de entropía 0.052 y error de longitud 0.064
Teorema de Shannon: Entropía <= Longitud media <= Entropía + 1
Inglés: 4.2827303843795885 <= 4.305460750853242 <= 5.2827303843795885
S_{en} cumple el teorema de Shannon
Español: 4.27846906600711 <= 4.304285714285715 <= 5.27846906600711
S_{esp} cumple el teorema de Shannon
Lorentz en la codificación en inglés: 10011100011001000001011101011110100, con longitud 35
Lorentz en la codificación en español: 11011111101101100101010111011101101, con longitud 35
Lorentz en la codificación binaria: 1001100110111111100101100101110111011101001111010, con longitud 49
La codificación binaria es 1.4 más larga respecto a la codificación inglesa
La codificación binaria es 1.4 más larga respecto a la codificación española
Tomando las codificaciones de Lorentz del apartado anterior:
Decodificado de 10011100011001000001011101011110100 en inglés: Lorentz
Decodificado de 11011111101101100101010111011101101 en español: Lorentz

4. Conclusión

La codificación de Huffman es igual o más eficiente que la binaria trivial, como se ve en nuestro caso particular de *Lorentz*. Se aprecia también que la cota máxima a cuánto podemos comprimir el mensaje de un emisor sin perder información está relacionada con su entropía (Shannon, 1948).

5. Código

Programa 1: practica1.py

```
1 from collections import Counter
2 import heapq
```

```
3  import math
4  from math import sqrt as sqrt
5
6  print("---------------\nPRIMER APARTADO \n---------------")
7
8  with open("GCOM2024_pract1_auxiliar_eng.txt", 'r', encoding='utf-8') as f:
9      en = f.read()
10
11 with open("GCOM2024_pract1_auxiliar_esp.txt", 'r', encoding='utf-8') as f:
12     esp = f.read()
13
14 with open("lorentz.txt", 'r', encoding='utf-8') as f:
15     lorentz = f.read()
16
17 def frec():
18
19     # Filtrar caracteres
20     caracter_en = [c for c in en]
21     caracter_esp = [c for c in esp]
22
23     # Contar la frecuencia de cada elemento
24     frec_en = dict(sorted(Counter(caracter_en).items(),
25     key=lambda item:item[1]))
26     frec_esp = dict(sorted(Counter(caracter_esp).items(),
27     key=lambda item:item[1]))
28
29     return frec_en, frec_esp
30
31 frec_en, frec_esp = frec()[0], frec()[1]
32 frec_en_char, frec_esp_char = frec()[0], frec()[1]
33 k_esp, k_en = [], []
34 c_esp, c_en = [], []
35
36 k_esp, k_en = [], []
37 c_esp, c_en = [], []
38
39 def tree_nodes():
40     """
41     c_esp : list; dupla (nodo, {0,1}) de caracteres en espanol
42     c_en : list; dupla (nodo, {0,1}) de caracteres en ingles
43     """
44
45     while len(frec_esp) > 1:
46
47         # Los dos valores mas pequenos y
48         # claves con los valores mas pequenos
```

```
49          svalue_esp = heapq.nsmallest(2, frec_esp.values())
50          skey_esp = [clave for clave, valor in frec_esp.items()
51                      if valor in svalue_esp]
52          k1_esp, k2_esp = skey_esp[0], skey_esp[1]
53
54          # Todos los nodos no terminales del arbol
55          k_esp.append(k1_esp + k2_esp)
56
57          # Codificado del arbol
58          c_esp.append((k1_esp, 0))
59          c_esp.append(((k2_esp, 1)))
60
61          # Actualiza el diccionario
62          frec_esp.update({k1_esp + k2_esp: sum(svalue_esp)})
63          frec_esp.pop(k1_esp)
64          frec_esp.pop(k2_esp)
65
66          # Ordena el diccionario
67          frec_esp1 = dict(sorted(Counter(frec_esp).items(),
68                                  key=lambda item: item[1]))
69
70      while len(frec_en) > 1:
71
72          # Los dos valores mas pequenos y
73          # claves con los valores mas pequenos
74          svalue_en = heapq.nsmallest(2, frec_en.values())
75          skey_en = [clave for clave, valor in frec_en.items()
76                     if valor in svalue_en]
77          k1_en, k2_en = skey_en[0], skey_en[1]
78
79          # Todos los nodos no terminales del arbol
80          k_en.append(k1_en + k2_en)
81
82          # Codificado del arbol
83          c_en.append((k1_en, 0))
84          c_en.append(((k2_en, 1)))
85
86          # Actualiza el diccionario
87          frec_en.update({k1_en + k2_en: sum(svalue_en)})
88          frec_en.pop(k1_en)
89          frec_en.pop(k2_en)
90
91          # Ordena el diccionario
92          frec_en1 = dict(sorted(Counter(frec_en).items(),
93                                 key=lambda item: item[1]))
94
```

```
95      return c_esp, c_en
96
97  tree_esp, tree_en = tree_nodes()[0], tree_nodes()[1]
98
99  def encode_alph():
100     """
101     dictionary_esp : list; codificacion Huffman de
102                                 caracteres en espanol
103     dictionary_en : list; codificacion Huffman de
104                                 caracteres en ingles
105     """
106     # Listas de claves
107     chars_esp = [x for x in frec_esp_char.keys()]
108     chars_en = [x for x in frec_en_char.keys()]
109     # Listas de nodos
110     nodes_esp = [node[0] for node in tree_esp]
111     nodes_en = [node[0] for node in tree_en]
112
113     dictionary_esp, dictionary_en = [], []
114
115     for char in chars_esp:
116
117         huffman_binary = ""
118
119         for i in range(len(tree_esp)):
120             if char in nodes_esp[i]:
121                 huffman_binary = str(tree_esp[i][1]) + huffman_binary
122
123         dictionary_esp.append((char, huffman_binary))
124
125     for char in chars_en:
126         huffman_binary = ""
127
128         for i in range(len(tree_en)):
129             if char in nodes_en[i]:
130                 huffman_binary = str(tree_en[i][1]) + huffman_binary
131
132         dictionary_en.append((char, huffman_binary))
133
134     return dictionary_esp, dictionary_en
135
136 dict_esp, dict_en = encode_alph()[0], encode_alph()[1]
137 print(f"Codigo Huffman de S_eng: \n{dict_en}\n\n\
138                 Codigo Huffman de S_esp:\n{dict_esp}\n")
139
140 # Listas de codigo Huffman de cada caracter
```

```
141 huffman_en, huffman_esp = [],[]
142 for duple in dict_en:
143     huffman_en.append(duple[1])
144 
145 for duple in dict_esp:
146     huffman_esp.append(duple[1])
147 
148 def probabilidades():
149     """
150     probabilidad_en : list; frecuencias relativas de caracteres en ingles
151     probabilidad_esp : list; frecuencias relativas de caracteres en espanol
152     """
153 
154     N_en, N_esp = len(en), len(esp)
155     probabilidad_en = list(map(lambda x: x/N_en,
156     list((frec_en_char.values()))))
157     probabilidad_esp = list(map(lambda x: x/N_esp,
158     list((frec_esp_char.values()))))
159 
160     return probabilidad_en, probabilidad_esp
161 
162 W_en, W_esp = probabilidades()[0], probabilidades()[1]
163 
164 def longitud_media():
165     """
166     L_en : float; longitud media del codigo Huffman en ingles
167     L_esp : float; longitud media del codigo Huffman en espanol
168     """
169     L_en, L_esp = 0, 0
170 
171     for i in range (len(W_en)):
172         L_en += W_en[i] * len(dict_en[i][1])
173 
174     for i in range (len(W_esp)):
175         L_esp += W_esp[i] * len(dict_esp[i][1])
176 
177     return L_en, L_esp
178 
179 def error_longitud():
180     """
181     error_en : float; error de longitud en ingles
182     error_esp : float; error de longitud en espanol
183     """
184 
185     e_en, e_esp = 0, 0
186 
```

```
187     for i in huffman_en:
188         e_en += (abs(len(i))**2)
189 
190 
191     for i in huffman_esp:
192         e_esp += (abs(len(i))**2)
193 
194     error_en = (1/len(en))*sqrt(e_en)
195     error_esp = (1/len(esp))*sqrt(e_esp)
196 
197     return error_en, error_esp
198 
199 def error_entropia():
200     """
201     error_en : float; error de entropia en ingles
202     error_esp : float; error de entropia en espanol
203     """
204     e_en, e_esp = 0, 0
205 
206     for i in range(len(frec_en_char)):
207         e_en += (abs(math.log2(W_en[i]) + 1/math.log(2)))**2
208 
209     for i in range(len(frec_esp_char)):
210         e_esp += (abs(math.log2(W_esp[i]) + 1/math.log(2)))**2
211 
212     error_en = (1/len(en))*sqrt(e_en)
213     error_esp = (1/len(esp))*sqrt(e_esp)
214 
215     return error_en, error_esp
216 
217 error_en_long, error_esp_long = error_longitud()[0], error_longitud()[1]
218 error_en, error_esp = error_entropia()[0], error_entropia()[1]
219 L_en, L_esp = longitud_media()[0], longitud_media()[1]
220 
221 print(f"Las longitudes medias son: \nL_en = {round(L_en, 3)} con error de\
222  entropia {round(error_en, 3)} y error de longitud {round(error_en_long,3)}
223       \nL_esp = {round(L_esp, 3)} con error de entropia {round(error_esp, 3)
224  y error de longitud {round(error_esp_long,3)}")
225 
226 def entropia():
227     """
228     -entropia_en : float; entropia total del sistema, caso ingles
229     -entropia_esp : float; entropia total del sistema, caso espanol
230     """
231     entropia_en, entropia_esp = 0, 0
232 
```

```
233     for i in range(len(W_en)):
234         entropia_en += W_en[i]*(math.log2(W_en[i]))
235 
236     for i in range(len(W_esp)):
237         entropia_esp += W_esp[i]*(math.log2(W_esp[i]))
238 
239     return -entropia_en, -entropia_esp
240 
241 entropia_en, entropia_esp = entropia()[0], entropia()[1]
242 
243 def shannon_th(L_en, L_esp):
244     """
245     L_en : float; longitud media del codigo Huffman en ingles
246     L_esp : float; longitud media del codigo Huffman en espanol
247     check_shannon_en : str; print si cumple el
248                           teorema de Shannon, caso ingles
249     check_shannon_esp : str; print si cumple el
250                           teorema de Shannon, caso espanol
251     """
252 
253     if entropia_en <= L_en and L_en <= entropia_en + 1:
254         check_shannon_en = "S_en cumple el teorema de Shannon"
255     else: check_shannon_en = "S_en no cumple el teorema de Shannon"
256 
257     if entropia_esp <= L_esp and L_esp <= entropia_esp + 1:
258         check_shannon_esp = "S_esp cumple el teorema de Shannon"
259     else: check_shannon_esp = "S_esp no cumple el teorema de Shannon"
260 
261     return check_shannon_en, check_shannon_esp
262 
263 shannon_en = shannon_th(L_en, L_esp)[0]
264 shannon_esp = shannon_th(L_en, L_esp)[1]
265 print(f"\nTeorema de Shannon: Entropia <= Longitud media <= Entropia + 1\
266       \nIngles: {entropia_en} <= {L_en} <= {entropia_en + 1}\
267     \n{shannon_en}\
268       \nEspanol: {entropia_esp} <= {L_esp} <= {entropia_esp + 1}\
269      \n{shannon_esp}")
270 
271 print("\n----------------\nSEGUNDO APARTADO \n----------------")
272 
273 def encode():
274     """
275     en_encode : str; codigo Huffman de texto ingles
276     esp_encode : str; codigo Huffman de texto espanol
277     lorentz_en : str; codigo Huffman de Lorentz en ingles
278     lorentz_esp : str; codigo Huffman de Lorentz en espanol
```

```
279     """
280     en_encode  = ""
281     for char in en:
282         for i in range(len(dict_en)):
283             if char == dict_en[i][0]:
284                 en_encode += dict_en[i][1]
285 
286     esp_encode  = ""
287     for char in esp:
288         for i in range(len(dict_esp)):
289             if char == dict_esp[i][0]:
290                 esp_encode += dict_esp[i][1]
291 
292 
293     lorentz_en, lorentz_esp = "", ""
294     for char in lorentz:
295         for i in range(len(dict_en)):
296             if char == dict_en[i][0]:
297                 lorentz_en += dict_en[i][1]
298 
299     for char in lorentz:
300         for i in range(len(dict_esp)):
301             if char == dict_esp[i][0]:
302                 lorentz_esp += dict_esp[i][1]
303 
304     return en_encode, esp_encode, lorentz_en, lorentz_esp
305 
306 en_encode, esp_encode = encode()[0], encode()[1]
307 lorentz_en, lorentz_esp = encode()[2], encode()[3]
308 
309 print(f"Lorentz en la codificacion en ingles: {lorentz_en}, \
310 con longitud {len(lorentz_en)}")
311 print(f"Lorentz en la codificacion en espanol: {lorentz_esp}, \
312 con longitud {len(lorentz_esp)}")
313 
314 def binary_encode():
315     """
316     binary_lorentz : str; codificacion binaria de Lorentz
317     """
318     binary_lorentz = ""
319     for letter in lorentz:
320         b = bin(ord(letter))[2:]
321         binary_lorentz += b
322 
323     return binary_lorentz
324 
```

```
binary_lorentz = binary_encode()

print(f"\nLorentz en la codificacion binaria: \n{binary_lorentz},\
con longitud {len(binary_lorentz)}\
\nLa codificacion binaria es {len(binary_lorentz)/len(lorentz_en)} \
mas larga respecto a la codificacion inglesa\
\nLa codificacion binaria es {len(binary_lorentz)/len(lorentz_esp)} \
mas larga respecto a la codificacion espanola")


print("\n---------------\nTERCER APARTADO \n---------------")

# Listas de codigo Huffman de cada caracter
huffman_en, huffman_esp = [],[]
for duple in dict_en:
    huffman_en.append(duple[1])

for duple in dict_esp:
    huffman_esp.append(duple[1])

def decode(word, lenguage):
    """
    word : str; palabra en codigo Huffman a decodificar
    lenguage : str; {eng, esp}, lenguaje seleccionado para decodificar
    decode_en : str; decodificado de word en ingles
    decode_esp : str; decodificado de word en espanol
    """

    if lenguage != "eng" and lenguage != "esp":
        print("El lenguaje debe ser eng o esp")

    else:

        decode_en, decode_esp = "", ""

        if lenguage == "eng":

            i = 0
            while len(word) >= 3:

                length = len(huffman_en[i])
                if huffman_en[i] in word[:length]:
                    decode_en += dict_en[i][0]
                    word = word[length:]
                    i = 0
                i += 1
```

```
371
372             return decode_en
373
374         if lenguage == "esp":
375
376             i = 0
377             while len(word) >= 3:
378
379                 length = len(huffman_esp[i])
380                 if huffman_esp[i] in word[:length]:
381                     decode_esp += dict_esp[i][0]
382                     word = word[length:]
383                     i = 0
384                 i += 1
385
386             return decode_esp
387
388 print("Tomando las codificaciones de Lorentz del apartado anterior:\n")
389 print("Decodificado de 1001110001100100000101111010111110100 en ingles:",
390       decode("1001110001100100000101111010111110100", "eng"))
391 print("Decodificado de 1101111110110110010101011101111011101 en espanol:",
392       decode("1101111110110110010101011101111011101", "esp"))
```