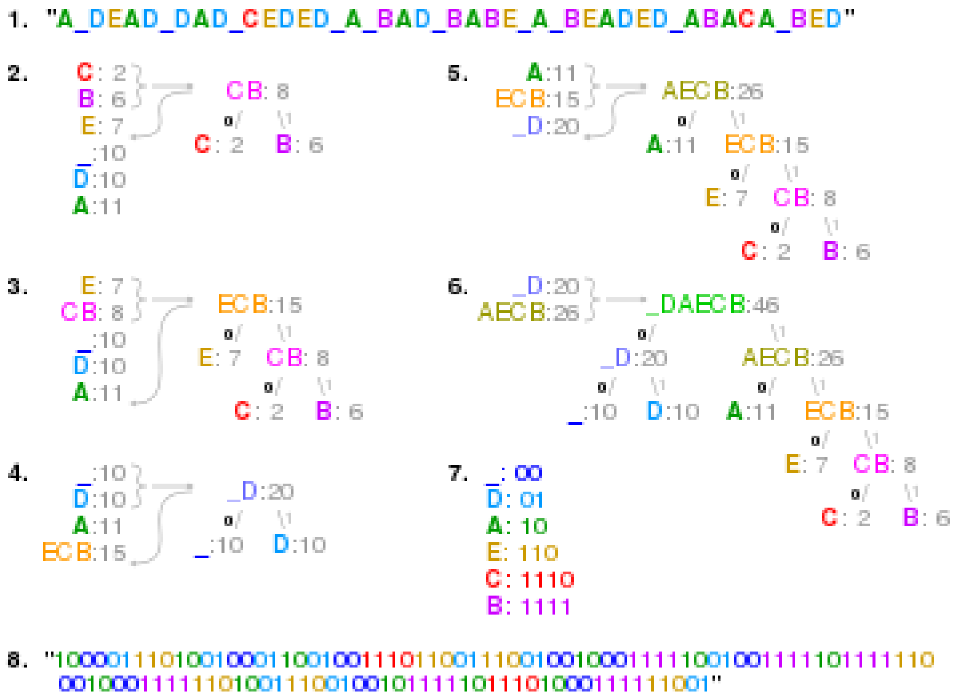
GEOMETRÍA COMPUTACIONAL

CÓDIGO HUFFMAN



Lucas de Torre

Índice

1. Introducción

Dadas las variables aleatorias $S_{English}$ = {a,b,c,...,!,?,0,...,9} y $S_{Español}$ = {a,b,c,...,ñ,...,!,?,0,...9} y su distribución de probabilidad (tomada de los archivos "auxiliar_enpract2.txt" y "auxiliar_es_pract2.txt", respectivamente), buscamos hallar el código Huffman binario de ambas variables, sus longitudes medias y comprobar que verifican el Primer Teorema de Shannon. Además, usaremos estos códigos binarios para codificar y decodificar palabras. Adicionalmente, calcularemos el índice de Gini y la diversidad ^2D de Hill.

2. Material

Partimos de un código que, para cada variable aleatoria, guarda en un array el árbol de Huffman dado por la distribución de probabilidad de dicha variable. A partir de dichos árboles, creamos un diccionario por cada una de las dos variables que, dado un elemento de nuestra variable aleatoria, devuelva la codificación Huffman binaria de dicho elemento. Estos diccionarios los creamos recorriendo los árboles desde la raíz hasta las hojas, hallando así el código Huffman binario de cada elemento.

Después, calculamos la longitud media de nuestras variables $L(S)$. Para ello, simplemente multiplicamos la longitud de cada elemento por su probabilidad y sumamos los resultados.

Ahora, calculamos la entropía de nuestros sistemas: $H(S) = -\sum_{j=1}^{N} P_j log_2 P_j$, donde N es el número de elementos del sistema y P_j es la probabilidad asociada al elemento j-ésimo. Podemos verificar que se cumple el Primer Teorema de Shannon, según el cual, $H(S) \leq L(S) < H(S)+1$.

Procedemos a codificar en ambas lenguas la palabra "fractal". Únicamente necesitamos hacer uso del diccionario anterior, que nos da, para cada letra de la palabra, su codificación. Una vez calculada la palabra, comparamos la eficiencia de esta codificación en comparación con la codificación binaria usual. Esto lo hacemos comparando el número de bits necesarios en cada caso.

A continuación decodificamos la cadena "0000111011111111111010". Comenzamos creando un diccionario que asocie a cada cadena de bits el carácter correspondiente (es el diccionario que teníamos pero intercambiando las claves por los valores). Después recorremos los bits a decodificar hasta encontrar una secuencia que coincida con una entrada de nuestro diccionario: por ejemplo, en este caso cogemos el primer 0 y vemos si coincide con una entrada de nuestro dicionario; si no, cogemos el siguiente elemento (0) y comprobamos si 00 tiene entrada en nuestro diccionario, y proseguimos hasta que así sea. Una vez encontrado, este será un carácter de la palabra decodificada, por lo que repetimos el proceso con el resto de la cadena hasta que esta sea vacía o ya no queden más elementos por mirar en la misma y tendremos la palabra buscada.

Proseguimos calculando el índice de Gini de nuestras variables. Utilizamos la fórmula $GI = 1-\sum_{j=2}^{N}(y_j + y_{j-1})(x_j + xj - 1)$, siendo y_j la frecuencia acumulada de aparición de todas las letras anteriores y la j-ésima, estando las letras ordenadas de menor a mayor probabilidad de aparición. Por último, calculamos la diversidad ^2D de Hill mediante la fórmula $^2D = \sum_{j=1}^{N} P_j^2$.

3. Resultados

Una vez hallado el diccionario, obtenemos fácilmente el código binario de cada letra. Por ejemplo, para $S_{English}$, tenemos que $a = 1011$ o que $u = 111111$. Las longitudes medias son:

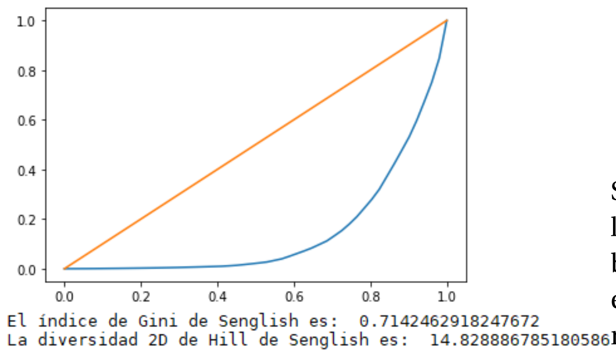
- $L(S_{English}) = 4{,}342324939634357$ (que verifica el Primer Teorema de Shannon porque la entropía es $H(S_{English}) = 4{,}314512109681417 <= 4{,}342324939634357 < H(S_{English})+1$
- $L(S_{Español}) = 4{,}214307970337135$ (que verifica el Primer Teorema de Shannon porque la entropía es $H(S_{Español}) = 4{,}183534871433366 <= 4{,}214307970337135 < H(S_{Español})+1$.

Al codificar la palabra "fractal" con el diccionario creado a partit del árbol de Huffman obetenemos el código:

- $S_{English}$: 0110001001011100011110101111110, que tiene 31 bits.
- $S_{Español}$: 011010001100011101111100000010, que tiene 30 bits.

En ambos casos vemos que este código es más eficiente que el binario usual, en el cual esta palabra necesita 42 bits.

En el cálculo del índice de Gini y a la diversidad ${}^2D$ de Hill para las dos variables se obtiene:



- $S_{English}$: El índice de Gini es $GI = 0{,}7142462918247$ y la diversidad ${}^2D$ de Hill es ${}^2D = 14{,}828886785180$.
- $S_{Español}$:El índice de Gini es $GI = 0{,}7332199409333$ y la diversidad ${}^2D$ de Hill es ${}^2D = 13{,}620973624876$.

Se observa que las variables tienen una desigualdad en la distribución similar (según el índice de Gini) y la probabilidad de que dos elementos tomados aleatoriamente en una de las variables tienen una probabilidad de pertenecer al mismo tipo muy parecida a la que tendrían dos elementos tomados al azar de la otra variable (según la diversidad ${}^2D$ de Hill).

4. Conclusiones

Resulta muy interesante comprobar como las longitudes medias de las codificaciones Huffman de ambas variables son menores que las del binario usual (aproximadamente 4,34 vs. 6). Esto se debe a que en binario usual no tenemos en cuenta la probabilidad de aparición de cada elemento. Una palabra como “fractal” necesita 31 bits con la codificación Huffman y 42 bits con la binaria usual, es decir, el binario usual necesita 1,35 veces más bits, muy parecido a lo que obtenemos comparando sus longitudes medias (1,38).

También llama la atención lo similares que son en índice de Gini y la diversidad ${}^2D$ de Hill para los dos alfabetos, ya que aunque ambos tengan un número de elementos similar, se da también que la distribución de probabilidad es similar en ambos.

5. Anexo A: Código

```python
"""
Pr ctica 2
"""

#import os
import numpy as np
import pandas as pd
import math
from itertools import accumulate as acc
import matplotlib.pyplot as plt
from collections import Counter

#### Carpeta donde se encuentran los archivos ####
#ubica = "C:/Users/Python"

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

with open('auxiliar_en_pract2.txt', 'r') as file:
      en = file.read()

with open('auxiliar_es_pract2.txt', 'r') as file:
      es = file.read()

#### Pasamos todas las letras a min sculas
en = en.lower()
es = es.lower()

#### Contamos cuantas letras hay en cada texto
from collections import Counter
tab_en = Counter(en)
tab_es = Counter(es)

##### Transformamos en formato array de los car cteres (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 funci n que haga ex ctamente 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][0] ## Esto proporciona un '0'
list(tree[0].items())[1][0] ## Esto proporciona un '1'

"""
A partir de aqu , c odigo escrito por:
- Jorge Sainero Valle
- Lucas de Torre Barrio
"""

#Extrae el c digo binario de Huffman de cada car cter y lo devuelve en un diccionario {car cter : c digo}
def extract_code():
    global tree
    d = dict()
    #Empezamos por el final porque es donde est  la ra z
    for i in range(tree.size-1,-1,-1):
        elem=tree[i]
        h1=list(elem.keys())[0]
        h2=list(elem.keys())[1]

        for c in h1:
            if c in d:
```

```
                d[c]+='0'
            else:
                d[c]='0'

        for c in h2:
            if c in d:
                d[c]+='1'
            else:
                d[c]='1'
    return d

#Calcula la longitud del c digo binario de Huffman
def longitudMedia(d):
    ac=0
    for i,k in distr.iterrows():
        ac+=len(d[k['states']])*k['probab']
    return ac

#Calcula la entrop a de una variable aleatoria con la distribuci n de probabilidades distr['probab']
def entropia():
    h=0
    for p in distr['probab']:
        h-=p*math.log(p,2)
    return h



def apartado1():
    print("APARTADO UNO:\n")
    global tree
    global distr
    distr = distr_en
    tree = huffman_tree(distr)
    d=extract_code()
    print("Una peque a muestra del diccionario con la codificaci n de Senglish:",dict(Counter(d).most_common(7)),'\n')
    print ("La longitud media de Senglish es: "+str(longitudMedia(d)))
    h_en=entropia()
    print("La entrop a de Senglish es: ",h_en)
    print("Vemos que se cumple el Teorema de Shannon ya que",h_en,"<=",str(longitudMedia(d)),"<",h_en+1,"\n")


    distr = distr_es
    tree = huffman_tree(distr)
    d=extract_code()
    print("Una peque a muestra del diccionario con la codificaci n de Sspanish:",dict(Counter(d).most_common(7)),'\n')
    print ("La longitud media de Sspanish es: "+str(longitudMedia(d)))
    h_es=entropia()
    print("La entrop a de Sspanish es: ",h_es)
    print("Vemos que se cumple el Teorema de Shannon ya que",h_es,"<=",str(longitudMedia(d)),"<",h_es+1,"\n\n")

#Codifica la palabra pal seg n el diccionario d
def codificar(pal, d):
    binario=""
    for l in pal:
        binario += d[l]
    return binario

```

```
def apartado2():
    print("APARTADO DOS:\n")
    global tree
    global distr
    distr = distr_en
    tree = huffman_tree(distr)
    d_en=extract_code()
    palabra = "fractal"
    pal_bin =codificar(palabra,d_en)
    print("El codigo binario de la palabra fractal en lengua inglesa es:",pal_bin,"y su longitud es:",len(pal_bin))
    #La longitud en binario usual es ceil(log2(cantidadDeCracteres)) por cada letra
    print("En binario usual ser a de longitud:", len(palabra)*math.ceil(math.log(len(d_en),2)),"\n\n")
    distr = distr_es
    tree = huffman_tree(distr)
    d_es=extract_code()
    pal_bin =codificar(palabra,d_es)
    print("El codigo binario de la palabra fractal en lengua espa ola es:",pal_bin," y su longitud es:",len(pal_bin))
    #La longitud en binario usual es ceil(log2(cantidadDeCracteres)) por cada letra
    print("En binario usual ser a de longitud:", len(palabra)*math.ceil(math.log(len(d_es),2)),"\n\n")

#Decodifica la palabra pal seg n el diccionario d
def decodificar(pal,d):
    aux=''
    decode=''
    for i in pal:
        aux+=i
        if aux in d:
            decode+=d[aux]
            aux=''
    return decode

def apartado3():
    print("APARTADO TRES:\n")
    global tree
    global distr
    distr = distr_en
    tree = huffman_tree(distr)
    d_en=extract_code()
    #Invertimos el diccionario para poder decodificar
    d_en_inv=dict(zip(list(d_en.values()),list(d_en.keys())))
    pal_bin='10101000011110111111100'
    palabra=decodificar(pal_bin,d_en_inv)
    print("La palabra cuyo c digo binario es:",pal_bin,"en ingl s es:",palabra,end='\n\n\n')

#Calcula el  ndice  de Gini de una variable aleatoria con la distribuci n de probabilidades distr['probab']
def gini():
    aux=0
    #ya est  ordenado as  que no hace falta ordenarlo
    accu=list(acc(distr['probab']))
    plt.plot(np.linspace(0,1,len(accu)),accu)
    plt.plot(np.linspace(0,1,len(accu)),np.linspace(0,1,len(accu)))
    plt.show()
    for i in range(1,len(accu)):
        aux+=(accu[i]+accu[i-1])/len(accu)
    return 1-aux
```

```
#Calcula la diversidad 2D de Hill de una variable aleatoria con la distribuci n de probabilidades distr['probab']
def diver2hill():
    aux=0
    for p in distr['probab']:
        aux+=p*p
    return 1/aux


def apartado4():
    print("APARTADO CUATRO:\n")
    global distr
    distr = distr_en
    print("El  ndice  de Gini de Senglish es: ",gini())
    print("La diversidad 2D de Hill de Senglish es: ",diver2hill())
    distr = distr_es
    print("El  ndice  de Gini de Sspanish es: ",gini())
    print("La diversidad 2D de Hill de Sspanish es: ",diver2hill())

apartado1()
apartado2()
apartado3()
apartado4()
```