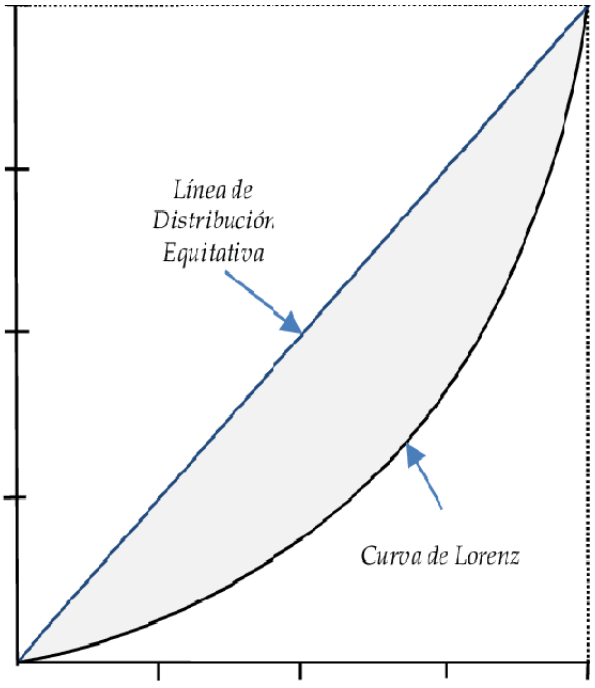
GEOMETRÍA COMPUTACIONAL

ÍNDICE DE GINI Y DIVERSIDAD ${}^2D$ DE HILL



LUCAS DE TORRE

Índice

1. Introducción

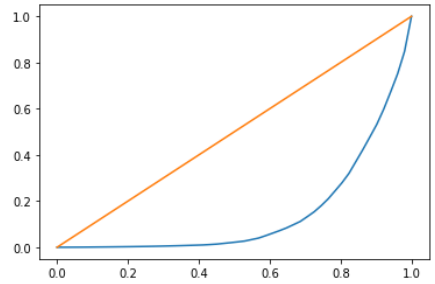
Dadas las variables aleatorias $S_{English}$ = {a,b,c,...,!,?,0,...,9} y $S_{Español}$ = {a,b,c,...,ñ,...,!,?,0,...9} y su distribución de probabilidad (tomada de los archivos "auxiliar_enpract2.txt" y "auxiliar_es_pract2.txt", respectivamente), buscamos hallar el índice de Gini y la diversidad ${}^2D$ de Hill de ambas variables.

2. Material

Calculamos 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 ${}^2D = \sum_{j=1}^{N} P_j^2$, en la cual P_j es la probabilidad de aparición de la letra j-ésima.

3. Resultados

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




```
El índice de Gini de Senglish es:  0.7142462918247672
La diversidad 2D de Hill de Senglish es:  14.828886785180586
```

- $S_{English}$: 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

LLama la atención lo similares que son el 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

```
"""
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:
```

```
122                 d[c]+='0'
123             else:
124                 d[c]='0'
125 
126         for c in h2:
127             if c in d:
128                 d[c]+='1'
129             else:
130                 d[c]='1'
131     return d
132 
133 #Calcula la longitud del c digo binario de Huffman
134 def longitudMedia(d):
135     ac=0
136     for i,k in distr.iterrows():
137         ac+=len(d[k['states']])*k['probab']
138     return ac
139 
140 #Calcula la entrop a de una variable aleatoria con la distribuci n de
    probabilidades distr['probab']
141 def entropia():
142     h=0
143     for p in distr['probab']:
144         h-=p*math.log(p,2)
145     return h
146 
147 
148 
149 def apartado1():
150     print("APARTADO UNO:\n")
151     global tree
152     global distr
153     distr = distr_en
154     tree = huffman_tree(distr)
155     d=extract_code()
156     print("Una peque a muestra del diccionario con la codificaci n de Senglish:",
    dict(Counter(d).most_common(7)),'\n')
157     print ("La longitud media de Senglish es: "+str(longitudMedia(d)))
158     h_en=entropia()
159     print("La entrop a de Senglish es: ",h_en)
160     print("Vemos que se cumple el Teorema de Shannon ya que",h_en,"<=",str(
    longitudMedia(d)),"<",h_en+1,"\n")
161 
162 
163     distr = distr_es
164     tree = huffman_tree(distr)
165     d=extract_code()
166     print("Una peque a muestra del diccionario con la codificaci n de Sspanish:",
    dict(Counter(d).most_common(7)),'\n')
167     print ("La longitud media de Sspanish es: "+str(longitudMedia(d)))
168     h_es=entropia()
169     print("La entrop a de Sspanish es: ",h_es)
170     print("Vemos que se cumple el Teorema de Shannon ya que",h_es,"<=",str(
    longitudMedia(d)),"<",h_es+1,"\n\n")
171 
172 #Codifica la palabra pal seg n el diccionario d
173 def codificar(pal, d):
174     binario=""
175     for l in pal:
176         binario += d[l]
177     return binario
178 
```

```
179 def apartado2():
180     print("APARTADO DOS:\n")
181     global tree
182     global distr
183     distr = distr_en
184     tree = huffman_tree(distr)
185     d_en=extract_code()
186     palabra = "fractal"
187     pal_bin =codificar(palabra,d_en)
188     print("El codigo binario de la palabra fractal en lengua inglesa es:",pal_bin,"y
        su longitud es:",len(pal_bin))
189     #La longitud en binario usual es ceil(log2(cantidadDeCracteres)) por cada letra
190     print("En binario usual ser a de longitud:", len(palabra)*math.ceil(math.log(len
        (d_en),2)),"\n\n")
191     distr = distr_es
192     tree = huffman_tree(distr)
193     d_es=extract_code()
194     pal_bin =codificar(palabra,d_es)
195     print("El codigo binario de la palabra fractal en lengua espa ola es:",pal_bin,"
        y su longitud es:",len(pal_bin))
196     #La longitud en binario usual es ceil(log2(cantidadDeCracteres)) por cada letra
197     print("En binario usual ser a de longitud:", len(palabra)*math.ceil(math.log(len
        (d_es),2)),"\n\n")
198
199 #Decodifica la palabra pal seg n el diccionario d
200 def decodificar(pal,d):
201     aux=''
202     decode=''
203     for i in pal:
204         aux+=i
205         if aux in d:
206             decode+=d[aux]
207             aux=''
208     return decode
209
210 def apartado3():
211     print("APARTADO TRES:\n")
212     global tree
213     global distr
214     distr = distr_en
215     tree = huffman_tree(distr)
216     d_en=extract_code()
217     #Invertimos el diccionario para poder decodificar
218     d_en_inv=dict(zip(list(d_en.values()),list(d_en.keys())))
219     pal_bin='10101000011111011111100'
220     palabra=decodificar(pal_bin,d_en_inv)
221     print("La palabra cuyo c digo binario es:",pal_bin,"en ingl s es:",palabra,end=
        '\n\n\n')
222
223 #Calcula el  ndice  de Gini de una variable aleatoria con la distribuci n de
        probabilidades distr['probab']
224 def gini():
225     aux=0
226     #ya est  ordenado as  que no hace falta ordenarlo
227     accu=list(acc(distr['probab']))
228     plt.plot(np.linspace(0,1,len(accu)),accu)
229     plt.plot(np.linspace(0,1,len(accu)),np.linspace(0,1,len(accu)))
230     plt.show()
231     for i in range(1,len(accu)):
232         aux+=(accu[i]+accu[i-1])/len(accu)
233     return 1-aux
234
```

```
235 #Calcula la diversidad 2D de Hill de una variable aleatoria con la distribuci n de
    probabilidades distr['probab']
236 def diver2hill():
237     aux=0
238     for p in distr['probab']:
239         aux+=p*p
240     return 1/aux
241
242
243 def apartado4():
244     print("APARTADO CUATRO:\n")
245     global distr
246     distr = distr_en
247     print("El  ndice  de Gini de Senglish es: ",gini())
248     print("La diversidad 2D de Hill de Senglish es: ",diver2hill())
249     distr = distr_es
250     print("El  ndice  de Gini de Sspanish es: ",gini())
251     print("La diversidad 2D de Hill de Sspanish es: ",diver2hill())
252
253 apartado1()
254 apartado2()
255 apartado3()
256 apartado4()
```