Práctica 2. Extra: Índice de Gini y de Hill

Guillermo Martín Sánchez

27/02/20

1 Introducción

Como parte extra de la Práctica 2 se nos sugiere calcular el índice de Gini y la diversidad ${}^2D$ de Hill de la variable aleatoria $S_{English}$. Nosotros hemos decidido además pintar la curva de Lorentz.

2 Material usado

2.1 Ejercicio 4

Considerando la variable aleatoria $S_{English}$, calcula el índice de Gini y la diversidad ${}^2D$ de Hill.

Primero hemos pintado la curva de Lorentz. Para ello, dada la tabla de frecuencias (que nos asocia cada uno de los N estados a_i a su frecuencia p_i) de $S_{English}$ ordenada por frecuencia, hemos calculado la acumulación empírica de las frecuencias Y para cada uno de los valores de la frecuencia acumulada X. Es decir, hemos calculado $x_i = i/N \quad \forall i$ y $y_i = \sum_{k=1}^{i} p_i$. Hemos impreso entonces la curva formada por los puntos $(x_i, y_i) \quad \forall i$ y comparado con la curva ideal que representa la mínima desigualdad: (x_i, y_i). El área bajo la primera curva y el área entre la curva ideal y la primera curva son denominadas A y S respectivamente, y una forma de calcular el índice de Gini es mediante la fórmula $GI = \frac{S}{S+A}$.

Para calcular el índice de Gini, sin embargo, hemos usado la siguiente fórmula:

$$GI = 1 - \sum_{j=2}^{N} (y_j + y_{j-1})(x_j - x_{j-1}) \tag{1}$$

donde, por construcción, $x_j - x_{j-1} = \frac{1}{N} \quad \forall j$ y, por tanto, se puede sacar factor común.

Finalmente para el cálculo de la diversidad ${}^2D$ de Hill hemos usado la fórmula general:

$$ {}^{q}D = \lim_{x \to q} (\sum_{j=1}^{N} p_j^x)^{\frac{1}{1-x}} \tag{2} $$

que para el caso $q = 2$ se vuelve:

$$ {}^{2}D = (\sum_{j=1}^{N} p_j^2)^{-1} \tag{3} $$

3 Resultados y discusión

3.1 Ejercicio 4

Hemos obtenido un índice de Gini de $GI \approx 0.7142$. Al estar este valor cerca de 1 nos indica que hay una desigualdad significativa en la distribución de la variable $S_{English}$. Viendo su curva de Lorentz en la Figura 1, podemos ver que efectivamente el área de S es bastante grande. En especifico podemos percibir como los estados más frecuentes lo son mucho más que los menos frecuentes.

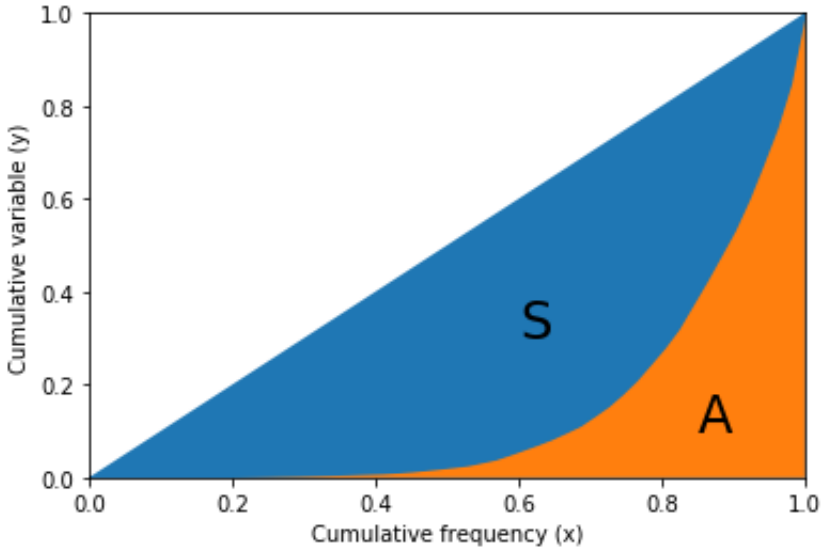

Figure 1: Curva de Lorentz para la variable aleatoria $S_{English}$

Respecto la diversidad ${}^{2}D$ de Hill hemos obtenido un valor de ${}^{2}D \approx 14.83$, que representa la verdadera diversidad de la variable. De hecho si la variable fuera equidistribuida, sería igual a la riqueza R, es decir, ${}^{2}D = R = |S| = 52$, mucho mayor. Visto de otra forma, el índice de Simpson $\lambda = \frac{1}{{}^{2}D}$ que representa la probabilidad de que dos entidades tomadas al azar del conjunto de datos

representen el mismo tipo sería $\lambda = \frac{1}{R} = \frac{1}{52} \approx 0.0192$ en el caso equidistribuido. Sin embargo, para $S_{English}$ es $\lambda \approx \frac{1}{14.83} \approx 0.0674$, es decir, más de tres veces mayor.

4 Conclusión

La curva de Lorentz, el índice de Gini, la diversidad ${}^2D$ de Hill y el índice de Simpson son cuatro métodos que nos permiten estudiar la desigualdad y diversidad de un sistema. En este caso hemos estudiado el sistema $S_{English}$ y hemos visto que la frecuencia (una aproximación de la variable aleatoria poblacional) está muy desigualmente distribuida entre los caracteres. Tanto es así, que la verdadera diversidad es menor que el número total de caracteres y la probabilidad de obtener dos caracteres iguales es mayor que el que nos encontraríamos en un caso equidistribuido.

5 Código

```
"""
Coursework 2: Huffman Codification
"""

import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from collections import defaultdict

ubica = "C:/Users/guill/Documents/Carrera/GEOComp"

os.getcwd()
os.chdir(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()


en = en.lower()
es = es.lower()
```

```
# Count frequency in each text
from collections import Counter
tab_en = Counter(en)
tab_es = Counter(es)


# Construct DataFrame and 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))

# Given a frequency table 'distr', computes a branch of the Huffman tree
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)

# Given a frequency table 'distr', computes the Huffman tree
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)
```

```
# Given a Huffman 'tree', computes it's Huffman table
def huffman_code(tree):
    dict = defaultdict(str)
    for i in range(len(tree) - 1, -1, -1):
        word0 = list(tree[i].keys())[0];
        for c in word0:
            dict[c]+='0'
        word1 = list(tree[i].keys())[1];
        for c in word1:
            dict[c]+='1'
    return(dict)

# Given a frequency table 'distr' and a Huffman table 'code', computes the
# average length L = 1/W * sum(wi * |ci|)
def average_length(distr, code):
    length = 0
    for i,row in distr.iterrows():
        length+=row['probab']*len(code[row['states']])
    return length

# Given a word 'word', and a Huffman table 'code' with 'word' belonging to the
# system 'code' codifies, computes the minimum number of bits necessary to
# represent the word, supossing we use the same number of bits to encode each
# possible state
def binary_length(word,code):
    return int(np.ceil(np.log2(len(code)))*len(word))

# Given a word 'word', and a Huffman table 'code' with 'word' belonging to the
# system 'code' codifies, computes the Huffman codification of that word
def codify(word,code):
    cod = ''
    for a in word:
        cod += code[a]
    return cod

# Given an injective dictionary 'code', computes its inverse dictionary
def inverse_dict(code):
    invcode = {}
    for x in code:
        invcode[code[x]] = x
    return invcode

# Given a word 'word' in binary, and the inverse of a Huffman table 'invcode',
# computes the corresponding word in the system that the Huffman table codifies
def decodify(word,invcode):
    acum = ''
```

```
    decod = ''
    for b in word:
        acum += b
        if(acum in invcode):
            decod += invcode[acum]
            acum = ''
    return decod

# Given a frequency table 'distr', computes its cumulative variable
def cumulative_variable(distr):
    acum = 0
    y = []
    for i,row in distr.iterrows():
        acum +=row['probab']
        y.append(acum)
    return y

# Given a cumulative variable 'y', computes its Gini Index
def gini_index(y):
    acum = 0
    for j in range(len(y)-1):
        acum += (y[j] + y[j+1])
    return 1 - (1/len(y))*acum

# Given a frequency table 'distr', computes its Hill Index with q = 2
def D2_Hill(distr):
    acum = 0
    for i,row in distr.iterrows():
        acum +=row['probab']**2
    return 1/acum

# Given a frequency table 'distr', computes its Entropy
def entropy(distr):
    entr = 0
    for i,row in distr.iterrows():
        entr+=row['probab']*np.log2(row['probab'])
    return -entr

#%%
"""
Exercise 1: Huffman Code
"""

# English
# Construct the Huffman tree
tree_en = huffman_tree(distr_en)
```

```
# Construct the Huffman table
code_en = huffman_code(tree_en)

# Compute the average length of the Huffman codification
l_en = average_length(distr_en,code_en)

# Compute the entropy of English
h_en = entropy(distr_en)

print("The average length of the random variable SEnglish is {:.4g}".format(l_en))

# Check Shannon's First Theorem: H(C) <= L(C) < H(C) + 1
print("Shannon's First Theorem: {:.4g} <= {:.4g} < {:.4g}".format(h_en, l_en, h_en+1))


# Spanish
tree_es = huffman_tree(distr_es)
code_es = huffman_code(tree_es)
l_es = average_length(distr_es,code_es)
h_es = entropy(distr_es)
print("The average length of the random variable SSpanish is {:.4g}".format(l_es))
print("Shannon's First Theorem: {:.4g} <= {:.4g} < {:.4g}".format(h_es, l_es, h_es+1))

#%%
"""
Exercise 2: Codification
"""
# English

# Codify with the Huffman codification
cod_en = codify('fractal',code_en)

# Estimate number of bits in usual binary codification
binlen_en = binary_length('fractal',code_en)

# Print in Huffman code
print("Codification Huffman: ", cod_en)

# Check the Huffman codification is shorter than usual binary
print("Length in Huffman codification:", len(cod_en),
      "< Length in Binary codification:", binlen_en)

# Spanish
cod_es = codify('fractal',code_es)
binlen_es = binary_length('fractal',code_es)
```

```
print("Codification Huffman: ", cod_es)
print("Length in Huffman codification:", len(cod_es),
      "< Length in Binary codification:", binlen_es)

#%%
"""
Exercise 3: Decodification
"""
# Construct inverse dictionary for decodification
invcode_en = inverse_dict(code_en)

# Decodify Huffman code
decod_en = decodify('1010100001111011111100', invcode_en)

# Print decodified word
print("The word 1010100001111011111100 =", decod_en)

#%%
"""
Exercise 4: Gini index and Hill index
"""
x = np.linspace(0,1,len(distr_en))

# Compute cumulative variable
y = cumulative_variable(distr_en)

# Print Lorentz curve
plt.text(0.6, 0.3, 'S', fontsize=24)
plt.text(0.85, 0.1, 'A', fontsize=24)
plt.plot(x,y)
ax = plt.gca()
ax.set_xlim([0,1])
ax.set_ylim([0,1])
plt.fill_between(x,0,x)
plt.fill_between(x,0,y)
plt.xlabel('Cumulative frequency (x)')
plt.ylabel('Cumulative variable (y)')

# Compute and print Gini Index
g = gini_index(y)
print("The Gini Index of English is G = {:.4g}".format(g))

# Compute and print Hill(q=2) Index
d = D2_Hill(distr_en)
print("The Hill Index for q = 2 of English is 2D = {:.4g}".format(d))
```