Team: Bakhtiyar Rakhimzhanov, Nazira Tukeyeva

Group: BD-1903

Github Link to the Repository: <https://github.com/nazirait/Information-Theory-Project>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Github Link of Bakhtiyar: <https://github.com/Godadoreu>

Github Link of Nazira: <https://github.com/nazirait>

**Assignment 3 Report**

|  |  |
| --- | --- |
| Task Part | Responsible team member |
| Part 1 – Sorting, dividing the list into parts, splitting text | Nazira |
| Part 2 - Encoding, calculating the compression ratio | Bakhtiyar |
| Assignment Report | Both |

**General Description:**

**Shannon-Fano Algorithm Implementation**

Step 1: Dividing the list and returning each part.

*Source code:*

def divide\_parts(lists):

'''The given function divides the list into two parts with the equal

sum of occurences on the right and left parts'''

sums\_left = 0

left = []

sums\_right = 0

right = []

for x,y in lists:

# sum the probabilities of the elements

sums\_left+=y

# store to the list 'left'

left.append(x)

if sums\_left>=0.5:

break # stop when the sum is 0.5 or more

for x,y in lists:

if x not in left:

# sum the probabilities of the elements (excluding those in the 'left' list)

sums\_right+=y

# store to the list 'left'

right.append(x)

# return the characters and sum of the probabilies on both sides (right and left)

return sums\_left,left,sums\_right,right

balance = divide\_parts(result)

balance

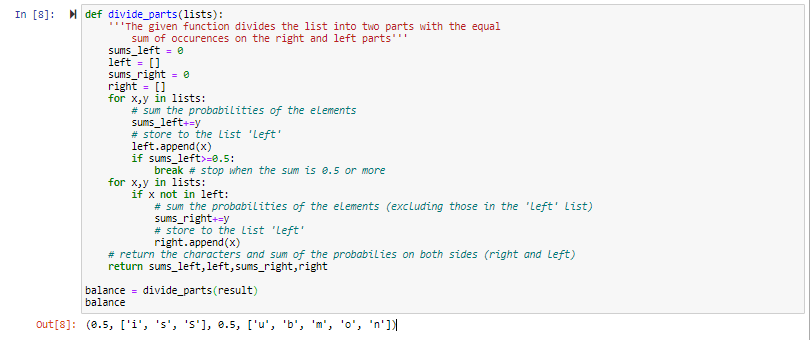


Fig. 1

Step 2: Labeling the elements on the left part as 0 and on the right part as 1.

*Source code:*

def encoding(lists\_desc,list\_balance):

'''The given function labels the left part

as 0 and the right part as 1'''

prob\_left,word\_left,prob\_right,word\_right = list\_balance

symbols\_left = '0'

symbols\_right = '1'

symbols\_for\_lleft = []

mean\_left = (prob\_left/len(word\_left))

print('Left avg: ', mean\_left)

mean\_right = (prob\_right/len(word\_right))

print('Right avg: ', mean\_right)

sums\_left= 0

sums\_right = 0

symbols\_for\_lright = []

count = 0

symbols\_for\_rleft = []

symbols\_for\_rright = []

# left part

for x in word\_left:

for key,value in lists\_desc:

if key in x:

sums\_left+=value

# if the probability value is greater or equal than the left average (0.16)

if sums\_left>=mean\_left:

symbols\_left+='0' # assign those elements as 0 --> 00

# store the key and the encoded part into LeftLeft list

symbols\_for\_lleft.append((key,symbols\_left))

break

else:

# assign those elements as 1 --> 01

symbols\_left+='1'

# store the key and the encoded part into leftRight list

symbols\_for\_lright.append((key,symbols\_left))

# ex.: i,s goes to the leftleft - 00, S goes to the leftright - 01

sums\_left = 0

symbols\_left='0'

# right part

for x in word\_right:

for key,value in lists\_desc:

if key in x:

symbols\_right = '1'

sums\_right+=value

# if the probability value is greater or equal than the right average (0.1)

if sums\_right>=mean\_right:

symbols\_right+='0' # assign those elements as 0 --> 10

# store the key and the encoded part into rightLeft list

symbols\_for\_rleft.append((key,symbols\_right))

break

else:

count+=1

# label those elements as '1' --> 11

symbols\_right+='1'

# store the key and the encoded part into rightRight list

symbols\_for\_rright.append((key,symbols\_right))

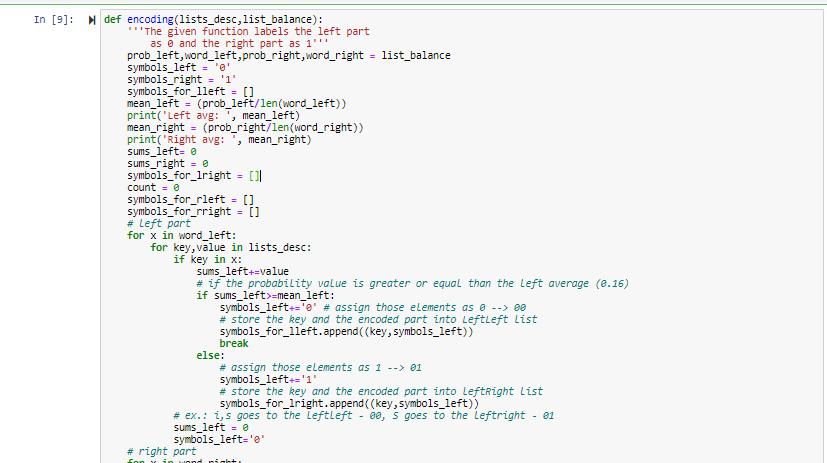
# ex.: all the elements in the right has the prob values of 0.1, so all are labeled as '10'

sums\_right=0

return symbols\_for\_lleft,symbols\_for\_lright,symbols\_for\_rleft,symbols\_for\_rright

encoder = encoding(result,balance)

encoder



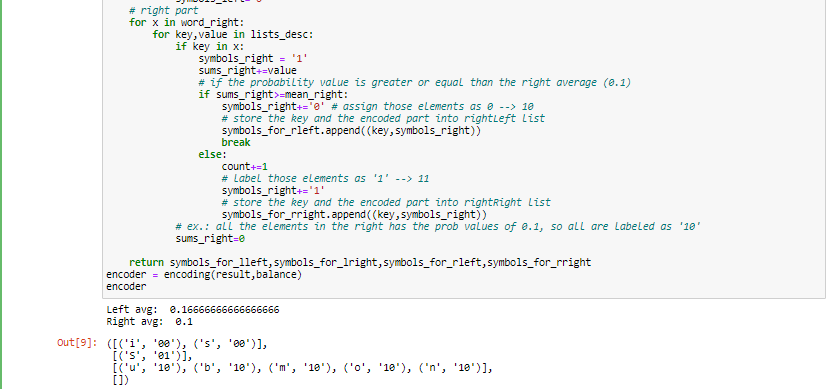


Fig. 2-2.1

Step 3: Displaying the elements along with their codewords sorted by the probability values in descending order.

*Source code:*

def adding(probabilities,code):

'''The given function returns the list with symbols and their

codewords sorted by their probabilities in descending order'''

result = []

symbols\_lleft,symbols\_lright,symbols\_rleft,symbols\_rright = code

for key,value in probabilities:

for key1,value1 in symbols\_lleft: # s, i - 00

if key in key1:

result.append((key,value,value1))

for key,value in probabilities:

for key1,value1 in symbols\_lright: # S - 01

if key in key1:

result.append((key,value,value1))

for key,value in probabilities:

for key1,value1 in symbols\_rleft: # u,b,m,o,n - 10

if key in key1:

result.append((key,value,value1))

for key,value in probabilities:

for key1,value1 in symbols\_rright:

if key in key1:

result.append((key,value,value1))

return result

third\_function = adding(result,encoder)

third\_function

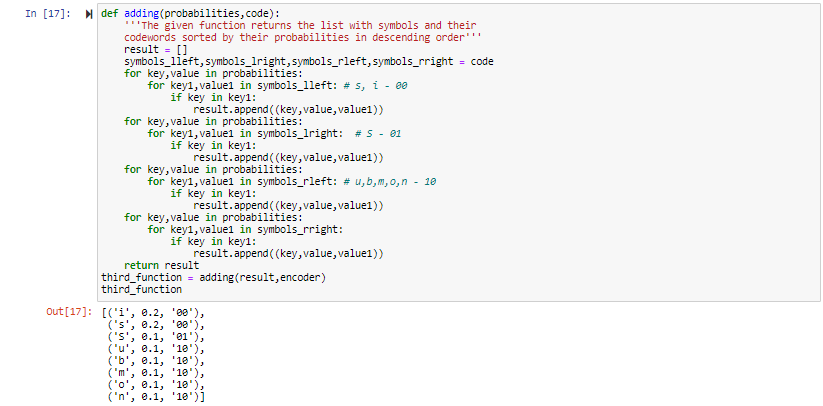


Fig. 3

Step 4: Splitting the text into separate characters.

*Source code:*

def split\_text(text):

'''The given function splits the text into separate characters'''

f = open(text)

# reading the file

new\_list = []

reading = f.readlines()

for x in reading:

list\_x = list(x)

for d in list\_x:

new\_list.append(d)

return new\_list

# Closing the file

f.close()

res = split\_text('vil.txt')

res

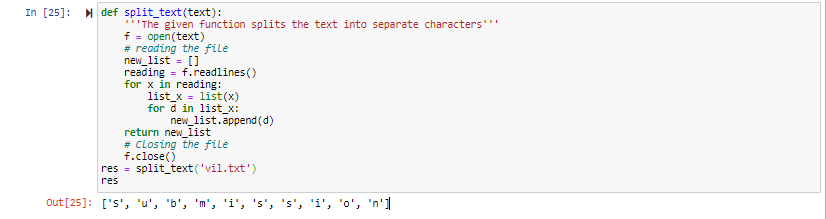


Fig. 4

Step 5: Outputting a sequence of binary digits and saving the result into a new txt file.

*Source code:*

def code\_replacement(symbols,lists):

'''The given function returns the encoded part (a sequence of binary digits)

and saves the result into new txt file'''

symbols\_lleft,symbols\_lright,symbols\_rleft,symbols\_rright = symbols

resultant = []

new\_string = []

for key,value in symbols\_lleft:

for i,x in enumerate(lists):

if key==x:

lists[i] = value

else:

continue

for key,value in symbols\_lright:

for i,x in enumerate(lists):

if key==x:

lists[i] = value

else:

continue

for key,value in symbols\_rleft:

for i,x in enumerate(lists):

if key==x:

lists[i] = value

else:

continue

for key,value in symbols\_rright:

for i,x in enumerate(lists):

if key==x:

lists[i] = value

else:

continue

encoded\_variant = ''.join(lists)

file = open('new\_file.txt','w')

file.write(encoded\_variant)

file.close()

return encoded\_variant

encoded = code\_replacement(encoder,res)

encoded



Fig. 5

Step 6: Calculating a data compression ratio including number of bits in original and compressed text, compression ratio and the average code length.

*Source code:*

def compress\_ratio(dicte,comp\_string,probs):

'''The given function calculates data compression ratio'''

original\_string = sum(dicte.values())\*7

comp\_string\_len = len(comp\_string)

sums = 0

for key,prob,code in probs:

sums+=prob\*len(code)

print('Number of bits in original text: ',original\_string)

print('Number of bits in compressed text: ', comp\_string\_len)

print('Compression ratio: ',original\_string/comp\_string\_len)

print('Average ratio: ', sums)

res = compress\_ratio(dicts, encoded,third\_function)

res

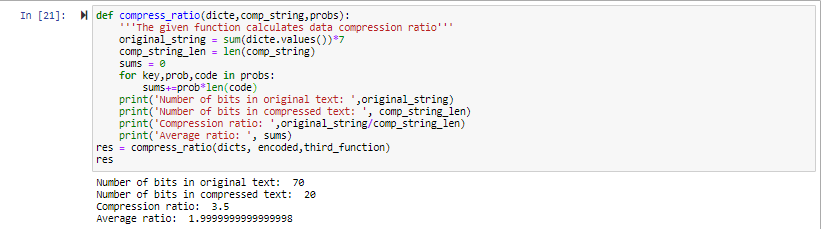


Fig. 6