Data_Compression_Project_Arithmetic_1

November 14, 2022

[20]:
```
import numpy as np
import matplotlib.pyplot as plt
import csv
from collections import Counter
```

[21]:
```
##############################################################################################
# Read Text file
##############################################################################################
f = open("Toy_Example_Arithmetic.txt", "r")
text = f.read()
print(text)
```

```
DACDDBCD
```

[22]:
```
##############################################################################################
# Compute the empirical distribution
##############################################################################################
def compute_distribution(text):
    """
    Inputs:
    - text: A string containing the text to be encoded.

    Returns:
    - symbols: a list of tuples of the form (char,prob), where char is a␣
 ↪character appears in the text
                and prob is the number of times this character appeared in text␣
 ↪divided by the length of text.
    """
    # ============================================================ #
    # YOUR CODE HERE:
    # ============================================================ #
    symbols = []
    counter = dict()
    for k in text:
        if k in counter:
            counter[k]+=1/len(text)
        else:
            counter[k]=1/len(text)
```

```
    symbols = []
    for j in counter.keys():
        symbols.append((j, counter[j]))
    symbols = sorted(symbols, key=lambda x:x[1])
    # ============================================================ #
    # END YOUR CODE HERE
    # ============================================================ #
    return symbols

symbols = compute_distribution(text)
size_symbols = len(symbols)
print(symbols)
```

```
[('A', 0.125), ('B', 0.125), ('C', 0.25), ('D', 0.5)]
```

0.1 Part 2: Arithmetic Codes

[23]:
```
##############################################################################################
# Compute the expected length of the Huffman code
##############################################################################################
def compute_CDF(symbols):
    """
    Inputs:
    - symbols: A list of tuples of the form (char,prob).
    Returns:
    - CDF_symbols: A list of tuples of the form (char,CDF)
    """
    CDF_symbols = []
    # ============================================================ #
    # YOUR CODE HERE:
    # ============================================================ #
    total=0
    for i in symbols:
        CDF_symbols.append((i[0], i[1]+total))
        total += i[1]
    # ============================================================ #
    # END YOUR CODE HERE
    # ============================================================ #
    return CDF_symbols

CDF_symbols = compute_CDF(symbols)
print(CDF_symbols)
```

```
[('A', 0.125), ('B', 0.25), ('C', 0.5), ('D', 1.0)]
```

[24]:
```
##############################################################################
# Decimal encoding
```

```
##############################################################################
def decimal_encoding(text,CDF_symbols) :
    """
    Inputs:
    - text: A string containing the text to be encoded.
    Returns:
    - lower: the lower value of the interval of the encoded text.
    - upper: the upper value of the interval of the encoded text.
    """
    lower = 0
    upper = 1
    # ============================================================= #
    # YOUR CODE HERE:
    # ============================================================= #
    d = dict(CDF_symbols)
    sy = [i[0] for i in CDF_symbols]
    for c in text:
        r = upper - lower
        upcdf = d[c]
        ind = sy.index(c)
        locdf = 0
        if (ind != 0):
            locdf = d[sy[ind-1]]
        lower += r* locdf
        upper -= r*(1-upcdf)



    # ============================================================= #
    # END YOUR CODE HERE
    # ============================================================= #
    return lower, upper

lower,upper = decimal_encoding(text,CDF_symbols)
print("Interval representing the text is:  ", lower, upper)
```

```
Interval representing the text is:   0.52801513671875 0.528076171875
```

[25]:
```
##############################################################################
# Binary encoding
##############################################################################
def Arithmetic_encoding(lower,upper):
    """
    Inputs:
    - lower: the lower value of the interval of the encoded text.
    - upper: the upper value of the interval of the encoded text.
    Returns:
```

```
    - txt_code: a string represents the code of the input text.
    """
    txt_code = ''
    # ================================================================ #
    # YOUR CODE HERE:
    # ================================================================ #
    l = np.ceil(np.log2(1/(upper -lower)))+1
    m = (upper + lower)/2
    m *= (2**l)
    m = int(m)
    return bin(m)[2:]
    # ================================================================ #
    # END YOUR CODE HERE
    # ================================================================ #
    return txt_code

txt_code = Arithmetic_encoding(lower,upper)
print("Encoded Text:  ", txt_code)
Expected_length_Arithematic = len(txt_code)/len(text)
print("Expected length of Arithematic code:   ", Expected_length_Arithematic)
```

```
Encoded Text:   100001110010111
Expected length of Arithematic code:    1.875
```

[26]:
```
##########################################################################
# Binary Decoding
##########################################################################
def decimal_decoding(txt_code):
    """
    Inputs:
    - txt_code: a string of zeros and ones represents the code of the input␣
 ↪text.
    Returns:
    - decoded_val: a real number between 0 and 1.
    """
    decoded_val = 0
    # ================================================================ #
    # YOUR CODE HERE:
    # ================================================================ #
    decoded_val =  int(txt_code, 2) / (2 ** len(txt_code))
    # ================================================================ #
    # END YOUR CODE HERE
    # ================================================================ #
    return decoded_val

decoded_val = decimal_decoding(txt_code)
print("The decoded Value:  ", decoded_val)
```

```
The decoded Value:   0.528045654296875
```

[27]:
```python
################################################################################
# Arithmetic Decoding
################################################################################
def Arithmetic_decode(decoded_val,CDF_symbols, n):
    """
    Inputs:
    - decoded_val: A real number between 0 and 1 represents the mid-point of␣
 ↪the interval of the encoded text.
    - CDF_symbols: A list containing the symbols and their corresponding CDF
    - n: number of symbols to be decoded
    Returns:
    - decoded_text: a string containing the decoded text.
    """
    decoded_text = ''
    # ================================================================ #
    # YOUR CODE HERE:
    # ================================================================ #
    sy = [i[0] for i in CDF_symbols]
    for i in range(n):
        j = 0
        while (CDF_symbols[j][1]<decoded_val):
            j+=1
        decoded_text += sy[j]
        l = 0
        if (j != 0):
            l = CDF_symbols[j-1][1]
        decoded_val = (decoded_val - l)/(CDF_symbols[j][1] - l)


    # ================================================================ #
    # END YOUR CODE HERE
    # ================================================================ #
    return decoded_text

decoded_text = Arithmetic_decode(decoded_val,CDF_symbols, len(text))
print("Orginal text:   ", text)
print("Decoded Text:   ", decoded_text)
```

```
Orginal text:    DACDDBCD
Decoded Text:    DACDDBCD
```

[ ]:
```python
```