Data_Compression_Project_Huffman

November 14, 2022

0.1 ECE 231A : Data Compression Project Module 1

Please follow our instructions in the same order and print out the entire results and codes when completed.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import csv
from collections import Counter
```

[3]:
```
###########################################################################################
# Read Text file
###########################################################################################
f = open("Toy_Example_Huffman.txt", "r")
text = f.read()
```

[25]:
```
###########################################################################################
# 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:
    # ================================================================ #
    chars, counts = np.unique(list(text), return_counts=True)
    symbols = [(str(char), count/len(text)) for char, count in zip(chars,␣
 ↪counts)]
    # ================================================================ #
    # 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)]
```

0.2 Part 1: D-ary Huffman Codes

[26]:
```
##########################################################################################
# Draw the tree for the Huffman code
##########################################################################################
def Huffman_tree(symbols, D = 3):
    """
    Inputs:
    - 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.

    Returns:
    - tree: a list of a single element that have probability one. at each␣
 ↪iteration sort your list according
             to their probabilities and combine the first D elements as a single␣
 ↪element
    """
    # ============================================================== #
    # YOUR CODE HERE:
    # ============================================================== #
    #check if we need to add dummy symbols

    if len(symbols)==1:
        return symbols
    if (len(symbols)-1)%(D-1)>0:
        for i in range((D-1)-((len(symbols)-1)%(D-1))):
            symbols.append(('dummy'+str(i),0))
    char,prob=zip(*symbols)
    char=list(char)
    prob=list(prob)

    #sort the chars and probs
    sorted_chars=[x for _,x in sorted(zip(prob,char),key=lambda pair: pair[0])]
    sorted_probs=sorted(prob)
    #combine the first D elements as a single element
    tree=[(tuple(sorted_chars[:D]),sum(sorted_probs[:D]))]
```

```python
    #add the rest of the elements
    for i in range(D,len(sorted_chars)):
        tree.append((sorted_chars[i],sorted_probs[i]))
    return Huffman_tree(tree,D)
    # ================================================================ #
    # END YOUR CODE HERE
    # ================================================================ #
    return tree

tree = Huffman_tree(symbols)
print(tree[0][0])
```

```
(('dummy0', 'A', 'B'), 'C', 'D')
```

[28]:
```python
########################################################################################
#Encode the Huffman Tree
########################################################################################
def Huffman_coding(seq, code='', D =3):
    """
    Inputs:
    - seq: a tuple of characters.
    - code: the code of this tuple
    Returns:
    - Dictionary: a dictionary containing the Huffman codes.
    """
    if type(seq) is str:
        return {seq : code}
    Dictionary = dict()
    # ================================================================ #
    # YOUR CODE HERE:
    # ================================================================ #
    dicts=[{}]*D
    for i in range(D):
        dicts[i]=Huffman_coding(seq[i],code+str(i),D)
    for d in dicts:
        Dictionary.update(d)
    # ================================================================ #
    # END YOUR CODE HERE
    # ================================================================ #
    return Dictionary

Huffman_code = Huffman_coding(tree[0][0])
print("Huffman CodeBook:  ", Huffman_code)
```

```
Huffman CodeBook:   {'dummy0': '00', 'A': '01', 'B': '02', 'C': '1', 'D': '2'}
```

[29]:
```python
##############################################################################
# Compute the expected length of the Huffman code
##############################################################################
def compute_expected_length(symbols, Huffman_code, D =3):
    """
    Inputs:
    - symbols: A list of tuples of the form (char,prob).
    - Huffman_code: a dictionary containing the Huffman codes.Each code is a␣
 ↪string
    Returns:
    - Expected_length: a number represents the expected length of Huffman code
    """

    # ================================================================ #
    # YOUR CODE HERE:
    # ================================================================ #
    Expected_length=0
    for symbol in symbols:
        char=symbol[0]
        Expected_length+=symbol[1]*len(Huffman_code[char])

    # ================================================================ #
    # END YOUR CODE HERE
    # ================================================================ #
    return Expected_length

Expected_length = compute_expected_length(symbols, Huffman_code)
print("Expected length of Huffman code:   ", Expected_length)
```

```
Expected length of Huffman code:    1.25
```

[30]:
```python
##############################################################################
# Encode a text
##############################################################################
def encode_text(text, Huffman_code, D =3):
    """
    Inputs:
    - text: A string containing the text to be encoded.
    - Huffman_code: a dictionary containing the Huffman codes.Each code is a␣
 ↪string
    Returns:
    - txt_code: a string represents the code of the input text.
    """
    txt_code = ''
    # ================================================================ #
    # YOUR CODE HERE:
    # ================================================================ #
```

```
    for char in text:
        txt_code+=Huffman_code[char]
    # ================================================================ #
    # END YOUR CODE HERE
    # ================================================================ #
    return txt_code
txt_code = encode_text(text, Huffman_code)
print("Encoded Text:  ", txt_code)
```

```
Encoded Text:   2011220212
```

[31]:

```
#######################################################################################
# Decode a text
#######################################################################################
def decode_text(txt_code, Huffman_code, symbols, D =3):
    """
    Inputs:
    -symbols: a list of symbols.
    - txt_code: A code of a text encoded by Huffman code as a string.
    - Huffman_code: a dictionary containing the Huffman codes. Each code is a␣
 ↪string
    Returns:
    - decoded_text: a string represents the decoded text.
    """
    decoded_text = ''
    # ================================================================ #
    # YOUR CODE HERE:
    # ================================================================ #
    r_Huffman_code={v:k for k,v in Huffman_code.items()}
    codeword=''
    for char in txt_code:
        codeword+=char
        if codeword in r_Huffman_code:
            decoded_text+=r_Huffman_code[codeword]
            codeword=''

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

decoded_text = decode_text(txt_code, Huffman_code, symbols)
print("Orginal text:   ", text)
print("Decoded Text:   ", decoded_text)
```

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

[ ]:

```

```