part1

November 15, 2024

1 Information Theory Project Part 1

Saifelden Mohamed Ismail - 202100432

1.0.1 requirement 1

Calculating probabilities

[46]:
```
import string
file_object = open("Test_text_file.txt")
file_raw=file_object.read();
symbols=string.ascii_lowercase+"().,/- "
symbols_probabilities=dict(zip(list(symbols),[0]*len(symbols)))
file_length=len(file_raw)

for i in range(file_length):
    symbols_probabilities[file_raw[i]]+=1*(1/file_length)

symbols_probabilities
```

[46]:
```
{'a': 0.07193732193732202,
 'b': 0.012108262108262113,
 'c': 0.026353276353276337,
 'd': 0.0455840455840456,
 'e': 0.10398860398860417,
 'f': 0.013532763532763538,
 'g': 0.01424501424501425,
 'h': 0.022079772079772072,
 'i': 0.0605413105413106,
 'j': 0.0007122507122507123,
 'k': 0.0007122507122507123,
 'l': 0.04202279202279203,
 'm': 0.02136752136752136,
 'n': 0.06410256410256417,
 'o': 0.06267806267806274,
 'p': 0.03703703703703703,
 'q': 0.0007122507122507123,
 'r': 0.05982905982905989,
 's': 0.056267806267806315,
```

```
 't': 0.07193732193732202,
 'u': 0.018518518518518517,
 'v': 0.009259259259259262,
 'w': 0.007834757834757837,
 'x': 0.004985754985754985,
 'y': 0.006410256410256412,
 'z': 0.002136752136752137,
 '(': 0.002136752136752137,
 ')': 0.002136752136752137,
 '.': 0.004273504273504274,
 ',': 0.007834757834757837,
 '/': 0.0007122507122507123,
 '-': 0.0113960113960114,
 ' ': 0.1346153846153847}
```

1.0.2 Requirement 2

Calculating Entropy

[47]:
```python
from math import log2
Entropy = sum([ i*-log2(i) if i!=0 else 0 for i in symbols_probabilities.
↪values()])
print(Entropy, "bits/symbol")
```

```
4.257010564738072 bits/symbol
```

1.0.3 Requirement 3

Fixed Length Code Calculation

[48]:
```python
fixed_length_bits_per_symbol=round(log2(len(symbols)))
fixed_length_efficiency= 6*sum([*symbols_probabilities.values()])
print(fixed_length_efficiency, "bits/symbol")
```

```
6.0000000000000036 bits/symbol
```

1.0.4 Requirement 4

Huffman Encoding function

[49]:
```python
import bisect


class Node:
    def __init__(self,left=None,right=None):
        self.left=left
        self.right=right
        if self.left!=None and self.right !=None :
            self.value=self.calculate_value()
    def calculate_value(self):
        left_value=self.left.value if isinstance(self.left, Node) else self.
↪left[1]
```

```
        right_value=self.right.value if isinstance(self.right, Node) else self.
↪right[1]
        return left_value +right_value

class encoder:
    def calculate_probabilities(self):
        file_object = open(self.file_name)
        file_raw=file_object.read();
        self.file_raw=file_raw
        self.symbols=list(set(file_raw))
        number_of_symbols=len(self.symbols)
        symbols_probabilities=[0]*number_of_symbols
        file_length=len(file_raw)
        for i in range(file_length):
            symbols_probabilities[self.symbols.index(file_raw[i])]+=(1/
↪file_length)

        symbols_probabilities_set= list(zip(self.symbols,symbols_probabilities))
        self.symbols_probabilities=sorted(symbols_probabilities_set, key=␣
↪lambda x: x[1] ,reverse=True)
    def generate_encoded_message(self):
        self.encoded_message=""
        for i in self.file_raw:
            self.encoded_message+=self.encoding[i]

    def generate_encoding(self,node=None,path="", leaves={}):
        if node is None:
            node= self.encoding_tree
        if not isinstance(node,Node):
            leaves[node[0]]=path
            return leaves
        self.generate_encoding(node.left,path+"0",leaves)
        self.generate_encoding(node.right,path+"1",leaves)
        return leaves



class huffman_encoder(encoder):
    def __init__(self,file_name,file_name_output):
        self.file_name=file_name
        super().calculate_probabilities()
        self.generate_huffman_tree()
        self.encoding=super().generate_encoding()
        super().generate_encoded_message()

        file_output=open(file_name_output,"w")
        file_output.write(self.encoded_message)
```

```python
        file_output.close()
    def generate_huffman_tree(self):
        tree_list=self.symbols_probabilities
        tree_list= sorted(tree_list, key=lambda x: x.value if isinstance(x, Node) else x[1])
        while(len(tree_list)>1):
            new_value=Node(tree_list.pop(0),tree_list.pop(0))
            bisect.insort(tree_list,new_value, key=lambda x:x.value if isinstance(x,Node) else x[1] )
        self.encoding_tree=tree_list[0]
```

[50]:

```python
from pandas import DataFrame as df
encoded_file=huffman_encoder("Test_text_file.txt","Test_text_file.zip")
table1=df({"Symbols":encoded_file.encoding.values(),"Encoding":encoded_file.encoding.keys()})
huffman_code=encoded_file.encoding.values()
table1
```

[50]:

	Symbols	Encoding
0	0000	d
1	000100	-
2	000101	b
3	00011	c
4	001	e
5	0100	s
6	010100	f
7	0101010	y
8	0101011	w
9	010110	g
10	0101110	,
11	01011110	.
12	010111110	z
13	010111111	(
14	0110	r
15	0111	i
16	1000	o
17	1001	n
18	101	
19	1100	a
20	1101	t
21	11100	p
22	111010	u
23	1110110	v

```
24     11101110        x
25    111011110        )
26   11101111100       k
27   11101111101       q
28   11101111110       /
29   11101111111       j
30         11110       l
31        111110       m
32        111111       h
```

1.0.5 REQUIREMENT 5

Huffman Decoding Function

[51]:
```
class decoder:
    def __init__(self,encoding_tree,file_input_name,file_output_name):
        self.encoding_tree=encoding_tree
        self.decode(file_input_name)
        self.file=open(file_output_name,"w")
        self.file.write(self.decoded_output)
        self.file.close()
    def decode(self,file_input_name):
        file_object = open(file_input_name)
        file_input_raw=file_object.read();
        self.file_input_raw=file_input_raw
        navigation_node=self.encoding_tree
        self.decoded_output=""
        for i in file_input_raw:
            if not isinstance(navigation_node,Node):
                self.decoded_output+=navigation_node[0]
                navigation_node=self.encoding_tree

            if i=='0':
                navigation_node=navigation_node.left
            elif i=='1':
                navigation_node=navigation_node.right
```

[52]:
```
decoded_file=decoder(encoded_file.encoding_tree,"Test_text_file.
↪zip","Test_text_file_unzipped.txt")
decoded_file.decoded_output
```

[52]:
```
'in this paper, a novel decorrelation-based concurrent digital predistortion
(dpd) solution is proposed for dual-band transmitters (tx) employing a single
wideband power amplifier (pa), and utilizing just a single feedback receiver
path. the proposed decorrelation-based parameter learning solution is both
flexible and simple, and operates in a closed-loop manner, opposed to the widely
applied indirect learning architecture. the proposed decorrelation-based
learning and dpd processing can also be effectively applied to more ordinary
```

```
single-band transmissions, as well as generalized to more than two transmit
bands. through a comprehensive analysis covering both the dpd parameter learning
and the main path processing, it is shown that the complexity of the proposed
concurrent dpd is substantially lower compared with the other state-of-the-art
concurrent dpd methods. extensive set of quantitative simulation and rf
measurement results are also presented, using a base-station pa as well as a
commercial lte-advanced mobile pa, to evaluate and validate the effectiveness of
the proposed dpd solution in various real world scenarios, incorporating single-
band/dual-band tx cases. the simulation and rf measurement results demonstrate
excellent linearization performance of the proposed concurrent dpd, even
outperforming current state-of-the-art methods, despite the significantly lower
complexity'
```

1.0.6 REQUIREMENT 6

Calculating efficiency of Huffman Code

[53]:
```
encoded_file.encoding
encoded_file.symbols_probabilities

average_length= sum([i[1]*len(encoded_file.encoding[i[0]]) for i in␣
 ↪encoded_file.symbols_probabilities])
print(f"the encoding's effieciency is {Entropy/average_length}")
```

```
the encoding's effieciency is 0.9954768209347515
```

1.0.7 REQUIREMENT 7

Creating a Shannon encoder

[64]:
```
class shannon_encoder(encoder):
    def __init__(self,file_name,file_name_output):
        self.file_name=file_name
        super().calculate_probabilities()
        self.generate_shannon_tree()
        self.encoding=super().generate_encoding()
        super().generate_encoded_message()

        with open(file_name_output, "w") as file_output:
            file_output.write(self.encoded_message)

    def generate_shannon_tree(self,symbols_probabilities=None,␣
 ↪navigation_node=None):

        if symbols_probabilities==None:
            symbols_probabilities=self.symbols_probabilities
        if navigation_node==None:
            self.encoding_tree=Node()
            navigation_node=self.encoding_tree
```

```python
        total_probability = sum(value for _, value in symbols_probabilities)
        accumulated_probabilities = 0
        midpoint = 0

        for i, (_, probability) in enumerate(symbols_probabilities):
            accumulated_probabilities += probability
            if accumulated_probabilities >= total_probability / 2:
                midpoint = i
                break

        left_split=symbols_probabilities[:midpoint]
        right_split=symbols_probabilities[midpoint:]
        if len(symbols_probabilities)==2:
            navigation_node.left=symbols_probabilities[0]
            navigation_node.right=symbols_probabilities[1]
            return
        if len(left_split)==1:
            navigation_node.left=symbols_probabilities[0]
            navigation_node.right=Node()
            self.generate_shannon_tree(right_split,navigation_node.right)
            return

        navigation_node.left=Node()
        navigation_node.right=Node()
        self.generate_shannon_tree(left_split,navigation_node.left)
        self.generate_shannon_tree(right_split,navigation_node.right)
```

[61]:
```python
from pandas import DataFrame as df
encoded_shannon=shannon_encoder("Test_text_file.txt","Test_text_file_Shannon.
 ↪zip")
symbols_probabilities=encoded_shannon.symbols_probabilities

table=df({"Symbols":encoded_shannon.encoding.values(),"Encoding":
 ↪encoded_shannon.encoding.keys()})
table
```

[61]:

	Symbols	Encoding
0	1100	d
1	1111011	-
2	1111010	b
3	110111	c
4	010	e
5	10111	s

```
6          111100     f
7        11111101     y
8         1111101     w
9         1110111     g
10       11111100     ,
11     1111111100     .
12     1111111101     z
13     1111111110     (
14          10110     r
15           1010     i
16            100     o
17          01111     n
18             00      
19           0110     a
20          01110     t
21         110110     p
22        1110110     u
23        1111100     v
24       11111110     x
25    11111111110     )
26   111111111110     k
27  1111111111110     q
28 11111111111110     /
29 11111111111111     j
30          11010     l
31         111010     m
32          11100     h
```

1.0.8 Requirement 8

Using the Decoder (same for both shannon and huffman) and Comparing the efficiency The efficiency for both is identical, to verify if there is an error, I used the encoding tree from the previous huffman encoding for comparison to show that both encodings are in fact seperate

[69]:
```python
decoded_file=decoder(encoded_shannon.encoding_tree,"Test_text_file_Shannon.zip","Test_text_file_unzipped_Shannon.txt")
decoded_file_wrong=decoder(encoded_file.encoding_tree,"Test_text_file_Shannon.zip","Test_text_file_unzipped_Shannon.txt")
print("Decoded file using shannon encoding tree")
print(decoded_file.decoded_output)
print("Decoded file using the previous huffman tree")
print(decoded_file_wrong.decoded_output)
```

```
Decoded file using shannon encoding tree
in this paper, a novel decorrelation-based concurrent digital predistortion
(dpd) solution is proposed for dual-band transmitters (tx) employing a single
wideband power amplifier (pa), and utilizing just a single feedback receiver
path. the proposed decorrelation-based parameter learning solution is both
```

```
flexible and simple, and operates in a closed-loop manner, opposed to the widely
applied indirect learning architecture. the proposed decorrelation-based
learning and dpd processing can also be effectively applied to more ordinary
single-band transmissions, as well as generalized to more than two transmit
bands. through a comprehensive analysis covering both the dpd parameter learning
and the main path processing, it is shown that the complexity of the proposed
concurrent dpd is substantially lower compared with the other state-of-the-art
concurrent dpd methods. extensive set of quantitative simulation and rf
measurement results are also presented, using a base-station pa as well as a
commercial lte-advanced mobile pa, to evaluate and validate the effectiveness of
the proposed dpd solution in various real world scenarios, incorporating single-
band/dual-band tx cases. the simulation and rf measurement results demonstrate
excellent linearization performance of the proposed concurrent dpd, even
outperforming current state-of-the-art methods, despite the significantly lower
complexity
Decoded file using the previous huffman tree
 epiiewn n  nghdrcpmeroaglg nrnntfimhstigciohiv  rslprw uie  cr
nryxnriyephvrtnhp.r tntfinwn  rn  eigcanrentn  i mnnmduaamhyiea r ahpjiho,
shrnhicbuilmncv o uripe  el gcigt wa egeht n h/dahdviwyht ehiehhltiida.nhitecas
nurthho nru iawcrriiehoetasrtrn  eigcb p  rgnntfimhstigcrr n teea re ee
ritehiein iriyep ,i siidlrnil i rndahd uut  eipcem-tngrie,fleotl eigi une ouri u
m-t   eigdudxecv o  iue      f ofhewerxe ee ritehicgtmfp  ttt r(ac asrtrn  eigcb
p  rgnntfimhstigcsn ritehicemce  ortrn t.lnhie tnprt.cted plerx iag mot     f
oetdunrss  nsltrhs uilmni mnnmduaamhy ) cjac,itg  dtietteu n  whwdisets
ndxenpiitoetrritm spi enm(mdxgnt tmdarligtr,eell ledalt ivl ie pmeg ehieuoxdxece
oratrrusp re ee ritehicemdxec e slraaxcr  e t.lnhihd eayp msm piiceaetasrligt
eil etmnepc asrtrn  eigciohiv  rslprrtoyp mtu taalun  r mn sm ge putn rgcvntac
assiibreiiena xinvxei   riciohiv  rslprrtoisntannihabheanll lebteaslchlvri tspau
lebuuvtentfioahd  petee iv nisnxerwlt eticgse  .cr n,nxgipet i ehiccte i.xpau
catnor am r  dticcintis rturtcsp vtnlenmuaetsl gncrrhdisble  irriedahdme
wceaoii-einpgx iasl i anpc asrtrn  eigce  o.r tntfinsliar  fv a ne  cvgtrblui  r
s haeshinrt errifh a uilmni mnnmihlavrti mnnmdjcir t(mdxebuuvtentfioahd  petee
iv nisnxerwlt etieo tsitttrried hrutrnexe fln r im ntfin ngls  tenmueepc asrtrn
eigciohiv  rslprrtnmd leepntnt ngls  tfh atmtr nexeiiena xinvxei   ric
eetannihdawv eaoii-. ttl p rvi t iur el ge put  eil etmh
```

[70]:
```python
average_length= sum([i[1]*len(encoded_shannon.encoding[i[0]]) for i in␣
 ↪encoded_shannon.symbols_probabilities])
print(f"the encoding's effieciency is {Entropy/average_length}")
```

```
the encoding's effieciency is 0.9530924625884625
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: