Home Assignment 2, Information Theory

Joel Bångdal (dat15jba)
Viktor Claesson (dat15vcl)

Kurs
EITN45
Lund Universitet

May 8, 2020

character	probability	code	character	probability	code
’ ’	0.194638	00	x	0.000970	1010001000
h	0.047737	0100	C	0.000970	1010001001
\n	0.024299	01010	F	0.000498	10100010100
m	0.012843	010110	z	0.000519	10100010101
:	0.001569	010111000	G	0.000552	10100010110
W	0.001596	010111001	K	0.000552	10100010111
T	0.003179	01011101	A	0.004297	10100011
.	0.006580	0101111	,	0.016285	101001
o	0.053643	0110	,	0.016285	101001
a	0.054882	0111	w	0.016413	101010
f	0.012971	100000	g	0.016473	101011
N	0.000808	1000010000	t	0.068776	1011
*	0.000404	10000100010	r	0.035648	11000
P	0.000431	10000100011	b	0.009314	1100100
q	0.000842	1000010010	-	0.004506	11001010
J	0.000054	1000010011000	Q	0.000566	11001011000
X	0.000027	100001001100100	B	0.000613	11001011001
_	0.000027	100001001100101	O	0.001185	1100101101
]	0.000013	1000010011001100	E	0.001266	1100101110
[	0.000013	1000010011001101	D	0.001293	1100101111
9	0.000007	10000100110011100	p	0.009819	1100110
2	0.000007	10000100110011101	)	0.000370	111101001110
\x1a	0.000007	10000100110011110	(	0.000377	111101001111
Z	0.000007	10000100110011111	I	0.004937	11001110
V	0.000283	100001001101	v	0.005408	11001111
U	0.000445	10000100111	s	0.042275	11010
j	0.000929	1000010100	i	0.045649	11011
R	0.000943	1000010101	e	0.090119	1110
H	0.001913	100001011	u	0.022912	111100
k	0.007247	1000011	S	0.001468	1111010100
y	0.014480	100010	”	0.000761	11110101010
c	0.015174	100011	Y	0.000768	11110101011
l	0.031081	10010	!	0.003024	111101011
d	0.031917	10011	’	0.011860	1111011
‘	0.007462	1010000	n	0.046423	11111
M	0.001347	1111010001	;	0.001307	1111010000
?	0.001360	1111010010	L	0.000660	11110100110

Total entropy(rounded too two decibels): 4.51

	ASCII length	Encoded length
total bits	1 187 848	676 374
bits/word	8	4.56

As we can see, the encoded length is not far of from the entropy.

Appendix

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import operator
import math


# SOURCE CHARACTER CODING
def inc(e, D):
    if e in D:
        D[e] += 1
    else:
        D[e] = 1


# Format of output: {char1: occurrences1, char2:
   occurrences2, ...}
def probabilities(fileName):
    output = {}
    with open(fileName) as f:
        c = f.read(1)
        while c:
            inc(c, output)
            c = f.read(1)

    characters = 0
    for key in output:
        characters += output[key]
```

```
    for key in output:
        output[key] /= characters

    return output, characters


# HUFFMAN COODING
class Leaf:
    def __init__(self, pair):
        self.char = pair[0]
        self.prob = pair[1]

    def __repr__(self):
        return f"Leaf({self.char}, {self.prob})"


class Node:
    def __init__(self, left, right):
        self.left = left
        self.right = right
        self.prob = left.prob + right.prob

    def __repr__(self):
        return f"Node({self.left}, {self.right}, {self
            .prob})"


def huffman(source):
    for i in range(len(source)):
        source[i] = Leaf(source[i])

    while(len(source) > 1):
        x_i = source.pop()
        x_j = source.pop()

        source.append(Node(x_i, x_j))
        source.sort(key=operator.attrgetter("prob"),
            reverse=True)

    return source[0]
```

```
# OUTPUT CODEWORDS
def codewords(huffman, infile, prefix=""):
    if(isinstance(huffman, Leaf)):
        infile.write(f"{repr(huffman.char)}, {huffman.
            prob:.6f}, {prefix}\n")
    else:
        codewords(huffman.left, infile, f"{prefix}0")
        codewords(huffman.right, infile, f"{prefix}1")


# HUFFMAN NODE TREE TO CODEWORD DICT
def tree2dict(huffman, out={}, prefix=""):
    if(isinstance(huffman, Leaf)):
        out[huffman.char] = len(prefix)
    else:
        tree2dict(huffman.left, out, f"{prefix}0")
        tree2dict(huffman.right, out, f"{prefix}1")
    return out


# HUFFMAN CONVERTER
def encoded_length(fileName, huffdict):
    size = 0
    with open(fileName) as f:
        for line in f:
            for c in line:
                size += huffdict[c]
        f.close()
    return size


# ENTROPY
def entropy(probabilities):
    ent = 0
    for k in probabilities:
        ent += -probabilities[k] * math.log(
            probabilities[k], 2)
    return ent
```

```
if __name__ == '__main__':
    source, characters = probabilities("Alice29.txt")
    print(f"Entropy: {entropy(source):.3f}")
    print(f"Decoded avg length: {8:.3f}, total length:
        {8 * characters:7}")
    source_sorted = sorted(
        source.items(), key=operator.itemgetter(1),
            reverse=True)

    huffm = huffman(source_sorted)
    huffdict = tree2dict(huffm)

    enc_len = encoded_length("Alice29.txt", huffdict)
    print(
        f"Encoded avg length: {enc_len / characters:.3
            f}, total length: {enc_len:7}")

    with open("source.out", "w") as f:
        codewords(huffm, f)
        f.close()
```