HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

ADVANCED CLASS 1501

PROJECTS FOR INFORMATION THEORY

Implementation of Huffman Coding and Shannon Coding

Author:
Haoran YOU

Supervisor:
Yayu GAO

June 10, 2017

Contents

1 Introduction

Here we will talk with two Coding Methods: *Huffman Coding* and *Shannon Coding* for the document *Steve Jobs Speech.doc*. It was presented by *Shannon*[1] the first time that the concept of information entropy and the *Shannon's First theorem*, which is called *zero-error coding theorem* for foreseeing the average code length per symbol of an instantaneous code can be made as closes to the entropy, but never be smaller.

2 Restatement of the Problem

The objective of this programming project is to deeply understand the Huffman coding and Shannon coding method. Here are the assignment:

- Collect the statistics of the letters, punctuation, space in the document *"Steve Jobs Speech.doc"*.
- Compute the entropy of *"Steve Jobs Speech.doc"*.
- Apply the *Huffman coding method* and *Shannon coding method* for *Steve Jobs Speech.doc*. Output the letters/punctuation/space and their Huffman codewords and Shannon codes. respectively.
- Decode the code which generated by the *Huffman coding method* and *Shannon coding method*, respectively.
- Compute the average code length of *Steve Jobs Speech.doc* using Huffman codes and Shannon codes, respectively.

3 Detailed requirements

3.1 Collect the statistics of *Steve Jobs Speech.doc*

First we should get the content of this document, the corresponding module in my code is *"Load Data"*. And then we can vote for different symbols in this document to get the frequency of occurrence for each symbol, which corresponding to the module *"Text Statistics"* in my code. The result can be seen from Table1: the number of different characteristics are $71 - symbols$, and the number of total characteristics in this document are $11838 - symbols$.

Table1: Symbols statistics in *Steve Jobs Speech.doc*.

[1]who published ≪A Mathematical Theory of Communication≫ in Bell System Technical Journal.1948.

symbols	Times	Probability
	28	2.365264e-03
	2228	1.882075e-01
”	12	1.013685e-03
$	1	8.447373e-05
’	40	3.378949e-03
,	101	8.531847e-03
-	9	7.602636e-04
.	142	1.199527e-02
0	11	9.292110e-04
1	7	5.913161e-04
2	2	1.689475e-04
3	6	5.068424e-04
4	1	8.447373e-05
5	2	1.689475e-04
6	2	1.689475e-04
7	5	4.223686e-04
8	1	8.447373e-05
9	2	1.689475e-04
:	9	7.602636e-04
;	2	1.689475e-04
?	4	3.378949e-04
A	31	2.618686e-03
B	11	9.292110e-04
C	4	3.378949e-04
D	9	7.602636e-04
E	5	4.223686e-04
F	3	2.534212e-04
G	2	1.689475e-04
H	6	5.068424e-04
I	116	9.798953e-03
J	1	8.447373e-05
K	2	1.689475e-04
L	5	4.223686e-04
M	11	9.292110e-04
N	9	7.602636e-04
O	4	3.378949e-04
P	4	3.378949e-04
R	5	4.223686e-04
S	22	1.858422e-03
T	22	1.858422e-03
W	12	1.013685e-03
X	3	2.534212e-04
Y	5	4.223686e-04
a	741	6.259503e-02
b	121	1.022132e-02
c	215	1.816185e-02
d	408	3.446528e-02
e	1074	9.072478e-02
f	203	1.714817e-02
g	205	1.731711e-02
h	434	3.666160e-02
i	526	4.443318e-02
j	8	6.757898e-04
k	63	5.321845e-03
l	397	3.353607e-02
m	205	1.731711e-02
n	588	4.967055e-02
o	768	6.487582e-02
p	179	1.512080e-02
q	4	3.378949e-04
r	493	4.164555e-02
s	489	4.130765e-02
t	906	7.653320e-02
u	283	2.390607e-02
v	115	9.714479e-03
w	233	1.968238e-02
x	13	1.098158e-03
y	252	2.128738e-02
z	4	3.378949e-04
–	3	2.534212e-04
¢	1	8.447373e-05
71	11838	1

3.2 Compute the entropy of document

The definition formula of Information entropy can be drived as:

$$H(X) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i} \tag{3.1}$$

Where X is a random variable, p_i is its probability distribution. If we view the document as a random variable, we can get it's probability distribution as Table1. So the entropy of this document can be calculated as:

$$H_{Steve-Jobs-Speech} \approx 4.387752 \tag{3.2}$$

3.3 Coding for the document

3.3.1 Huffman Coding

The fundamental thought of *Huffman Coding* is make the code symbols whose occurrence probability is large as short as possible. So we should sort the symbols according to its occurrence probability and link the two least likely outcome symbols as a new symbol whose occurrence probability equals to the sum of that probabilities of two original symbols.

Recur the symbols until we get two symbols only, so it's easy for us to code them as $'0'$ and $'1'$. Then we find their l-child and r-child, coding them as $'x0'$ and $'x1'$, respectively. Finally we can get the instantaneous code which has the average code length near to information entropy of this document, called *Huffman code*.

Here is our implementation of *Huffman Coding*:

```
function w = Huffman(r, P, S)
%
% Function  : Huffman coding
% input     : r --- the number of the source symbols
%             P --- the probability distribution of source symbols
%             S --- the number of source symbols si
% output    : w --- the Huffman codewords wi corresponding to si
%
format long;
if(r == 2)
    w{S(1)} = '0';
    w{S(2)} = '1';
else
    [P_descend, idx] = sort(P, 'descend');    % sort {Pi} in descending order
    S = S(idx);                               % sort {Si}
    P = [P_descend(1:r-2)  (P_descend(r-1) + P_descend(r))];    % update P
    lchild = S(r - 1);                        % left child
    rchild = S(r);                            % right child
    S = S(1:r-1);                             % update S
    w = Huffman(r-1, P, S);                   % recursion
    w{rchild} = [w{lchild} '1'];
    w{lchild} = [w{lchild} '0'];
end
```

Code1: Huffman Coding

The result of *Huffman Coding* can be seen as:

Table2: Huffman code for *Steve Jobs Speech.doc*

Symbols	Pi	Code
	2.365264e-03	110011010
	1.882075e-01	01
”	1.013685e-03	1100110111
$	8.447373e-05	0001000001111
’	3.378949e-03	00010011
,	8.531847e-03	1100111
-	7.602636e-04	10010100010
.	1.199527e-02	000101
0	9.292110e-04	0001001000
1	5.913161e-04	11001100110
2	1.689475e-04	1100110010110
3	5.068424e-04	11001100111
4	8.447373e-05	00010000011100
5	1.689475e-04	1100110010111
6	1.689475e-04	1100110010100
7	4.223686e-04	00010001111
8	8.447373e-05	00010000011101
9	1.689475e-04	1100110010101

Symbols	Pi	Code
:	7.602636e-04	10010100011
;	1.689475e-04	1001010011010
?	3.378949e-04	100101001010
A	2.618686e-03	110011000
B	9.292110e-04	0001001001
C	3.378949e-04	100101001011
D	7.602636e-04	10010100000
E	4.223686e-04	00010010110
F	2.534212e-04	000100011100
G	1.689475e-04	1001010011011
H	5.068424e-04	00010000010
I	9.798953e-03	1010110
J	8.447373e-05	10010100110010
K	1.689475e-04	1001010011000
L	4.223686e-04	00010010111
M	9.292110e-04	0001000110
N	7.602636e-04	10010100001
O	3.378949e-04	100101001000
P	3.378949e-04	100101001001
R	4.223686e-04	00010010100
S	1.858422e-03	000100001
T	1.858422e-03	000100010
W	1.013685e-03	0001000000
X	2.534212e-04	000100011101
Y	4.223686e-04	00010010101
a	6.259503e-02	0000
b	1.022132e-02	1001011
c	1.816185e-02	110010
d	3.446528e-02	11011
e	9.072478e-02	1000
f	1.714817e-02	111010
g	1.731711e-02	110100
h	3.666160e-02	11000
i	4.443318e-02	0011
j	6.757898e-04	11001100100
k	5.321845e-03	10010101
l	3.353607e-02	11100
m	1.731711e-02	110101
n	4.967055e-02	0010
o	6.487582e-02	1111
p	1.512080e-02	111011
q	3.378949e-04	100101001110
r	4.164555e-02	10011
s	4.130765e-02	10100
t	7.653320e-02	1011
u	2.390607e-02	00011
v	9.714479e-03	1010111
w	1.968238e-02	101010
x	1.098158e-03	1100110110
y	2.128738e-02	100100
z	3.378949e-04	100101001111
–	2.534212e-04	000100000110
¢	8.447373e-05	10010100110011

3.3.2 Shannon Coding

The construction of *Shannon Code* is very ingenious, it use the conversion of decimal and binary to get instantaneous coding method.

The basic steps of our Shannon Coding Algorithm are:

Step 1 Sort probability distribution $P(i)$ in descending order.

Step 2 For i, changed from 1 to r, where r is total number of our symbols, we can calculate the cumulative distribution $F(i)$, and change it to binary description.

Step 3 Then we should calculate the shortest code length for the probability $P(i)$, denoted as $l(i)$.

Step 4 Take $l(i)$ digits after the dot of binary description $F(i)$ as the codeword for the source symbol $S(i)$.

Here is our implementation of *Shannon Coding*:

```
function w = Shannon(r, P, S)
%
% Function : Shannon coding
% input    : r --- the number of the source symbols
%            P --- the probability distribution of source symbols
%            S --- the number of source symbols si
% output   : w --- the Huffman codewords wi corresponding to si
%
format long;
[P_descend, idx] = sort(P, 'descend');      % sort {Pi} in descending order
F = zeros(1,r);                             % initialize {Fi}
l = zeros(1,r);                             % initialize {li}
for i = 1:r
    F(i) = sum(P_descend(1:i-1));           % cumulative distribution function
    l(i) = ceil(log2(1 / P_descend(i)));    % codelength
    F_binary = dec_to_bin(F(i), l(i));      % convert decimal to binary

    % take li digit after the dot as the codeword for the source symbol idx(i)
    w{idx(i)} = num2str(F_binary);          % convert to string
    w{idx(i)} = strrep(w{idx(i)}, ' ', ''); % eliminate space
end
```

Code2: Shannon Coding

The result of *Shannon Coding* can be seen as:

Table2: Shannon code for *Steve Jobs Speech.doc*

Symbols	Pi	Code
	2.365264e-03	111110011
	1.882075e-01	000
”	1.013685e-03	1111101110
$	8.447373e-05	1111111111001
’	3.378949e-03	111110000
,	8.531847e-03	1111010
-	7.602636e-04	11111100111
.	1.199527e-02	1110101
0	9.292110e-04	11111100010
1	5.913161e-04	11111101111
2	1.689475e-04	111111110010
3	5.068424e-04	11111110000
4	8.447373e-05	1111111111010
5	1.689475e-04	111111110100
6	1.689475e-04	111111110101
7	4.223686e-04	111111100101
8	8.447373e-05	1111111111011
9	1.689475e-04	111111110111
:	7.602636e-04	11111101001
;	1.689475e-04	111111111000
?	3.378949e-04	111111101110
A	2.618686e-03	111110010
B	9.292110e-04	11111100011
C	3.378949e-04	111111101111
D	7.602636e-04	11111101010
E	4.223686e-04	111111100111
F	2.534212e-04	111111110110
G	1.689475e-04	111111111001
H	5.068424e-04	11111110001
I	9.798953e-03	1110111
J	8.447373e-05	1111111111101
K	1.689475e-04	111111111011
L	4.223686e-04	111111101000
M	9.292110e-04	11111100101
N	7.602636e-04	11111101100
O	3.378949e-04	111111110000
P	3.378949e-04	111111110010
R	4.223686e-04	111111101010
S	1.858422e-03	1111101010
T	1.858422e-03	1111101011
W	1.013685e-03	1111101111
X	2.534212e-04	111111110111
Y	4.223686e-04	111111101100
a	6.259503e-02	0110
b	1.022132e-02	1110110
c	1.816185e-02	110101
d	3.446528e-02	10110
e	9.072478e-02	0011
f	1.714817e-02	111000
g	1.731711e-02	110110
h	3.666160e-02	10101
i	4.443318e-02	10001
j	6.757898e-04	11111101110
k	5.321845e-03	11110111
l	3.353607e-02	10111
m	1.731711e-02	110111
n	4.967055e-02	01111
o	6.487582e-02	0101
p	1.512080e-02	1110011
q	3.378949e-04	111111110011
r	4.164555e-02	10010
s	4.130765e-02	10011
t	7.653320e-02	0100
u	2.390607e-02	110000
v	9.714479e-03	1111001
w	1.968238e-02	110011
x	1.098158e-03	1111101101
y	2.128738e-02	110010
z	3.378949e-04	111111110100
–	2.534212e-04	111111111000
¢	8.447373e-05	1111111111110

3.4 Decoding for the code of document

3.4.1 Decoding Huffman Code

After coding the document *Steve Jobs Speech.doc*, we must decode it to verify the correctness of our *Huffman Coding*. Here we load a string from the *'coding Huffman.txt'*, which is the result of *Huffman Coding*, and compare the part of string, form start to next characteristic, to that $71 - symbols$ codes, translate it to the corresponding symbols till the end of string, and find that we translate it to *'Decoding Huffman.txt'*, which is the same as original document!

Here is our implementation of *Decoding huffman code*.

```
fids = fopen('Coding_Huffman.txt','r');                % get code
[A,COUNT] = fscanf(fids,'%c');

fidss = fopen('Decoding_Huffman.txt','wt');            % Decoding Huffman code
now = 1;
next = 1;
while (next ≤ COUNT)
    idx = 0;
    for i = 1:num_char
        if(strcmp(A(now:next), w_H{i}) == 1)           % compare to the codewords
            idx = i;
        end
    end
    if(idx ≠ 0)
        fprintf(fidss, '%s', char_doc{idx});           % translate to symbols
        now = next + 1;
        next = now + 1;
    else
        next = next + 1;                               % +1
    end
end
fclose(fids);
```

Code3: Decoding Huffman code

3.4.2 Decoding Shannon Code

The process of decoding *Shannon code* is the same as decoding *Huffman code*. The only difference is we must compare our part of string to the *Shannon Codeword* instead of *Huffman Codeword*. The results turns to be alright with our methods.

Here is our implementation of *Decoding Shannon code*.

```
fids = fopen('Coding_Shannon.txt', 'r');                % get code
[A,COUNT] = fscanf(fids, '%c');

fidss = fopen('Decoding_Shannon.txt', 'wt');            % Decoding Shannon code
now = 1;
next = 1;
while (next <= COUNT)
    idx = 0;
    for i = 1:num_char
        if(strcmp(A(now:next), w_S{i}) == 1)            % compare to the codewords
            idx = i;
        end
    end
    if(idx ~= 0)
        fprintf(fidss, '%s', char_doc{idx});            % translate to symbols
        now = next + 1;
        next = now + 1;
    else
        next = next + 1;                                % + 1
    end
end
fclose(fids);
```

Code4: Decoding Shannon code

3.5 Average code length

3.5.1 Average codelength for Huffman coding

The average code length for this document with *Huffman Coding* is:

$$l_{Huffman-Coding} \approx 4.428873 \tag{3.3}$$

3.5.2 Average codelength for Shannon coding

The average code length for this document with *Shannon Coding* is:

$$l_{Shannon-Coding} \approx 4.774286 \tag{3.4}$$

4 Conclusion

From the results of two coding methods, we could find that *Huffman Coding* is more efficient compared to *Shannon Coding*. But neither of them can be less than or equal to the information entropy of this document, which can be seen as the replication experiment for the correctness of *Shannon's First Theorem*!

Through this programming project, I get a deeper understanding of information entropy and the two coding methods: *Huffman Coding* and *Shannon Coding*. This programming experiment can promote my learning of *Information Theorem Course* and the ability for implementing algorithm. Thanks to this programming project, I could implement coding process by myself without extra help except the algorithm provided by teacher.

Wish I could go further in the information road!

References

[1] Thomas M.Cover , Joy A.Thomas . Elements of Information Theory. Wiley-Blackwell Press.