EE5139/EE6139
Solution 3

Semester I, AY2021/2022
Marco Tomamichel

Exercise 3.1 Huffman code [EE5139]

Consider a source that outputs independent letters according to the frequency that they appear in the English language (use the frequencies listed in Table 1).

a.) Calculate the entropy of this source.

Solution: Let $\mathcal{X}$ denote the alphabet and $X \in \mathcal{X}$. Then the entropy of this source is given by

$$H(X) = \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)} \approx 4.19.$$

b.) Construct a Huffman code for this source. You may either construct the Huffman code manually according to the algorithm discussed in the lecture, or write a compute program that does this for you.

Solution: One possible Huffman code: (See the Matlab code "huffman.m")

a	0111	b	001000	c	10100	d	01101	e	110	f	10101
g	001010	h	1111	i	0011	j	111000010	k	111001	l	01100
m	10110	n	0001	o	0100	p	001001	q	111000011	r	0101
s	0000	t	100	u	11101	v	1110001	w	10111	x	111000000
y	001011	z	111000001								

Listing 1: huffman.m

```
function [code,compression,avelen]=huffman5(p)
%HUFFMAN5
%HUFFMAN CODER FOR V5
% Format [CODE,COMPRESSION,AVELEN]=HUFFMAN5(P)
%
% P is the probability (or number of occurences) of each alphabet symbol
% CODE gives the huffman code in a string format of ones and zeros
% COMPRESSION gives the compression rate
% AVELEN gives the expected length of the code
%
% Huffman5 works by first building up a binary tree (eg p =[ .5 .2 .15 .15])
%
%      a_1     a_4
%    1/       1/
%    /        /
%  b3       b1
```

a	8.4%	b	1.5%	c	2.2%	d	4.2%	e	11.0%	f	2.2%	g	2.0%
h	6.0%	i	7.4%	j	0.1%	k	1.3%	l	4.0%	m	2.4%	n	6.7%
o	7.4%	p	1.9%	q	0.1%	r	7.5%	s	6.2%	t	9.2%	u	2.7%
v	0.9%	w	2.5%	x	0.1%	y	2.0%	z	0.1%				

Table 1: Statistical distribution of letters in the English language. Source: `https://en.wikipedia.org/wiki/Letter_frequency`, but normalized so that they add up to 100%.

```
%    \    /  \
%    0\ 1/   0\
%      b2      a_3
%        \
%        0\
%          a_2
%
% Such that the tree always terminates at an alphabet symbol and the
% symbols furthest away from the root have the lowest probability.
% The branches at each level are  labeled 0 and 1.
% For this example CODE would be
%     1
%     00
%     010
%     011
% and the compression rate 1.1111
% Sean Danaher University of Northumbria at Newcastle UK 98/6/4

p=p(:)/sum(p);    %normalises probabilities
c=huff5(p);
code=char(getcodes(c,length(p)));
compression=ceil(log(length(p))/log(2))/ (sum(code' ~= ' ')*p);
avelen=sum(sum(code' ~= ' ').*p);
%-------------------------------------------------------------------
function c=huff5(p)
% HUFF5 Creates Huffman Tree
% Simulates a tree structure using a nested cell structure
% P is a vector with the probability (number of occurences)
%   of each alphabet symbol
% C is the Huffman tree. Note Matlab 5 version
% Sean Danaher 98/6/4        University of Northumbria, Newcastle UK

c=cell(length(p),1);                    % Generate cell structure
for i=1:length(p)                               % fill cell structure with
    1,2,3...n
   c{i}=i;                                              %       (n=number of
        symbols in alphabet)
end
while size(c)~=2                                % Repeat till only two
    branches
        [p,i]=sort(p);                                  % Sort to ascending
            probabilities
        c=c(i);                                                 % Reorder
            tree.
        c{2}={c{1},c{2}};c(1)=[];       % join branch 1 to 2 and prune 1
        p(2)=p(1)+p(2);p(1)=[];         % Merge Probabilities
end
%-------------------------------------------------------------------
function y= getcodes(a,n)
% Y=GETCODES(A,N)
% Pulls out Huffman Codes for V5
% a is the nested cell structure created by huffcode5
```

```
% n is the number of symbols
% Sean Danaher 98/6/4   University of Northumbria, Newcastle UK
global y
y=cell(n,1);
getcodes2(a,[])
%-----------------------------------------------------------------
function getcodes2(a,dum)
% GETCODES(A,DUM)
%getcodes2
% called by getcodes
% uses Recursion to pull out codes
% Sean Danaher 98/6/4   University of Northumbria, Newcastle UK

global y
if isa(a,'cell')
         getcodes2(a{1},[dum 0]);
         getcodes2(a{2},[dum 1]);
else
   y{a}=setstr(48+dum);
end
```

c.) Compute the expected length of the codeword for this code. How does this compare to the entropy computed in a)?

Solution: Expected length: $4.22 \geq H(X) = 4.19$.

Exercise 3.2 Huffman code [all]

Which of the following sets of codewords can never be valid Huffman codes (for any assignment of probabilities)? Argue why.

a.) $\{0, 10, 11\}$,

Solution: Can.

b.) $\{00, 01, 10, 11\}$,

Solution: Can.

c.) $\{00, 01, 10, 110\}$,

Solution: Cannot. 110 can be shortened to 11.

d.) $\{01, 10\}$,

Solution: Cannot. The codewords can be shortened to $\{0, 1\}$.

e.) $\{1, 01, 10\}$.

Solution: Cannot. It is not a prefix code.

Exercise 3.3 A code that allows a prefix [EE5139]

As we have seen in the lecture, we like codes to be prefix-free as otherwise we do not know when a codeword ends and decoding might no longer be unique. One simple (but not necessarily very efficient) way to overcome this problem is to simply add a special symbol that indicates the end of a codeword. In this exercise we will thus consider codewords that are comprised of "0" and "1" and always end with a special space character "_".

a.) Construct a code for this source in the following way: the two most frequent laters are assigned codewords of length $1+1$, the next four most frequent letters codewords of lenght $2+1$, etc.

Solution: (See the Matlab code "ex3_2.m")

a	b	c	d	e	f	g	h	i	j	k
00_	0101_	0000_	011_	0_	0001_	0010_	010_	10_	1000_	0110_
l	m	n	o	p	q	r	s	t	u	v
100_	111_	000_	11_	0100_	1001_	01_	001_	1_	101_	0111_
w	x	y	z							
110_	1010_	0011_	1011_							

Listing 2: ex3_2.m

```
%%%Exercise 3.2 prefix code%%%

pp=[8.4,1.5,2.2,4.2,11,2.2,2.0,6,7.4,0.1,1.3,4,2.4,6.7,7.4,1.9,0.1,7.5,...
    6.2,9.2,2.7,0.9,2.5,0.1,2,0.1];
A='a':'z';
AA=mat2cell(A,1,ones(26,1));
AAA=[];

%sort probabilities and record the index
[pp0,pindex]=sort(pp,'descend');

%generate codewords
code={};
for i=1:4
    code(2^i-1:(2^i-1)*2)=cellstr(dec2bin(0:2^i-1));
end
code=code(1:26);

%combine symbols and codewords
for i=1:26
    AA(2,pindex(i))=cellstr([cell2mat(code(i)),'_']);
end

%expected length
Sum=0;
for i=1:26
    Sum=Sum+pp(i)*length(cell2mat(AA(2,i)));
end
Sum=Sum/100;

%expected time
time=0;
for i=1:26
   t0=2*length(find(cell2mat(AA(2,i))=='0'));
   t1=4*length(find(cell2mat(AA(2,i))=='1'));
   time=time+pp(i)/100*(t0+t1);
end
time=time+3
```

b.) Compute the expected length of the codewords.

Solution: Expected length: 3.433

c.) The code you arrived at is very similar to a very famous code that has been in use since the 1830s. Can you figure out which code that is? Compare the expected length of codewords of your code to that of this historical code.

Solution: Morse code. The expected length of Morse code is 3.558, which is longer than that of our code.

Hint: Replace "0" by "·" and "1" by "−" in your codewords.

Exercise 3.4 Kraft–McMillan inequality for uniquely decodable codes [all]

Assume a uniquely decodable code has codeword lengths $l_1, \ldots, l_M$. Our goal is to derive Kraft's inequality for uniquely decodable codes:

$$\sum_{j=1}^{M} 2^{-l_j} \leq 1\,.$$

a.) Prove the following identity (this is easy):

$$\left(\sum_{j=1}^{M} 2^{-l_j}\right)^n = \sum_{j_1=1}^{M}\sum_{j_2=1}^{M}\cdots\sum_{j_n=1}^{M} 2^{-(l_{j_1}+l_{j_2}+\ldots+l_{j_n})}.$$

Solution: Perform the multiplication...

b.) Let A_l be the number of concatenations of n codewords that have overall length $l = l_{j_1}+l_{j_2}+\ldots+l_{j_n}$ and let $l_{\max} = \max\{l_1, l_2, \ldots, l_M\}$ be the maximum length of a codeword. Show that

$$\left(\sum_{j=1}^{M} 2^{-l_j}\right)^n = \sum_{l=n}^{nl_{\max}} A_l\, 2^{-l}.$$

Solution: The smallest value this exponent can take is n, which would happen if all code words had the length 1. The largest value the exponent can take is $nl_{\max}$ where $l_{\max}$ is the maximal codeword length. The summation can then be written as above.

c.) Using unique decodability, show that $A_i \leq 2^i$ and hence

$$\left(\sum_{j=1}^{M} 2^{-l_j}\right)^n \leq nl_{\max}.$$

Use this to derive the desired inequality.

Solution: The number of possible binary sequences of length i is 2^i. Since the code is uniquely decodable, we must have $A_i \leq 2^i$ in order to be able to decode. Plugging this into the above bound yields

$$\left(\sum_{j=1}^{M} 2^{-l_j}\right)^n \leq \sum_{i=n}^{nl_{\max}} 2^i 2^{-i} = nl_{\max} - n + 1 \leq nl_{\max}.$$

We then have

$$\sum_{j=1}^{M} 2^{-l_j} \leq \left[n(l_{\max}-1)\right]^{1/n} = \exp\left[\frac{1}{n}\log(n(l_{\max}-1))\right]$$

The exponent goes to zero as $n \to \infty$ and hence, $\sum_{j=1}^{M} 2^{-l_j} \leq 1$, Kraft's inequality for uniquely decodable codes.

Exercise 3.5 Shannon code [all]

Let X be a random variable distributed on $\{0, 1, 2, \ldots, d-1\}$ $(d>1)$, and let P_X be its probability mass function. Without loss of generality, we assume $P_X(0) \geq P_X(1) \geq \cdots \geq P_X(d-1) > 0$. A Shannon code for this source is constructed as follows. For each $x \in \mathcal{X}$, we consider the binary representation of the real number $\sum_{x'<x} P_X(x')$, and use the first $\lceil \log_2 \frac{1}{P_X(x)} \rceil$ bits in its fractional part to represent x. For example, the binary representation of the real number $\frac{1}{3}$ is '0.01010101...', and the first 2 bits in its fractional part are '01'.

a.) Show that the above code is unique decodable.

Solution: Denote the codeword representing x as $C(x)$ (namely, for this specific random variable X, Shannon code maps each x to $C(x)$). It suffices to show the above code to be a prefix code. Suppose the opposite, i.e., one of $C(x)$ and $C(\tilde{x})$ is a prefix to the other for some $x < \tilde{x}$. Note that $P_X(x) \geq P_X(\tilde{x})$. Thus, $\lceil \log_2 \frac{1}{P_X(x)} \rceil \leq \lceil \log_2 \frac{1}{P_X(\tilde{x})} \rceil$; namely, $C(x)$ is shorter and must be a prefix to $C(\tilde{x})$. On the other hand, denoting $\ell = \lceil \log_2 \frac{1}{P_X(x)} \rceil$ the length of $C(x)$, we have

$$\left| \sum_{x'<x} P_X(x') - \sum_{x'<\tilde{x}} P_X(x') \right| \geq P_X(x) \geq 2^{-\ell}.$$

However, this implies a contradiction since the first ℓ bits of these two numbers are same. Hence, the above code must be a prefix code.

b.) Produce a Shannon code table for the following source

x	0	1	2	3
$P_X(x)$	1/2	1/6	1/6	1/6

and compute the expected code length.

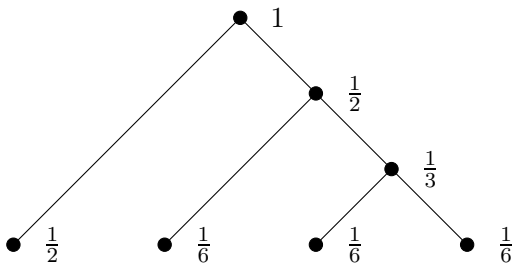
Solution:

x	$\sum_{x'<x} P_X(x)$	LHS in binary	$\lceil \log_2 \frac{1}{P_X(x)} \rceil$	codeword
0	0	0.00000000	1	0
1	1/2	0.10000000	3	100
2	2/3	0.10101010...	3	101
3	5/6	0.11010101...	3	110

The expected code length is $\frac{1}{2} \cdot 1 + 3 \cdot \frac{1}{6} \cdot 3 = 2$.

c.) Show that the expected length of the Huffman code of the above source is shorter.

Solution: The Huffman tree constructed for this case looks like the following:



The expected code length is $\frac{1}{2} \cdot 1 + \frac{1}{6} \cdot 2 + 2 \cdot \frac{1}{6} \cdot 3 = 1\frac{5}{6} < 2$.

Exercise 3.6 Huffman code algorithm [EE6139]

(from 2013/2014 final exam)

Consider a discrete memoryless source X with alphabet $\{1, 2, \ldots, M\}$. Suppose that the symbol probabilities are ordered and satisfy $p_1 > p_2 > \ldots > p_M$ and also satisfy $p_1 < p_{M-1} + p_M$. Let $l_1, l_2, \ldots, l_M$ be the lengths of a prefix-free code of minimum expected length for such a source.

a.) **(1 point)** Is the following statement true or false? $l_1 \leq l_2 \leq \ldots \leq l_M$. Argue why.

Solution: For an optimal code, if $p_i > p_j$, then $l_i \leq l_j$. Suppose otherwise, i.e., that $l_i > l_j$, then I can create a new code with lengths $l'_i = l_j$ and $l'_j = l_i$ for symbols i and j respectively. It is then easy to see that the expected cost will decrease. Indeed, the contribution to the expected cost from symbols i and j in the old code is $c_{ij}^{\text{old}} = p_i l_i + p_j l_j$. The contribution to the expected cost from symbols i and j in the new code is $c_{ij}^{\text{new}} = p_i l_j + p_j l_i$. Because $p_i > p_j$ and $l_i > l_j$, we have that $c_{ij}^{\text{old}} > c_{ij}^{\text{new}}$ which is a contradiction and so the initial assumption that $l_i > l_j$ must be false. Hence, $l_i \leq l_j$. This argument can be repeated for all symbols.

b.) **(2 points)** Show that if the Huffman algorithm is used to generate the above code, then $l_M \leq l_1 + 1$.

Solution: For the Huffman algorithm, we will first merge symbols $M - 1$ and M to form a symbol with probability $p_{M-1} + p_M$. The codeword for this new symbol is $l_M - 1$. Since $p_{M-1} + p_M > p_1$, by part (a), we must have that $l_1 \geq l_M - 1$ which is the desired result.

c.) **(2 points)** Show that $l_M \leq l_1 + 1$ for *any* (not necessarily Huffman generated) prefix-free code of minimum expected length. **Hint:** A minimum-expected-length code must be full.

Solution: A minimum-expected-length code must be full, and thus the codeword for letter M must have a sibling, say letter j. Since $p_j \geq p_{M-1}$, we have $p_j + p_M > p_1$ because of the condition $p_1 < p_{M-1} + p_M$. Let l' be the length of the intermediate node that is parent to j and M. Now $l' \leq l_1$ since otherwise the codeword for 1 could be interchanged with this intermediate node for a reduction in $\bar{L}$. Again $l_M - 1 \leq l_1$.

d.) **(2 points)** Suppose $M = 2^k$ for some integer k. Show that all codewords must have the same length. **Hint:** What does the Kraft inequality look like for an optimal code? Consider the three cases $l_1 = k$, $l_1 < k$ and $l_1 > k$.

Solution: An optimal prefix-free code must be full. For full codes, Kraft must be satisfied with equality, i.e., $\sum_{j=1}^{2^k} 2^{-l_j} = 1$. First assume that $l_1 = k$. Then all codewords have length k or $k + 1$, but as mentioned above, the Kraft's inequality can be satisfied with equality (i.e., the code can be full) only if all codewords have length k. If $l_1 > k$, then $l_j > k$ for all j and the Kraft inequality can not be satisfied with equality. Finally, if $l_1 < k$ with $l_j \leq k$, then the Kraft inequality cannot be met. Thus all codewords have length k.