BE1 Traitement et protection de l'information

URIEN, PRÉVOST

March 2, 2023

0.1 1. Huffman Algorithm

[93]:
```
clear all; close all; clc;
```

1. We first write the function `DMS(A, P, m, n)` that computes a discrete source matrix of size $m \times n$ from an alphabet A and a probability vector P.

[94]:
```
%%file DMS.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function DMS =  DMS(A, P, m, n)
% generate discrete source matrix from alphabet A and probabilities P
% m: number of rows, n: number of columns
% DMS: discrete source matrix
% returns DMS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function DMS =  DMS(A, P, m, n)
    DMS = randsrc(m, n, [A; P]);
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE1/DMS.m'.
```

We then try it with the following alphabet and probability vector, and compare the result with the one obtained with the `repmat` function.

[95]:
```
A = [0, 1];
P = [0.5, 0.5];
m = 5;
n = 6;
X = DMS(A, P, m, n);
```

[96]:
```
Y = repmat(A, m, n);
```

[97]:
```
X
Y
```

```
X =

     1     1     0     0     0     1
     1     1     0     1     1     1
```

```
     1     0     1     1     1     0
     1     1     0     1     0     1
     0     1     1     0     0     0


Y =

     0     1     0     1     0     1     0     1     0     1     0     1
     0     1     0     1     0     1     0     1     0     1     0     1
     0     1     0     1     0     1     0     1     0     1     0     1
     0     1     0     1     0     1     0     1     0     1     0     1
     0     1     0     1     0     1     0     1     0     1     0     1
```

2. That being done, we write the function `entropy(P)` that computes the entropy of a discrete random variable from its probability vector, and test it with a probability vector.

[98]:
```
%%file entropy.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function H = entropy(P)
% compute entropy of discrete random variable
% P: probability vector
% return H
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function H = entropy(P)
    H = -sum(P .* log2(P));
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE1/entropy.m'.
```

[99]:
```
P = [.2, .5, .1, .2];
entropy(P)
```

```
ans =

    1.7610
```

3. We write the function `moybits(N, P)` that computes the average number of bits per symbol from a probability vector and a vector of codeword lengths.

[100]:
```
%%file moybits.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function moybits = moybits(N, P)
% compute average number of bits per symbol
% P: probability vector
% N: length of codewords
% moybits: average number of bits per symbol
```

```
% returns moybits
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function moybits = moybits(N, P)
    moybits = sum(sum(P .* N)) / length(P);
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE1/moybits.m'.
```

4. Now, we try Matlab functions `huffmanenco` and `huffmandeco` to encode and decode a discrete source matrix.

[101]:
```
% test alphabet and probability vector
A = [1:6];
P = [.05, .125, .25, .12, .3, .155];

% discrete source
X = DMS(A, P, 3, 3);

% create dictionary
dict = huffmandict(A, P);
```

[102]:
```
% reshape X to a vector
X = reshape(X, 1, numel(X));
X

% encode
E = huffmanenco(X, dict)

% decode
D = huffmandeco(E,dict)
```

```
X =

     5     1     5     6     5     3     4     3     5


E =

  Columns 1 through 13

     0     0     1     1     1     0     0     0     1     0     0     0     1

  Columns 14 through 21

     0     1     1     0     1     0     0     0


D =
```

```
     5     1     5     6     5     3     4     3     5
```

We then try the same thing with different alphabets and probability vectors.

[103]:
```
A = [1:6];
P = [.05, .125, .25, .12, .3, .155];

A2 = [5:9];
P2 = [.05, .125, .25, .275, .3];

X = DMS(A, P, 3, 4);
X2 = DMS(A2, P2, 3, 4);

dict = huffmandict(A, P);
dict2 = huffmandict(A2, P2);

X = reshape(X, 1, numel(X))
E = huffmanenco(X, dict)
D = huffmandeco(E,dict)

X2 = reshape(X2, 1, numel(X2))
E2 = huffmanenco(X2, dict2)
D2 = huffmandeco(E2,dict2)
```

```
X =

     5     4     6     5     4     6     3     3     2     5     5     2


E =

  Columns 1 through 13

     0     0     1     1     0     0     1     0     0     0     1     1     0

  Columns 14 through 26

     0     1     0     1     0     1     0     0     1     1     0     0     0

  Columns 27 through 30

     0     0     1     1


D =
```

```
     5     4     6     5     4     6     3     3     2     5     5     2


X2 =

     9     8     9     8     7     6     7     7     8     5     8     8


E2 =

  Columns 1 through 13

     0     0     0     1     0     0     0     1     1     0     1     1     0

  Columns 14 through 26

     1     0     1     0     0     1     1     1     1     0     1     0     1


D2 =

     9     8     9     8     7     6     7     7     8     5     8     8
```

We now check the Kraft inequality for the two examples.

[104]:
```
% lengths of codewords
N = cellfun(@length, dict(:,2));
N2 = cellfun(@length, dict2(:,2));

%lengths of alphabets
L = length(A);
L2 = length(A2);

% efficiency
eff = entropy(P) / moybits(N, P)
eff2 = entropy(P2) / moybits(N2, P2)

% kraft inequality
kraft = sum(L.^(-N))
kraft2 = sum(L2.^(-N2))
```

```
eff =

    0.8986


eff2 =
```

```
    0.8852


kraft =

    0.0741


kraft2 =

    0.1360
```

5. We use the functions **arithenco** and **arithdeco** to encode and decode a discrete source matrix on several alphabets and probability vectors.

[105]:
```
A = [1:4];
P = [.1 .5 .3 .1];
X = DMS(A, P, 1, 500);
E = arithenco(X, [sum(X==1) sum(X==2) sum(X==3) sum(X==4)]);
D = arithdeco(E, [sum(X==1) sum(X==2) sum(X==3) sum(X==4)], 500);
"sum D - X = " + sum(D - X)

A2 = [1:6];
P2 = [.1 .2 .3 .1 .2 .1];
X2 = DMS(A2, P2, 1, 500);
E2 = arithenco(X2, [sum(X2==1) sum(X2==2) sum(X2==3) sum(X2==4) sum(X2==5) sum(X2==6)]);
D2 = arithdeco(E2, [sum(X2==1) sum(X2==2) sum(X2==3) sum(X2==4) sum(X2==5) sum(X2==6)], 500);
"sum D2 - X2 = " + sum(D2 - X2)
```

```
ans =

    "sum D - X = 0"


ans =

    "sum D2 - X2 = 0"
```

=> Encoding and decoding work fine, messages are the same before and after encoding and decoding.

[106]:
```
% Kraft inequality
L = length(A);
```

```
L2 = length(A2);
N = ceil(log2(L));
N2 = ceil(log2(L2));
kraft = sum(L.^(-N))
kraft2 = sum(L2.^(-N2))
```

```
kraft =

    0.0625


kraft2 =

    0.0046
```

0.2 2 Source coding of words with memory or unknown probability of occurrence

1. Constructing the `LempelZivDic` function using `DMS` and Matlab functions.

[107]:
```
%%file LempelZivDic.m

function Dict = LempelZivDic(X, n)
    %LempelZivDic - Generate a dictionary of length 2^n from the discrete
 ↪source X.
    %
    % Syntax: Dict = LempelZivDic(X,n)
    %
    % The output of this function is a vector of structures (dict.mot and dict.
 ↪code; a word of the code must be on n+1 bits).
    %
    % Input:
    % - X: vector representing the discrete source
    % - n: length of the code word (a word of the code must be on n+1 bits)
    % Output:
    % - Dict: vector of structures (dict.mot and dict.code)

    % Initialize the dictionary with the single symbols from the source
    Dict = struct('mot', num2str(X), 'code', num2str(0:length(X) - 1)');

    % Initialize the variables for the loop
    code_length = length(X);
    code_number = code_length;

    % Loop through the source to build the dictionary
    for i = 1:length(X)
```

```
        code_length = code_length + 1;
        code_word = dec2bin(X(i));
        j = i + 1;

        while j <= length(X) && code_length <= 2 ^ n
            code_word = strcat(dec2bin(code_word), dec2bin(X(j)));
            j = j + 1;

            if ~ismember(code_word, [Dict.mot])
                Dict(code_number + 1) = struct('mot', {code_word}, 'code',␣
↪code_number);
                code_number = code_number + 1;

                if code_number == 2 ^ n
                    break;
                end

                code_word = dec2bin(X(i));
                code_length = code_length + 1;
            end

        end

        if code_number == 2 ^ n
            break;
        end

    end

    % Remove any extra entries in the dictionary if necessary
    if code_number < 2 ^ n
        Dict = Dict(1:code_number);
    end

end
```

Created file '/Users/thomasprevost/github/ProtectInfo/BE1/LempelZivDic.m'.

[108]:
```
n1 = 4
X = (rand(1, 50) > 0.1);

Dict1 = LempelZivDic(X, n1);
Dict1.mot
```

```
n1 =

     4
```

```
ans =

    '1  1  1  1  0  1  1  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1'
```

2. Constructing the `LempelZivEnCo` function using `LempelZivDic` and Matlab functions.

[109]:
```
%%file LempelZivEnCo.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function X = LempelZivEnCo(X,Dict)
% LempelZivEnCo - Encodes a DMS X using the dictionary Dict.
% returns the encoded DMS X.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function E = LempelZivEnCo(X,Dict)
    % initialize variables
    E = [];
    n = log2(length(Dict));
    X = X(:)';
    X = [X, zeros(1, n)];
    i = 1;

    % loop over the DMS
    while i <= length(X) - n
        % find the longest word in the dictionary that matches the current
        % word
        for j = 1:length(Dict)
            if X(i:i+n) == Dict(j).mot
                E = [E, Dict(j).code];
                i = i + n + 1;
                break
            end
        end
    end

    % remove the last n bits
    E = E(1:end-n);
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE1/LempelZivEnCo.m'.
```

3. Constructing the `LempelZivDeCo` function using `LempelZivDic` and Matlab functions.

[110]:
```
%%file LempelZivDeco.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function Y = LempelZivDeco(X,Dict)
% LempelZivDeco - Decodes a received message Y using the Dict dictionary.
```

```
% returns the decoded DMS Y.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Y = LempelZivDeco(X,Dict)
    % initialize variables
    Y = [];
    n = log2(length(Dict));
    X = X(:)';
    X = [X, zeros(1, n)];
    i = 1;

    % loop over the DMS
    while i <= length(X) - n
        % find the longest word in the dictionary that matches the current
        % word
        for j = 1:length(Dict)
            if X(i:i+n) == Dict(j).code
                Y = [Y, Dict(j).mot];
                i = i + n + 1;
                break
            end
        end
    end

    % remove the last n bits
    Y = Y(1:end-n);
end
```

```
Created file '/Users/thomasprevost/github/ProtectInfo/BE1/LempelZivDeco.m'.
```

3. Applying functions on a large sequence (> 1000 bits)

[111]:

```
```

5. Given program to test the functions. Need to add the results in comments.

[112]:

```
%Programme I
n1 = 4;
n2 = 11;
n3 = 6;
rand('state', 0);
X = (rand(1, 50000) > 0.1);

Dict1 = LempelZivDic(X, n1);
longmot1 = Dict1(2 ^ n1).mot
lgmot1 = length(Dict1(2 ^ n1).mot)

Dict2 = LempelZivDic(X, n2);
lgmot2 = length(Dict2(2 ^ n2).mot)
```

```
X1 = LempelZivEnCo(X, Dict1);
R1 = length(X1) / length(X)

X2 = LempelZivEnCo(X, Dict2);
R2 = length(X2) / length(X)

Y = (rand(1, 1000) > 0.5);
Y1 = LempelZivEnCo(Y, Dict2);
RY1 = length(Y1) / length(Y)

DictY = LempelZivDic(Y, 6);
Y2 = LempelZivEnCo(Y, DictY);
RY2 = length(Y2) / length(Y)

X = X(1:1000);

X2 = LempelZivEnCo(X, Dict2);
R2 = length(X2) / length(X)

X3 = LempelZivDeco(X2, Dict2);

% The output of LempelZivDeco is a "string"
for i = 1:length(X3)
    X4(i) = str2num(X3(i))
end

BER = sum((X(1:length(X3)) - X4) .^ 2) %BER
```

```
Index exceeds the number of array elements. Index must not exceed 1.
```