Information theory Project

Part 1

Team 28

Name	ID
Mohammad Wael Monir	2001249
Omar Hossam Eldin Esmail	2000310
Ahmed Ibrahim Ali	2000267
Ahmed Gamal Sdeek Ahmed	2002048
Mohamed Gamal Eldin Abduljalil	2001078
Seif Ashraf Mohamed	2001549

Note: All the implementations of the non-built in functions are at the end of the Report.

1- Information Gain & Entropy

```
clc;
clear all;

filename=input('Enter the file name : ', 's');
symbols=fileread(filename);

[symbols,symbols_size,unique_symb,unique_symb_size,repeated_symbols_pro
b] = generating_data(symbols);
[information_gain,entropy] =
Information_gain_entropy(repeated_symbols_prob,unique_symb,unique_symb_
size)
```

```
Enter the file name : trial.txt

information_gain =

  2×79 cell array

  Columns 1 through 8

    {' '     }    {'e'     }    {'t'     }    {'i'     }    {'o'     }    {'r'     }    {'n'     }    {'a'     }
    {[2.4523]}    {[3.3733]}    {[3.9980]}    {[4.1583]}    {[4.1602]}    {[4.1789]}    {[4.2181]}    {[4.2424]}

  Columns 9 through 16

    {'s'     }    {'c'     }    {'d'     }    {'h'     }    {'l'     }    {'m'     }    {'↵'     }    {'p'     }
    {[4.3798]}    {[5.1076]}    {[5.2221]}    {[5.2928]}    {[5.3400]}    {[5.6076]}    {[5.8718]}    {[5.8908]}

  Columns 17 through 24

    {'u'     }    {'f'     }    {'g'     }    {'y'     }    {'.'     }    {','     }    {'b'     }    {'L'     }
    {[5.9374]}    {[6.0794]}    {[6.2753]}    {[6.4016]}    {[6.5841]}    {[6.8561]}    {[6.9484]}    {[7.0637]}

  Columns 25 through 32

    {'A'     }    {'w'     }    {'v'     }    {'P'     }    {'T'     }    {'S'     }    {'C'     }    {'M'     }
    {[7.1002]}    {[7.2781]}    {[7.3731]}    {[7.5945]}    {[7.8233]}    {[8.0328]}    {[8.0423]}    {[8.0661]}

  Columns 33 through 40

    {'k'     }    {'N'     }    {'I'     }    {'E'     }    {'O'     }    {'U'     }    {'D'     }    {'('     }
    {[8.3466]}    {[8.4246]}    {[8.5070]}    {[8.6511]}    {[8.6950]}    {[8.8111]}    {[8.8437]}    {[9.0855]}

  Columns 41 through 48

    {')'     }    {'1'     }    {'W'     }    {'R'     }    {'"'     }    {'2'     }    {'x'     }    {'3'     }
    {[9.0855]}    {[9.1250]}    {[9.2075]}    {[9.2396]}    {[9.2505]}    {[9.4001]}    {[9.4123]}    {[9.7251]}

  Columns 49 through 55

    {'''     }    {'F'      }    {'_'      }    {'4'      }    {'-'      }    {'q'      }    {'5'      }
    {[9.7714]}    {[10.1250]}    {[10.1656]}    {[10.1864]}    {[10.2288]}    {[10.2726]}    {[10.4123]}

  Columns 56 through 62

    {'H'      }    {'B'      }    {'j'      }    {'G'      }    {';'      }    {'0'      }    {'/'      }
    {[10.4369]}    {[10.5670]}    {[10.7100]}    {[10.8687]}    {[10.9374]}    {[11.2075]}    {[11.6511]}

  Columns 63 through 69

    {':'      }    {'9'      }    {'Y'      }    {'z'      }    {'6'      }    {'V'      }    {'7'      }
    {[11.7714]}    {[11.9026]}    {[11.9026]}    {[11.9026]}    {[12.0470]}    {[12.1250]}    {[12.2075]}

  Columns 70 through 76

    {'8'      }    {'+'      }    {'['      }    {']'      }    {'%'      }    {'J'      }    {'K'      }
    {[12.2075]}    {[13.7100]}    {[14.2949]}    {[14.2949]}    {[14.7100]}    {[14.7100]}    {[14.7100]}

  Columns 77 through 79

    {'X'      }    {'\'      }    {'Q'      }
    {[14.7100]}    {[15.2949]}    {[16.2949]}
```

```
entropy =

    4.5610
```

2- Shannon & Huffman Encoding/Decoding

```
[transmitted_data_shannon,key_shannon] =
Shannon_binary_encode(symbols,symbols_size,unique_symb,unique_symb_size
,repeated_symbols_prob);
[decoded_shannon,n_shannon] =
decode_algorithm(transmitted_data_shannon,key_shannon,unique_symb_size)
;

[transmitted_data_huffman,key_huffman] =
Huffman_encode(symbols,symbols_size,unique_symb,unique_symb_size,repeat
ed_symbols_prob);
[decoded_huffman,n_huffman] =
decode_algorithm(transmitted_data_huffman,key_huffman,unique_symb_size)
;
```

decoded_huffman	*1x80401 char*
decoded_shannon	*1x80401 char*
entropy	4.5610
filename	'trial.txt'
information_gain	*2x79 cell*
key_huffman	*79x2 cell*
key_shannon	*79x2 cell*
n_huffman	*1x79 double*
n_shannon	*1x79 double*
repeated_symbols_prob	*1x79 double*
symbols	*1x80401 char*
symbols_size	80401
transmitted_data_huffman	*1x369895 cell*
transmitted_data_shannon	*1x415162 cell*
unique_symb	*1x79 char*
unique_symb_size	79

3- Efficiency of Huffman & Shannon

```
efficiency_shannon =

   88.3282


efficiency_huffman =

   99.1376
```

4- Comparing the outputs of both codes to the input

```
if(strcmp(decoded_shannon,symbols)~=1)
    display("Error:Shannon Received data not equal input data");
else
    display("Shannon Received data equal input data");
end

if(strcmp(decoded_huffman,symbols)~=1)
    display("Error: Huffman Received data not equal input data");
else
   display("Huffman Received data equal input data");
```

```
"Shannon Received data equal input data"

"Huffman Received data equal input data"
```

- Functions Implementation

```
function
[symbols,symbols_size,unique_symb,unique_symb_size,repeated_symbols_prob] =
generating_data(symbols)

%% Extracting the symbols %%

unique_symb=unique(symbols);
[~,symbols_size]=size(symbols);
[~,unique_symb_size]=size(unique_symb);

%% Generating symbols frequancies %%

repeated_symbols=zeros(1,unique_symb_size);

for i=1:unique_symb_size

    for j=1:symbols_size

        if(unique_symb(i)==symbols(j))
        repeated_symbols(i)=repeated_symbols(i)+1;
        end
    end
end

%% Ordering the symbols and their probabilities in decending order %%

[repeated_symbols, idx] = sort(repeated_symbols,'descend');
unique_symb = unique_symb(idx);

repeated_symbols_prob=repeated_symbols./symbols_size;


end
```

```
function [information_gain_per_symbol,entropy] =
Information_gain_entropy(repeated_symbols_prob,unique_symb,unique_symb_size)

    information_gain=zeros(1,unique_symb_size);

    for i=1:unique_symb_size

    information_gain(i) = -log2(repeated_symbols_prob(i));

    end


    information_gain_per_symbol = [num2cell(unique_symb);
num2cell(information_gain)];

entropy = -sum(repeated_symbols_prob .* log2(repeated_symbols_prob));


end


function [transmitted_data,key] =
Shannon_binary_encode(symbols,symbols_size,unique_symb,unique_symb_size,repeat
ed_symbols_prob,n)

index_space = strfind(unique_symb, ' ');
index_enter = strfind(unique_symb, newline);

%Finding each alpha values


a(1)=0;

for j=2:unique_symb_size
    a(j)=a(j-1)+repeated_symbols_prob(j-1);
end

% Finding each code length

for i=1:unique_symb_size
    n(i)= ceil(-1*(log2(repeated_symbols_prob(i))));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Computing each code

z=[];

for i=1:unique_symb_size
    int=a(i);
for j=1:n(i)
    frac=int*2;
    c=floor(frac);
```

```
    frac=frac-c;
    z=[z c];
    habinaryStr = num2str(z); % Convert to string and add spaces
    encoded_binary{i} = strrep(habinaryStr, ' ', ''); % Remove the spaces
    % Encoded binary contains the Code value for each unique charechter
    int=frac;
end
z=[];
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Encoding the data of the txt file
for i=1:symbols_size

    for j=1:unique_symb_size

        if(unique_symb(j)==symbols(i))

            transmitted_data_shannon(i)=encoded_binary(j);

            break;

        end

    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Concatenating all the transmitted data

transmitted_data = []; % THe array which will hold the Binary
transmitted_data = {strjoin(transmitted_data_shannon, '')};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Splitting each Bit into its own cell

transmitted_data = cellstr(transmitted_data{1}');
transmitted_data = transmitted_data';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The KEYYYYYYYYYYYYYYYYYYYYYYYYYY

unique_symb = cellstr(unique_symb(:))';
key = [encoded_binary;unique_symb];
key = key';

key{index_space,2}=' ';
key{index_enter,2}=char(10);
end
```

```
function [transmitted_data,sortedCodes] =
Huffman_encode(symbols,symbols_size,unique_symb,unique_symb_size,repeated_symb
ols_prob)

%% Generating Huffman tree %%

fullTree = [(1:unique_symb_size)', repeated_symbols_prob',
zeros(unique_symb_size, 1), zeros(unique_symb_size, 1)];
    % fullTree columns: [index, probability, left_child, right_child]

    % Working array for merging
    tree = fullTree;

    % Building the Huffman tree
    while size(tree, 1) > 1

        % Sort the new tree with decending order arter each merging
        tree = sortrows(tree, -2);

        % Merge the two smallest nodes (last two rows)
        leftChild = tree(end-1, 1);   % Second smallest node
        rightChild = tree(end, 1);   % Smallest node
        newProb = tree(end-1, 2) + tree(end, 2);  % Combined probability

        % Add a new parent node to the fullTree
        newNodeIndex = max(fullTree(:, 1)) + 1; % Generate a new index
        fullTree = [fullTree; newNodeIndex, newProb, leftChild, rightChild];

        % Remove the last two rows and add the new parent node to the working
tree
        tree = tree(1:end-2, :);
        tree = [tree; newNodeIndex, newProb, leftChild, rightChild];
    end

    % Now the array (tree) has a single row which is the root of the Huffman
tree
    root = tree(end, :);




    codes = GenerateHuffmanCodes(fullTree, root(1), unique_symb, '');
    codes(:, [1, 2]) = codes(:, [2,1]);


%%
% Assuming 'Codes' is a cell array where each row is [binary_pattern, symbol]
% and 'unique_symb' is the array of unique symbols.
% Initialize a new cell array to store the sorted codes
sortedCodes = cell(unique_symb_size, 2);

% To make codes sorted according to their order in the unique_symb Array
```

```
% Loop through the unique symbols to match and sort the codes

for i = 1:unique_symb_size
    % Find the index of the symbol in 'Codes'
    idx = find(strcmp(unique_symb(i), codes(:, 2)), 1);

    % Add the corresponding code to the sortedCodes array

    sortedCodes{i, 1} = codes{idx, 1};  % Store the binary code
    sortedCodes{i, 2} = codes{idx, 2};  % Store the symbol
end

%%%%%


for i=1:symbols_size

    for j=1:unique_symb_size

        if(unique_symb(j)==symbols(i))
            transmitted_data_huffman(i)=sortedCodes(j,1);
            break;

        end

    end

end


% Concatenating all the transmitted data

transmitted_data = [];
transmitted_data = {strjoin(transmitted_data_huffman, '')};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Splitting each Bit into its own cell

transmitted_data = cellstr(transmitted_data{1}');

% Transposing the result
transmitted_data = transmitted_data';


%% Generating the codes %%


function codes = GenerateHuffmanCodes(fullTree, nodeIndex, unique_symb,
currentCode)
    % Look for the current node in the tree
    row = fullTree(fullTree(:, 1) == nodeIndex, :);

    % Check if it's a leaf node (no children)
```

```
    if row(3) == 0 && row(4) == 0

        % Leaf node: print symbol and code
        codes = {unique_symb(nodeIndex), currentCode};

    else
        % Internal node: go to its children (left and right)
        leftCodes = GenerateHuffmanCodes(fullTree, row(3), unique_symb,
strcat(currentCode, '0')); % Left

        rightCodes = GenerateHuffmanCodes(fullTree, row(4), unique_symb,
strcat(currentCode, '1')); % Right

        codes = [leftCodes; rightCodes];
    end
end

end
```

```
function [decoded,n] = decode_algorithm(transmitted_data,key,unique_symb_size)

% Encoded binary data

% Initialize decoding variables
decoded = '';
buffer = '';

% Decode iteratively by matching prefixes
for i = 1:length(transmitted_data)
    buffer = [buffer, transmitted_data{i}];  % Append next bit to the buffer

    % Check if the buffer matches any key
    idx = find(strcmp(buffer, key(:, 1)), 1);
    if ~isempty(idx)
        decoded = [decoded, key{idx, 2}];  % Append the matched symbol
        buffer = '';  % Reset buffer for the next symbol
    end
end

% getting the nuumber of character that represents each symbol
for i = 1:unique_symb_size
    n(i) = strlength(key(i,1));
end


end
```

```matlab
function [efficiency] =
Efficiency_calc(repeated_symbols_prob,n,unique_symb_size,entropy)

ACL = 0;
for i = 1:unique_symb_size
    ACL = ACL + n(i)*repeated_symbols_prob(i);
end

% Calculating the entropy

efficiency = (entropy / ACL) *100;

end
```