ECE F344 - Information Theory and Coding Assignment 1

Group 22

Kushaal Tummala	2018AAPS0422H
Sai Vamshi Kamaraju	2018AAPS0420H
Padharthi Sai Sridhar	2018AAPS0472H
Sushant Kunchala	2018AAPS0411H
V Abhinav Sai Venkat	2018AAPS0451H

Question:

Write a computer program to generate a table of size (2k + 1) × (2k + 1), which shows that sum (XOR) of each pair of codeword results in another valid codeword. All the entries in this table will be codewords. Carry out this exercise for the (7, 4) Hamming code.

THEORY:

To prevent loss of data during transmission due to the noise we try to add redundant bits to the data in a specific manner which allows us to decode the message correctly even with erasures and errors. LInear Block code is an error control coding scheme. The properties of a linear block code:

- The sum of any two codewords should be a codeword belonging to the corresponding code.
- The code should always include the all-zero codeword.
- The minimum hamming distance between two codewords of a linear code is equal to the minimum weight of any non-zero codeword ⟹ $d^*=w^*$

Hamming codes are part of linear block codes with the property:

$$(n,k) = (2^m-1, 2^m-1-m) \text{ where m is any positive integer.}$$

All linear block codes can be represented as generator matrices. When you multiply the message word with the generator matrix you get the codewords.

$c=iG$

We can verify the received codeword using a parity check matrix.

$cH^T=0$

The dimensions of G and H are (k x n) and ((n-k) x n). This theory is implemented in our code.

- We take n and k input from the user and create an object of the “fec_hamming” class.
- We print the generator matrix of the (n,k) hamming code. Then we create an array of decimals that contains all possible messages(0 to 2^k-1) and then convert them into binary words of fixed length k bits.
- Then we encode all the messages by multiplying them with the generator matrix to get the hamming codewords.
- We generated a table of size (2k + 1) × (2k + 1) which consists of all the possible combinations of sum(XOR) of any two codewords.
- Now we verify the three properties of an LBC. All-zero codeword in the code.
- Then we check if all the words in the table are codewords by comparing them with the original valid codewords or with the parity check matrix.
- Then we find the minimum hamming distance and the minimum weight of a non-zero codeword and if they both are equal.
- While checking with the parity check matrix we might get multiples of 2 instead of zero, this is because we are not doing mod 2 addition or multiplication.

CODE:

The assignment was coded in python in a notebook environment. Please use the link below or the attached notebook

Link to the colab notebook:
(https://colab.research.google.com/drive/1MPFL61mhS-24ACDxZzn5FLbxU3rOd3Dh?usp=sharing)

1. Install and import all necessary libraries

```
pip install scikit-dsp-comm
```

```
Collecting scikit-dsp-comm
  Downloading https://files.pythonhosted.org/packages/8b/6d/72a4d6fed1537fc5dbd547e31a40e96b3a93d07e8ec844f0aa866e3758b6/scikit_dsp_comm-1.2.0-py3-none-any.whl (116kB)
     |████████████████████████████████| 122kB 4.2MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from scikit-dsp-comm) (1.4.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from scikit-dsp-comm) (3.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from scikit-dsp-comm) (1.18.5)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->scikit-dsp-comm) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->scikit-dsp-comm) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib->scikit-dsp-comm) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->scikit-dsp-comm) (1.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.1->matplotlib->scikit-dsp-comm) (1.15.0)
Installing collected packages: scikit-dsp-comm
Successfully installed scikit-dsp-comm-1.2.0
```

[2]
```
import sk_dsp_comm.fec_block as block
import numpy as np
import pandas as pd
```

2. Read n, k values, and instantiate an instance of the fec_hamming class.

```
[3]
n = int(input("Enter n: \t")) #Input the n,k values for a hamming code
k = int(input("Enter k: \t"))
parity = n-k
```

```
Enter n:        7
Enter k:        4
```

```
[4]
hh1 = block.fec_hamming(parity) #Instantiating a fec_hamming class instance(hh1)
```

3. Print out the values of n, k, Parity check matrix, and the generator matrix of the hamming code and create a list of all possible messages.

```
print(f"n: {hh1.n} \nk: {hh1.k}\nGenerator Matrix:\n{hh1.G}\nParity Check Matrix:\n{hh1.H}") #n,k and generator matrix of the hamming code
```

```
n: 7
k: 4
Generator Matrix:
[[1 0 0 0 1 1 1]
 [0 1 0 0 1 1 0]
 [0 0 1 0 0 1 1]
 [0 0 0 1 1 0 1]]
Parity Check Matrix:
[[1 1 0 1 1 0 0]
 [1 1 1 0 0 1 0]
 [1 0 1 1 0 0 1]]
```

```
msg_len = (2**hh1.k) - 1
```

```
[7]
msg_array = [x for x in range(msg_len+1)]
print(f"Message array(Decimal Form): {msg_array}") #Creating a decimal message array
```

```
Message array(Decimal Form): [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

4. Define the required functions. dToBi returns a fixed-length binary equivalent of a decimal number. tostring concatenates all elements of an array into a string. ham encodes a given message and returns an array of the encoded message.

```
[8]
def dToBi(n):
  '''Returns a fixed length binary bit array for a given message'''
  return bin(n).replace("0b","").zfill(hh1.k)
def tostring(aray):
  '''Converts a bit array to a string'''
  string = ''
  aray = aray.tolist()
  for i in range(len(aray)):
    string = string + str(aray[i])
  return string
```

```
def ham(msg):
  '''Returns a hamming encoded message for a given input message'''
  x = np.array([int(a) for a in dToBi(msg)])
  y = hh1.hamm_encoder(x)
  y = np.array([int(a) for a in y])
  return y
```

5. Iteratively encode the messages using the ham function. Add (XOR) all possible codewords pairwise and create a table of the results. tostring is used to convert the codewords from array to a string.

```python
code = [] #Empty list to hold codewords
for i in msg_array:
  code.append(ham(i))
code = np.array(np.concatenate(code, axis=0)) #Concatenate the codeword(list to string)
code.resize((2**hh1.k,hh1.n)) #Resize the codeword list
msg_len = (2**hh1.k)-1
Table1 = [] #Table of added codewords
Table = []
for i in range(msg_len+1):
  for j in range(msg_len+1):
    Table.append(tostring(code[i,:]^code[j,:]))
    Table1.append((code[i,:]^code[j,:]))
```

```python
[11]
code1 = [] #container to hold the original codewords in a string form
m,n = np.shape(code)
for i in range(m):
  code1.append(tostring(code[i,:]))
print(f"Codewords:\n{code1}") #All possible codewords
```

```
Codewords:
['0000000', '0001101', '0010011', '0011110', '0100110', '0101011', '0110101', '0111000', '1000111', '1001010', '1010100', '1011001', '1100001', '1101100', '1110010', '1111111']
```

6. Create a square matrix of the added codewords.

```python
Table_grid = np.array(Table) #Array to print a table
Table_grid.resize((2**hh1.k,2**hh1.k)) # Resize the table to (2^k + 1)x(2^k + 1)
```

```python
[13]
np.shape(Table_grid)
```

```
(16, 16)
```

```python
[14]
Table_grid = pd.DataFrame(Table_grid, index = code1, columns = code1)
print("Table:\n")
display(Table_grid)
```

7. Code to check if the all-zero codeword is part of the codewords generated.

Property 1: The all zero codeword is always a codeword.

```python
[15]
'0000000' in code1
```

8. Iteratively check if all elements in the “Table” are in the codewords set.

```python
[16]
#Check if all elements in 'Table' are valid codewords(elements in 'code')
bol = True
for i in range(len(Table)):
  if Table[i] in code1:
    print("Resultant codeword is valid")
  else:
    print("Not Valid")
    bol = False
if bol == True:
  print("\nAll codewords are valid.Hence proved.")
```

Iteratively check if all the codewords are valid by multiplying with the parity check matrix and verifying the product is zero.

```python
#Parity check implemented for all the codewords in the Table
b = []
for i in range(len(Table1)):
  b.append(np.dot(Table1[i],np.transpose(hh1.H)))
for i in range(len(b)):
  for j in range(3):
    if b[i][j] % 2 ==0:
      b[i][j] = 0
print(b)
#Checking to see if all elements are zero
st = False
for i in range(len(b)):
  for j in range(3):
    if b[i][j] != 0:
      st = True
    break
if st == False:
  print("All codewords are valid")
```

9. Iteratively calculate distances between all possible unique codeword pairs and find the minimum distance. Iteratively calculate the weights of all codewords (Weights can be formulated as hamming distances between the codewords and the all-zero codeword) and find the minimum codeword. Check if the minimum weight is equal to the minimum hamming distance.

Property 3: The minimum Hamming distance between two codewords of a linear block code is equal to the minimum Hammingweight of any non-zero codeword, i.e., d* = w*.

```python
code_xor = []
for i in range(msg_len+1):
  for j in range(msg_len+1):
    if i == j:
      continue
    code_xor.append(np.count_nonzero(code[i,:]^code[j,:]))
min_dist = np.min(code_xor) #Minimum hamming distance
weights = []
for i in range(1, msg_len+1):
  weights.append(np.count_nonzero(code[i,:]^code[0,:])) #Excluded all zeros as weight will be zero
min_weight = np.min(weights)

min_weight == min_dist
```

OUTPUTS:

Table:

Table:

	0000000	0001101	0010011	0011110	0100110	0101011	0110101	0111000	1000111	1001010	1010100	1011001	1100001	1101100	1110010	1111111
0000000	0000000	0001101	0010011	0011110	0100110	0101011	0110101	0111000	1000111	1001010	1010100	1011001	1100001	1101100	1110010	1111111
0001101	0001101	0000000	0011110	0010011	0101011	0100110	0111000	0110101	1001010	1000111	1011001	1010100	1101100	1100001	1111111	1110010
0010011	0010011	0011110	0000000	0001101	0110101	0111000	0100110	0101011	1010100	1011001	1000111	1001010	1110010	1111111	1100001	1101100
0011110	0011110	0010011	0001101	0000000	0111000	0110101	0101011	0100110	1011001	1010100	1001010	1000111	1111111	1110010	1101100	1100001
0100110	0100110	0101011	0110101	0111000	0000000	0001101	0010011	0011110	1100001	1101100	1110010	1111111	1000111	1001010	1010100	1011001
0101011	0101011	0100110	0111000	0110101	0001101	0000000	0011110	0010011	1101100	1100001	1111111	1110010	1001010	1000111	1011001	1010100
0110101	0110101	0111000	0100110	0101011	0010011	0011110	0000000	0001101	1110010	1111111	1100001	1101100	1010100	1011001	1000111	1001010
0111000	0111000	0110101	0101011	0100110	0011110	0010011	0001101	0000000	1111111	1110010	1101100	1100001	1011001	1010100	1001010	1000111
1000111	1000111	1001010	1010100	1011001	1100001	1101100	1110010	1111111	0000000	0001101	0010011	0011110	0100110	0101011	0110101	0111000
1001010	1001010	1000111	1011001	1010100	1101100	1100001	1111111	1110010	0001101	0000000	0011110	0010011	0101011	0100110	0111000	0110101
1010100	1010100	1011001	1000111	1001010	1110010	1111111	1100001	1101100	0010011	0011110	0000000	0001101	0110101	0111000	0100110	0101011
1011001	1011001	1010100	1001010	1000111	1111111	1110010	1101100	1100001	0011110	0010011	0001101	0000000	0111000	0110101	0101011	0100110
1100001	1100001	1101100	1110010	1111111	1000111	1001010	1010100	1011001	0100110	0101011	0110101	0111000	0000000	0001101	0010011	0011110
1101100	1101100	1100001	1111111	1110010	1001010	1000111	1011001	1010100	0101011	0100110	0111000	0110101	0001101	0000000	0011110	0010011
1110010	1110010	1111111	1100001	1101100	1010100	1011001	1000111	1001010	0110101	0111000	0100110	0101011	0010011	0011110	0000000	0001101
1111111	1111111	1110010	1101100	1100001	1011001	1010100	1001010	1000111	0111000	0110101	0101011	0100110	0011110	0010011	0001101	0000000

Property 1:

True implies that the all-zero codeword is in the generated codeword.

```
1 '0000000' in code1
True
```

Property 2:

“Resultant codeword is valid” is printed for each codeword in “Table” if it is in the list of all possible codewords. “All codewords are valid. Hence proved.” is printed after all codewords are checked and all are valid.

```
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid
Resultant codeword is valid

All codewords are valid.Hence proved.
```

```
if st == False:
  print("All codewords are valid")
```

```
[array([0, 0, 0]), array([0, 0, 0]), array([0, 0, 0]), array([0, 0, 0]), array([0, 0,
All codewords are valid
```

Property 3:

True implies that the minimum weight is equal to the minimum hamming distance of the code.

```
min_weight == min_dist
```

```
True
```