Hand in problem 3, Information Theory

Joel Bångdal (dat15jba)
Viktor Claesson (dat15vcl)

Kurs
EITN45
Lund Universitet

May 19, 2020

$$G(1,3) = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Message	Codeword	Weight
0000	0000 0000	0
0001	0000 1111	4
0010	0011 0011	4
0011	0011 1100	4
0100	0101 0101	4
0101	0101 1010	4
0110	0110 0110	4
0111	0110 1001	4
1000	1010 1010	4
1001	1010 0101	4
1010	1001 1001	4
1011	1001 0110	4
1100	1111 1111	8
1101	1111 0000	4
1110	1100 1100	4
1111	1100 0011	4

Table 1: All messages and the ruslting codewords when ran through the generator

$d_{min} = 4$
$detect = 3 = d_{min} - 1$
$correct = 1.5 = \frac{d_{min}}{2}$
We can detect any error up to 3 bits and we can correct any 1 bit change.

Error	Syndrom
0000 0000	0000
1000 0000	1000
0100 0000	0100
0010 0000	1010
0001 0000	0110
0000 1000	1001
0000 0100	0101
0000 0010	1011
0000 0001	0111

Table 2: Error and their corresponding syndrome

$codeword = 00110011$
$error_{codeword} = 01110011$
$syndrom = 0100$
$error_{codeword} - error = 01110011 - 01000000 = 00110011 = codeword$

```
import numpy as np


def G(r, m):
    if r == 0:
        return np.ones((1, 2**m))  # 2^m 1's
    elif r == m:
        return np.identity(2**m)  # unit matrix of
            size 2^m
    else:
        M = G(r, m-1)
        N = G(r-1, m-1)
        z = np.zeros(N.shape)
        upper = np.concatenate((M, M), axis=1)
        lower = np.concatenate((z, N), axis=1)
        return np.concatenate((upper, lower), axis=0)


def bin_array(num, m):
    """Convert a positive integer num into an m-bit
       bit vector"""
    return np.array(list(np.binary_repr(num).zfill(m))
       ).astype(np.int8)


def error_vec(idx, m):
    err = np.zeros((1, m))
    err[0][idx] = 1
    return err


def binify_vector(vec):
    newVec = np.zeros(vec.shape)
    for (idx, val) in enumerate(vec):
        newVec[idx] = (val % 2)
    return newVec


def binify_matrix(mat):
    newMat = np.zeros(mat.shape)
```

```
    for (idx, vec) in enumerate(mat):
        newMat[idx] = binify_vector(vec)
    return newMat


G = G(1, 3)
G_t = G.transpose()
for vec in [binify_vector(bin_array(i, 4).dot(G)) for
    i in range(16)]:
    # print(vec)
    continue


verify = binify_matrix(G.dot(G_t))
# print(verify)


for vec in [error_vec(i, 8) for i in range(8)]:
    # print(vec.dot(G_t))
    continue


print(binify_vector(np.array([0, 1, 1, 1, 0, 0, 1, 1])
    .dot(G_t)))
```