L2_Coding test_Sol

March 7, 2025

0.1 Question 1: Convert Ciphertext to Binary Vector

0.1.1 Problem Statement

The given ciphertext: gzvorps$qmr!#fhbhyzo

- Each character corresponds to a 5-bit binary representation.
- The total length of the binary vector is: $5 \times 20 = 100$
- Convert the ciphertext into its binary form and store it as a vector y.

[147]:
```
import itertools
import numpy as np

def create_encoding_dict():
    # Generate all elements of Z_2^5
    Z2_5 = list(itertools.product([0, 1], repeat=5))

    # Define characters (A-Z) and six special characters
    characters = [chr(97 + i) for i in range(26)] + ['@', '#', '$', '&', '*', '!
↪']

    # Create dictionary mapping characters to Z_2^5 vectors
    return {characters[i]: Z2_5[i] for i in range(32)}

def encode_text(text, encoding_dict):
    text = text.lower()   # Convert to uppercase for consistency
    encoded_vectors = []

    for char in text:
        if char in encoding_dict:
            encoded_vectors.extend(encoding_dict[char])

    return encoded_vectors

def format_as_column_vector(encoded_vectors):
    return "\n".join(map(str, encoded_vectors))

if __name__ == "__main__":
    encoding_dict = create_encoding_dict()
```

```
text = "gzvorps$qmr!#fhbhyzo"
encoded_vectors = encode_text(text, encoding_dict)
formatted_output = format_as_column_vector(encoded_vectors)
y=np.matrix(formatted_output).reshape(-1,1)
```

0.2 Question 2: Determine the Length of the Message

0.2.1 Problem Statement

- We are given that the encryption follows the equation: $y = Ax$
- The matrix **$ A $ has dimensions** $ 100 ×55 $.
- The unknown vector $ x $ represents the bit-string of the original message.

0.2.2 Solution

- Since $ A $ maps a 55-dimensional vector to a 100-dimensional vector, x must have 55 elements.
- As each character in the message corresponds to 5 bits, the length of the original message is: $\frac{55}{5} = 11$ characters

0.3 Question 3: Define the Vector Spaces V and W

0.3.1 Problem Statement

- Define the vector spaces V and W, along with their respective fields F_V and F_W.

0.3.2 Solution

We define the following: - Domain (Input Space):
$V = \{0,1\}^{55} = \{x \mid x \text{ is a 55-dimensional binary vector}\}$ - Codomain (Output Space):
$W = \{0,1\}^{100} = \{y \mid y \text{ is a 100-dimensional binary vector}\}$ - Field over which operations are performed:
$F_V = F_W = \{0,1\}$ where arithmetic follows modulo 2 operations.

0.4 Question 4: Define Addition and Multiplication Rules

0.4.1 Problem Statement

Define the addition and multiplication rules for: - The vector spaces V and W - The fields F_V and F_W

0.4.2 Solution

Since we are working in binary space {0,1}: - Addition:
$u \oplus v = (u_1 + v_1, u_2 + v_2, ..., u_n + v_n) \mod (2)$. This is equivalent to bitwise XOR. - Multiplication (Scalar-Vector Product):
$\alpha * v = (\alpha v_1, \alpha v_2, ..., \alpha v_n) \mod (2)$, where $ {0,1} $.

For the fields F_V and F_W, the operations are defined as: - $\alpha + \beta = (\alpha + \beta) \mod (2)$ - $\alpha \cdot \beta = (\alpha * \beta) \mod (2)$

0.5 Question 5: Suggest Bases for V and W

0.5.1 Problem Statement

- Identify bases for the vector spaces V and W.

0.5.2 Solution

- The standard basis for $ V $ is: $b = \{e_1, e_2, \cdots, e_{55}\}, \quad \text{where} b_i = e_i, 1 \leq i \leq 55$
- Similarly, the standard basis for W is: $b' = \{e_1, e_2, ..., e_{100}\}, \quad \text{where} b'_j = e_j, 1 \leq j \leq 100$
- e_i is the unit vector with a 1 at the i^{th} position and 0s elsewhere.

0.6 Question 6: Compute $\dim(V)$ and $\dim(W)$ and Compute $T(b_{17})$

0.6.1 Solution

- The dimension of V is: $\dim(V) = 55$
- The dimension of W is: $\dim(W) = 100$
- Load matrix key A from the file.
- Since matrix A multiplication represents a linear transformation, $T(b_{17})$ is simply the 17th column of (A). From the below code it is verified.

[161]:
```
b_17=np.eye(55,dtype=int)[:,16] #b_17 is the usual basis of vector space␣
↪{0,1}^55
T_b_17=A*sp.Matrix(b_17) # T(b_17)=A(b_17), that is the 17th column of matrix A
print(T_b_17==A[:,16])
```

```
True
```

0.7 Question 7: Construct the Matrix A_L

0.7.1 Solution

To compute A_L: If $A^T A$ is invertible, then: $A_L = (A^T A)^{-1} A^T$. Below is the code to find the matrix A_L

[196]:
```
A_L=(((A.T*A).inv_mod(2))*(A.T))%2 # All operations are under modulo 2
```

0.8 Question 8: Decode the Message

0.8.1 Problem Statement

- Compute x and convert it into a string of characters using the dictionary. Print the decoded message.

0.8.2 Solution

Compute: $x = A_L y$

[200]:
```
x=(A_L*y)%2 #operations under modulo 2
```

- Convert each 5-bit segment of x to its corresponding character using the given dictionary.

[203]:
```python
def create_encoding_dict():
    # Generate all elements of Z_2^5
    Z2_5 = list(itertools.product([0, 1], repeat=5))

    # Define characters (A-Z) and six special characters
    characters = [chr(97 + i) for i in range(26)] + ['@', '#', '$', '&', '*', '!
↪']

    # Create dictionary mapping characters to Z_2^5 vectors
    return {characters[i]: Z2_5[i] for i in range(32)}, {Z2_5[i]: characters[i]␣
↪for i in range(32)}

def encode_text(text, encoding_dict):
    text = text.lower()  # Convert to uppercase for consistency
    encoded_vectors = []

    for char in text:
        if char in encoding_dict:
            encoded_vectors.extend(encoding_dict[char])

    return encoded_vectors

def decode_vector(encoded_vector, decoding_dict):
    decoded_text = ""
    for i in range(0, len(encoded_vector), 5):
        chunk = tuple(encoded_vector[i:i+5])
        if chunk in decoding_dict:
            decoded_text += decoding_dict[chunk]
    return decoded_text

def format_as_column_vector(encoded_vectors):
    return "\n".join(map(str, encoded_vectors))

if __name__ == "__main__":
    encoding_dict, decoding_dict = create_encoding_dict()


    # Decoding example
    input_vector = np.array(x).flatten().tolist()
    decoded_text = decode_vector(input_vector, decoding_dict)
    print("Decoded Text:")
    print(decoded_text)
```

```
Decoded Text:
iloveyouall
```

0.9 Question 9: Properties of an Effective Matrix Key A

0.9.1 Problem Statement

- What properties must A have to ensure effective decoding?

0.9.2 Solution

For A to work effectively: - It must have full column rank (i.e., rank 55).

0.10 Question 10: Repeating decoding procedure over ternary field

0.10.1 Problem Statement

- In this case re-design the matrix key $A_L(A_{L_{new}})$ based on the new matrix key (A_{new}) provided to extract the message text from the ciphertext "uvdundjharndifbwavla"
- Encode the given text "uvdundjharndifbwavla" to the ternary string and stored in a vector y_{new}

[174]:
```
import numpy as np
import itertools
import sympy as sp

# Define the field Z_3
Z3 = [0, 1, 2]

# Define three linearly independent basis vectors in Z_3^5
v1 = np.array([1, 0, 0])
v2 = np.array([0, 0, 1])
v3 = np.array([0, 1, 0])

# Generate all possible linear combinations of the basis vectors
def generate_codes():
    codes = []
    for c1, c2, c3 in itertools.product(Z3, repeat=3):
        code = (c1 * v1 + c2 * v2 + c3 * v3) % 3  # Compute in Z_3
        codes.append(tuple(code))
    return codes

# Generate 27 unique codes
codes = generate_codes()

# Assign characters to codes
characters = list("abcdefghijklmnopqrstuvwxyz#")
char_to_code = {char: code for char, code in zip(characters, codes)}
#print(char_to_code)
# Function to encode text into a ternary column vector
def encode_to_ternary(text):
    text = text.lower()
```

```
    encoded_vectors = []
    for char in text:
        if char in char_to_code:
            encoded_vectors.extend(char_to_code[char])
    return np.array(encoded_vectors).reshape(-1, 1)  # Column vector

# Example usage
text = "uvdundjharndifbwavla"
y_new = encode_to_ternary(text)
#y_new
```

- Load the given matrix key A_{new} from the file
- Designing the matrix key $A_{L_{new}} = ((A_{new})^T(A_{new}))^{-1} * (A_{new})^T$

[178]:
```
A_L_new=(((A_new.T*A_new).inv_mod(3))*A_new.T)%3 # All vector and field␣
↪operations are under modulo 3
```

- Compute x_{new} and convert it into a string of characters using the dictionary. Print the decoded message.

[180]:
```
x_new=(A_L_new*y_new)%3

# Define the field Z_3
Z3 = [0, 1, 2]

# Define three linearly independent basis vectors in Z_3^5
v1 = np.array([1, 0, 0])
v2 = np.array([0, 0, 1])
v3 = np.array([0, 1, 0])

# Generate all possible linear combinations of the basis vectors
def generate_codes():
    codes = []
    for c1, c2, c3 in itertools.product(Z3, repeat=3):
        code = (c1 * v1 + c2 * v2 + c3 * v3) % 3  # Compute in Z_3
        codes.append(tuple(code))
    return codes

# Generate 27 unique codes
codes = generate_codes()

# Assign characters to codes
characters = list("abcdefghijklmnopqrstuvwxyz#")
char_to_code = {char: code for char, code in zip(characters, codes)}
code_to_char = {code: char for char, code in char_to_code.items()}

# Function to encode text into a ternary column vector
```

```python
def encode_to_ternary(text):
    text = text.lower()
    encoded_vectors = []
    for char in text:
        if char in char_to_code:
            encoded_vectors.extend(char_to_code[char])
    return np.array(encoded_vectors).reshape(-1, 1)  # Column vector
# Function to decode a ternary column vector back to text
def decode_from_ternary(ternary_vector):
    decoded_text = ""
    vector_list = ternary_vector.flatten().tolist()  # Convert to list
    for i in range(0, len(vector_list), 3):
        code_tuple = tuple(vector_list[i:i+3])
        if code_tuple in code_to_char:
            decoded_text += code_to_char[code_tuple]
    return decoded_text
Original_text = decode_from_ternary(np.array(x_new))
print("Original Text:", Original_text)
```

```
Original Text: iloveyouall
```

```
[ ]:
```