ABIN - Assignment 1

Group-16

Question 1

Importing necessary libraries

```
from itertools import combinations
```

Functions

Function to generate mutations

```
def generate_mutations(input_string):
    mutations = []
    n = len(input_string)

    #all possible mutations with 0 substitution
    mutations.append(input_string)
    zero = len(mutations)
    print('No. of strings with 0 mutations: ',zero)

    #all possible mutations with 1 substitution
    for i in range(n):
        for base in 'ACGT':
            if input_string[i] != base:
                mutation = input_string[:i] + base + input_string[i+1:]
                mutations.append(mutation)
    one = len(mutations) - zero
    print('No. of strings with 1 mutations: ',one)

    #all possible mutations with 2 substitutions
    for i, j in combinations(range(n), 2):
        for base1 in 'ACGT':
            for base2 in 'ACGT':
                if input_string[i] != base1 and input_string[j] != base2:
                    mutation = input_string[:i] + base1 + input_string[i+1:j] + base2 + input_string[j+1:]
                    mutations.append(mutation)

    two = len(mutations) - one - zero
    print('No. of strings with 2 mutations: ',two)

    return mutations
```

Function to find consensus string

```
def consensus_string(strings):
    length = len(strings[0])
    consensus = ''

    for i in range(length):
        bases = [s[i] for s in strings]                                    #forms a list of all bases found at ea
        base_counts = {base: bases.count(base) for base in 'ACGT'}         #counts the number of individual base
        consensus += max(base_counts, key=base_counts.get)                 #adds the base with most count in cons
    return consensus
```

Function to find hamming distance

```

```

```
def hamming_distance(string_original, string_consensus):
    return sum(c1 != c2 for c1, c2 in zip(string_original, string_consensus))                #checks for differenc
```

Driver code

```
input_string = "ATGCTTGCAT"
mutations = generate_mutations(input_string)
consensus = consensus_string(mutations)
distance = hamming_distance(input_string, consensus)
print()
print("Original String:", input_string)
print("Consensus String:", consensus)
print("Hamming Distance:", distance)
```

```
No. of strings with 0 mutations:  1
No. of strings with 1 mutations:  30
No. of strings with 2 mutations:  405

Original String: ATGCTTGCAT
Consensus String: ATGCTTGCAT
Hamming Distance: 0
```

Formulas

Let the length of string $= n$

- Strings with 0 mutations $= 1$
- Strings with 1 mutations $= 3 \cdot n$
- Strings with 2 mutations $= \binom{n}{2} \cdot 3^2$

Question 2

Import the fastq.gz file which is a zipped fastq file as store it as its accession number

```
import requests

url = "https://www.be-md.ncbi.nlm.nih.gov/Traces/sra-reads-be/fastq?acc=SRR1780095"
fastq_gz_file = "SRR1780095.fastq.gz"

response = requests.get(url, stream=True)
if response.status_code >= 200 and response.status_code <= 299:
    with open(fastq_gz_file, 'wb') as f:
        for chunk in response.iter_content(chunk_size=1024):
            if chunk:
                f.write(chunk)
    print(f"Downloaded {url} to {fastq_gz_file}")
else:
    print(f"Failed to download {url}, status code: {response.status_code}")
```

```
Downloaded https://www.be-md.ncbi.nlm.nih.gov/Traces/sra-reads-be/fastq?acc=SRR1780095 to SRR1780095.fastq.gz
```

Unzip the fastq.gz file to get a fastq file which can be traversed using SeqIO libarary from BIO

```
import gzip
import shutil

fastq_file = "SRR1780095.fastq"

if fastq_gz_file.endswith(".gz"):
    with gzip.open(fastq_gz_file, 'rb') as f_in, open(fastq_gz_file[:-3], 'wb') as f_out:
```

```
        shutil.copyfileobj(f_in, f_out)
    print(f"Decompressed {fastq_gz_file} to {fastq_file}")
```

```
Decompressed SRR1780095.fastq.gz to SRR1780095.fastq
```

get the Bio package from pip installer

```
!pip install Bio
```

```
Requirement already satisfied: Bio in /usr/local/lib/python3.10/dist-packages (1.5.9)
Requirement already satisfied: biopython>=1.80 in /usr/local/lib/python3.10/dist-packages (from Bio) (1.81)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from Bio) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from Bio) (4.66.1)
Requirement already satisfied: mygene in /usr/local/lib/python3.10/dist-packages (from Bio) (3.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from Bio) (1.5.3)
Requirement already satisfied: pooch in /usr/local/lib/python3.10/dist-packages (from Bio) (1.7.0)
Requirement already satisfied: gprofiler-official in /usr/local/lib/python3.10/dist-packages (from Bio) (1.0.
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from biopython>=1.80->Bio) (
Requirement already satisfied: biothings-client>=0.2.6 in /usr/local/lib/python3.10/dist-packages (from myger
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->Bio) (20
Requirement already satisfied: platformdirs>=2.5.0 in /usr/local/lib/python3.10/dist-packages (from pooch->Bi
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from pooch->Bio) (
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requ
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->Bio) (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8
```

Install plotly for plotting, pandas to convert nucleotide count at a position into a dataframe and the Bio file to read the the fastq file using SeqIO

```
import plotly.express as px
import pandas as pd
from Bio import SeqIO

sequences = []

#parse the fastq file and store all the sequences in the sequences list
for record in SeqIO.parse(fastq_file, 'fastq'):
    sequences.append(str(record.seq))

# Calculate the length of the reads (assuming all reads have the same length)
read_length = len(sequences[0])
total_sequences = len(sequences)

sequences[:5]
```

```
['TGGCTGTTTGGTGCAAGAGGCCTAGCTTTCGGCCTATCTTGGCTTTTGACATGTGTTCTTCACTAAGCTTAGTCAT',
 'AATGAAATTAGGCCAAGTAATAACCGAACAATGGTCTTTAAGTGATTAACTGAAAGAAAGAATCACATGTTTCTCA',
 'GAAACAGAGGGGGACGGGGCAGGCGCGCTGGGCCCGCCCCCCGTGCCTGGGGCAGCTTCTCATTGGTGAACCCTCC',
 'GGACCAGTCCCGCCGCCTCCGCCGTCCGCACCCGGCCCGCAGCTCTTAAACGCGCCCCGCCCCCTCCCGGGGCCGG',
 'ACCTCAACCTAGGCCTCCTATTTATTCTAGCCACCTCTAGCCTAGCCGTTTACTCAATCCTCTGATCAGGGTGAGC']
```

Print read length and number of sequences

```
print(f'Filename           : {fastq_file}')
print(f'Read Length        : {read_length}')
print(f'Number of reads    : {total_sequences}')
```

```
Filename           : SRR1780095.fastq
Read Length        : 76
Number of reads    : 141590
```

get occurrence count of every nucleotide(including ambiguous N) at each position across the sequences

```
#create a dictnary where every nucleotide has an array of frequency counts at each of the positions in the sequenc
nucleotide_counts = {nucleotide: [0] * read_length for nucleotide in 'ATCGN'}
for sequence in sequences:
    for i, nucleotide in enumerate(sequence):
```

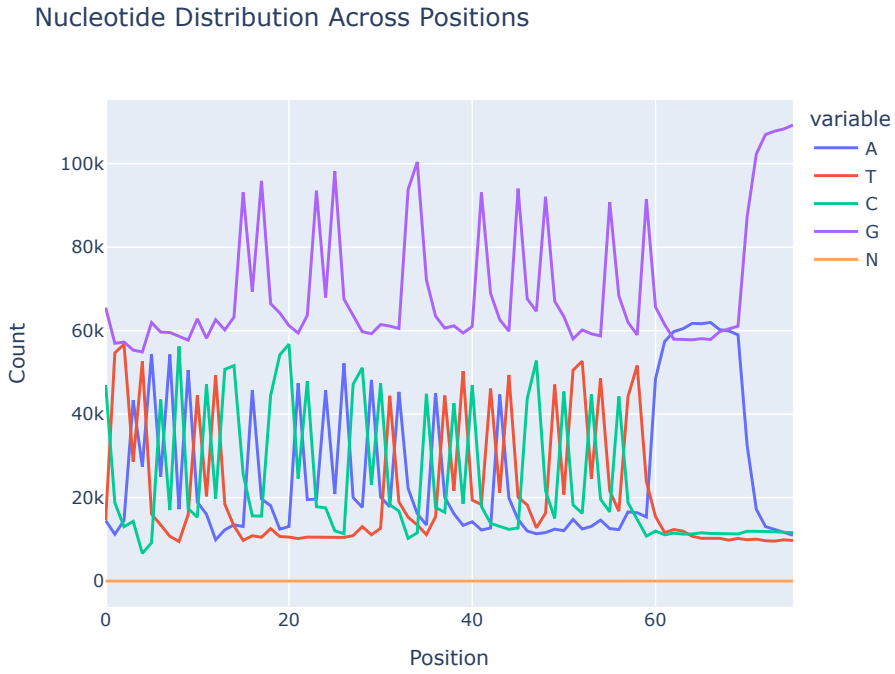
```
        nucleotide_counts[nucleotide][i] += 1

#convert the dictonary into dataframe for plotting and visualising
df = pd.DataFrame(nucleotide_counts)
df
```

	A	T	C	G	N
0	14492	14602	46987	65505	4
1	11219	54655	18751	56962	3
2	14540	56755	13018	57277	0
3	43322	28618	14311	55337	2
4	27407	52651	6642	54890	0
...	...	...	...	...	...
71	17212	10053	11992	102330	3
72	13051	9660	11887	106990	2
73	12386	9580	11817	107800	7
74	11658	9885	11718	108322	7
75	10998	9724	11601	109261	6

76 rows × 5 columns

Plot the count versus position graph

```
#plotting using plotly library
fig = px.line(df, x= df.index, y=list(df.columns), labels={'index': 'Position', 'value': 'Count'}, title='Nucleoti
fig.show()
```



Graph Interpretation:

- The sequences on average have higher concentration of nucleotide A
- Nucleotide G, C and T apppear to have a similar frequency of occurance
- Though insignificant compared to other nucleotides, there are some trace positions where no nucleotide could be recorded probably due to the drawbacks of the sequencing equipment.

Question 3

Import necessary libraries

```
import random
import numpy as np
```

Supporting functions

Function to generate random sequences with of a specified length

```
def generate_random_sequence(length):
    return ''.join(random.choice('ACGT') for _ in range(length))
generate_random_sequence(10)
```

```
'TATTGTGAGC'
```

Function to introduce specified number of mutations in a given sequence

```
def introduce_mutations(sequence, num_mutations):
    sequence = list(sequence)
    for _ in range(num_mutations):
        position = random.randint(0, len(sequence) - 1)
        mutation = random.choice('ACGT')
        sequence[position] = mutation
    return ''.join(sequence)
introduce_mutations("AAAAAAAAAAAAAAAAA",3)
```

```
'ATAAAAGAGAAAAAAAAA'
```

Function to plant a motif sequence in a parent sequence at a given position

```
def plant_motif(sequence, motif, position):
    return sequence[:position] + motif + sequence[position + len(motif):]
plant_motif("GGGGGGGGGGG","MOTIF", 5)
```

```
'GGGGGMOTIFGG'
```

The gibbs sampling function

- Implements the gibbs sampler until the same consensus sequence is reached for 10 consecutive iterations
- Prints profile matrix and consesus sequence motif for each iteration

```
def gibbs_sampling(seqs, motif_length, seed, ITR = 5000):
    """
    Perform Gibbs sampling to identify motif locations and consensus motif in a set of sequences.

    Args:
        seqs (list of str): List of DNA sequences.
        motif_length (int): Length of the motif to be identified.
        seed (int): Random seed for reproducibility.
        ITR (int): Maximum number of iterations to perform Gibbs sampling (default: 5000).

    Returns:
        tuple: A tuple containing the motif locations and consensus motif.

    Note:
        The function uses random.seed(seed) and np.random.seed(seed) for reproducibility.

    """

    # Get the number of sequences
```

```
t = len(seqs)

 # Initialize motif locations randomly within each sequence
motif_locations = [random.randint(0, len(seq) - motif_length) for seq in seqs]

def calculate_profile_except_i(i,itr):
    """
    Calculate the profile matrix except for the i-th sequence.

    Args:
        i (int): Index of the sequence to exclude.
        itr (int): Current iteration for printing.

    Returns:
        numpy.ndarray: Profile matrix.

    """

    # Initialize the profile matrix with zeros
    profile = np.zeros((4, motif_length))

    # Iterate through all sequences except the i-th one
    for j, seq in enumerate(seqs):
        if j != i:
            for k in range(motif_length):
                nt = seq[motif_locations[j] + k]
                if nt in "ACGT":
                    nt_index = "ACGT".index(nt)
                    profile[nt_index][k] += 1
    def print_profile(profile):
        print("-"*75)
        print(f"\nIteration :{itr}\n")
        nucleotides = "ACGT"
        print('Profile Matrix: ')
        for nt_index, nt in enumerate(nucleotides):
            print(f"{nt}:", end=" ")
            for k in range(motif_length):
                print(f"{profile[nt_index][k]:.4f}", end=" ")
            print()
     # Assign small probabilities instead of 0
    profile = (profile + 1) / ((t - 1) + 1)

    # Print the profile matrix for the current iteration
    print_profile(profile)
    return profile

# Initialize variables for tracking convergence
old_mot = ""
counter = 0
counter2 = 1

# Main Gibbs sampling loop
while(counter < 10 and counter2 < ITR):

    # Randomly select an index for a sequence
    i = random.randint(0, t - 1)

    # Calculate the profile matrix except for the i-th sequence
    profile_except_i = calculate_profile_except_i(i, counter2)

    # Get the sequence for the selected index
    seq = seqs[i]

    # Initialize a list to store motif location probabilities
    motif_location_probabilities = []

    # Iterate through all possible motif positions in the sequence
    for j in range(len(seq) - motif_length + 1):
        lmer = seq[j:j + motif_length]
        probability = 1.0
        for k in range(motif_length):
            nt_index = "ACGT".index(lmer[k])
            probability *= profile_except_i[nt_index][k]
        motif_location_probabilities.append(probability)
```

```
        # Convert motif location probabilities to a NumPy array
        motif_location_probabilities = np.array(motif_location_probabilities)

        # Normalize probabilities to sum to 1
        motif_location_probabilities /= motif_location_probabilities.sum()

         # Select a new motif location based on probabilities
        new_location = np.random.choice(range(len(seq) - motif_length + 1), p=motif_location_probabilities)
        new_mot = seq[new_location:new_location + 10]
        motif_locations[i] = new_location
        counter2 += 1

        # Print old and new motifs for tracking convergence
        print()
        print(f"Old Motif: {old_mot}")
        print(f"New Motif: {new_mot}")

        # Check if the motif remains the same for 10 consecutive iterations
        if(old_mot == new_mot):
            counter += 1
        else:
            counter = 0
            old_mot = new_mot

    # final consensus motif predicted using gibbs sampling
    consensus_motif = new_mot

    return motif_locations, consensus_motif
```

Driver Code

- Generates 100 1KB sequences and plants a motif specified by MOTIF variable randomly in each of the 100 sequences and stores the positions for future comparision
- Calls the gibbs sampling function and eventually prints the consensus Motif identified by the gibbs sampler.

```
num_sequences = 100
sequence_length = 1024
motif_length = 10
num_mutations_list = [0, 1, 2]
MOTIF = 'AAAAAAAAAA'

sequences = []
motif_positions = []

# Loop to generate random sequences with mutations and planted motifs
for _ in range(num_sequences):
    # Generate a random DNA sequence of the specified length
    seq = generate_random_sequence(sequence_length)

    # Randomly choose the number of mutations to introduce
    num_mutations = random.choice(num_mutations_list)

    # Introduce mutations into the sequence
    seq = introduce_mutations(seq, num_mutations)

    # Randomly choose a position to plant the motif within the sequence
    motif_position = random.randint(0, sequence_length - motif_length)

    # Plant the motif into the sequence at the chosen position
    seq = plant_motif(seq, MOTIF, motif_position)

    # Append the generated sequence to the list of sequences
    sequences.append(seq)

    # Store the motif position for reference
    motif_positions.append(motif_position)

# Use Gibbs sampling to identify motif locations and consensus motif
motif_locations, consensus_motif = gibbs_sampling(sequences, motif_length, 42)

# Print the identified motif locations and consensus motif
print("-"*75)
```

```
print(f"\nConsensus Motif:{consensus_motif}")
print()
print("Planted Motif Locations:")
print(motif_positions)
print()
print("Consensus Motif Location:")
print(motif_locations)
```

```
New Motif: AAAAAAAAAA
--------------------------------------------------------------------------------

Iteration :4034

Profile Matrix:
A: 0.8200 0.9500 0.8500 0.8600 0.9100 0.8800 0.9100 0.8600 0.8300 0.8000
C: 0.0500 0.0100 0.0800 0.1100 0.0200 0.0300 0.0100 0.0300 0.0300 0.0700
G: 0.0600 0.0100 0.0200 0.0500 0.0800 0.0300 0.0100 0.1000 0.1600 0.1200
T: 0.1000 0.0600 0.0800 0.0100 0.0200 0.0900 0.1000 0.0400 0.0100 0.0400

Old Motif: AAAAAAAAAA
New Motif: AAAAAAAAAA
--------------------------------------------------------------------------------

Iteration :4035

Profile Matrix:
A: 0.8200 0.9500 0.8500 0.8600 0.9100 0.8800 0.9100 0.8600 0.8300 0.8000
C: 0.0500 0.0100 0.0800 0.1100 0.0200 0.0300 0.0100 0.0300 0.0300 0.0700
G: 0.0600 0.0100 0.0200 0.0500 0.0800 0.0300 0.0100 0.1000 0.1600 0.1200
T: 0.1000 0.0600 0.0800 0.0100 0.0200 0.0900 0.1000 0.0400 0.0100 0.0400

Old Motif: AAAAAAAAAA
New Motif: AAAAAAAAAA
--------------------------------------------------------------------------------

Iteration :4036

Profile Matrix:
A: 0.8200 0.9500 0.8500 0.8600 0.9100 0.8800 0.9100 0.8600 0.8300 0.8100
C: 0.0500 0.0100 0.0800 0.1100 0.0200 0.0300 0.0100 0.0300 0.0300 0.0600
G: 0.0600 0.0100 0.0200 0.0500 0.0800 0.0300 0.0100 0.1000 0.1600 0.1200
T: 0.1000 0.0600 0.0800 0.0100 0.0200 0.0900 0.1000 0.0400 0.0100 0.0400

Old Motif: AAAAAAAAAA
New Motif: AAAAAAAAAA
--------------------------------------------------------------------------------

Iteration :4037

Profile Matrix:
A: 0.8200 0.9500 0.8500 0.8600 0.9100 0.8800 0.9100 0.8600 0.8300 0.8100
C: 0.0500 0.0100 0.0800 0.1100 0.0200 0.0300 0.0100 0.0300 0.0300 0.0600
G: 0.0600 0.0100 0.0200 0.0500 0.0800 0.0300 0.0100 0.1000 0.1600 0.1200
T: 0.1000 0.0600 0.0800 0.0100 0.0200 0.0900 0.1000 0.0400 0.0100 0.0400

Old Motif: AAAAAAAAAA
New Motif: AAAAAAAAAA
--------------------------------------------------------------------------------

Consensus Motif:AAAAAAAAAA

Planted Motif Locations:
[550, 263, 731, 668, 946, 354, 324, 462, 607, 959, 692, 246, 71, 959, 397, 316, 90, 988, 530, 975, 699, 121,

Consensus Motif Location:
[550, 263, 731, 669, 948, 354, 322, 462, 607, 959, 693, 246, 71, 958, 397, 316, 90, 776, 527, 952, 700, 120,
```