Discovering and Evaluating hidden motifs in Sequence

❓ For your second exercise you will need to implement the Gibbs Sampler for motif discovery in the language of your choice.

A dataset of 50 short sequences is given may be found here. A short motif has been embedded in some of these sequences which you will have to discover through a Gibbs sampling approach.

What you need to do:

Part 1

```
1. Implement the Gibbs Sampler in Python
2. Use a free parameter for the size k of the motif you will be searching f
or
3. Repeat the search for various values of k (e.g. from k=3 to k=7)
```

Result 1: Return the discovered motifs as PWMs

Part 2

`1. Calculate the Information Content of the discovered PWMs for each k`

Result 2: Report the k and the motif with the greatest (scaled) Information Content

Code

```
#STEPS

#1st: Build all kmers for every seq
#2nd: Choose a random kmer for each seq
#3rd: Built a PWM with the random kmers
#4th: Use the PWM to scan each seq
#5th: Keep the ones with better score
#6th: Each time find the I
#7th: Stop when I=threshold (I=1.8) or repeat 3-5
#8th: Return the final PWM
```

```
########### READ THE SEQUENCES AS LISTS ####################

# Open the file and read its contents
with open('motifs_in_sequence.fa', 'r') as file:
    contents = file.read()

# Split the contents into individual sequences
sequences_contents = contents.split()

# Create an empty list to store the sequences
sequences = []

# Append each sequence to the list
for sequence in sequences_contents:
    sequences.append(sequence) #sequences contains all the sequences as strings


######### BUILD ALL KMERS FOR EVERY SEQ & CHOOSE A RANDOM KMER FOR EACH SEQ ##########
#####3

import random

def all_kmers(sequences, k):
    all_motifs=[]
    for seq in sequences:
        motifs = []
        for i in range(len(seq)-k+1):
            motif = seq[i:i+k]
            motifs.append(motif)
        all_motifs.append(motifs)
    return all_motifs

random_kmers = [item for sublist in [random.choices(i) for i in all_kmers(sequences,
 3)] for item in sublist]


############ BUILD A PWM WITH THE RANDOM KMERS ###########

def pwm(sequences):
    nuc = ['A', 'C', 'G', 'T']
    profile=[[0 for i in range(len(sequences[0]))] for j in range(len(nuc))]

    for instance in sequences:
        for j in range(len(instance)):
            residue=instance[j]
            profile[nuc.index(residue)][j]+=1
            profile[nuc.index(residue)][j]=float(profile[nuc.index(residue)][j])
    import numpy as np
    pwm = np.array(profile)
    pwm = pwm/len(sequences)
    return(pwm)

mypwm=pwm(random_kmers)
print(mypwm)


########### SCORE THE PWM ##################
```

```
def score_kmers(kmers, pwm):
    nucleotides= ['A', 'C', 'G', 'T']
    scores = []
    for kmer in kmers:
        score = 0
        for i in range(len(kmer)):
            nucleotide = kmer[i]
            score += pwm[nucleotides.index(nucleotide), i]
        scores.append(score)
    return scores
#print(score_kmers(random_kmers,mypwm))


######### INFORMATION CONTENT ###################

def pwmEntropyInformation(pwm):

    import numpy as np
    k = pwm.shape[1]

    information = np.zeros([1,k]) #computing the information of each position
    for i in range(k):
        information[0,i] = 2-abs(sum([elem*np.log2(elem) for elem in pwm[:,i] if elem
 > 0]))

    sumInfo = np.sum(information)
    scaledSumInfo = sumInfo/k

    return(information, sumInfo, scaledSumInfo

############ BEST RANDOMS ############
I= 0
threshold = 1.2
while I < threshold:

    new_random_kmers = [item for sublist in [random.choices(i) for i in all_kmers(sequ
ences, 7)] for item in sublist]
    new_mypwm=pwm(new_random_kmers)
    new_scores_of_random_kmers = score_kmers(new_random_kmers,new_mypwm)

    for i in range(len(random_kmers)):
        if new_scores_of_random_kmers[i] > scores_of_random_kmers[i]:
            random_kmers[i] = new_random_kmers[i]

    final_pwm = pwm(random_kmers)

    I = pwmEntropyInformation(final_pwm)[1]

print (random_kmers)
```

Description of the code - Comments

1. create all possible kmers for a certain k (i.e. k = 7) for every sequence

2. take a random kmer for every sequence and store them in a list called `random_kmers`
3. built a pwm with the random kmers and score them
4. continue to create a random list of kmers and score them. Keep every kmer that has a higher score than the previous random of that sequence
5. for every loop of scoring and storing kmers calculate the information content, using the `pwmEntropyInformation` function.
6. stop the looping when I > threshold (~1.8)

The output was supposed to be a list of 50 kmers (one for every sequence) that have the max score. Hypothetically, the kmers would have been similar and the motifs would be discovered.

The whole code would then be in a for loop in order to repeat the search from k=3 to k=7.

However, for an unknown reason, the information content was never higher than 1.2 and therefore the motifs were not discovered.