Algorithms for Bioinformatics - Exercise 2

Mourouzidou Eleni

In Gibbs Sampler Algorithm, we are looking for k-length motifs that appear in the input sequences.

As we sample a new k-mer we replace the motifs. We define the number k as the length of the motifs we are searching for, and use it to slide towards the sequence to cover its whole length.

In this implementation of Gibbs Sampler, the code reads 50 DNA sequences of the given file ("`motifs_in_sequence.fa`") and searches for motifs of length k that are more likely to be overrepresented in a sequence.

This function below, takes a filename input, initializes an empty list called 'seq' and appends to it each sequence that the file contains (separated by tab) as an element.
The list that it returns, contains all the sequences given by the file.

```
import random

#define a function that will take th filename as an input and return a list of
all the sequences in it
def load_sequence(filename):
   with open(filename) as f:
       #initialize an empty list
       seq = []
       #append every sequence in this list without \n
       for l in f:
           seq.append(l.strip())
   return seq
```

The gibbs_sampler function below, takes two inputs, the sequence list (sequence) and an integer (k) to define the length of the motifs that we would like to search.
We first initialize an empty list called 'motifs' and we then choose a random starting index point to get a random kmer for each sequence.

```
#get a random integer in a range from the start of the string (0 index) to
the last possible index that could be used as starting point to have a
kmer

start = random.randint(0, len(seq)-k)
```

After extracting the random indexes for each sequence we retrieve the k-length substrings based on them, and append them to the motifs list.
In the next step, we need to create the profile matrix. We initialize a dictionary with keys the 4 nucleotides and values, to start with 1.
We initialize the values with 1 instead of zeros as a pseudo-observation in order to calculate later the probabilities more effectively by avoiding the sensitivity to smaller variations. It would also be impossible to calculate the probabilities later, since we would have to multiply by 0.
The frequencies of all nucleotides at the ‘pos’ position are counted across all motifs.
Then, we iterate over all the strings of the sequence list to count the scores, after having it initialized to 0. A variable best_motif is also initialized in order to store the k-mers.
For each k-mer, a score is calculated as a product of the probabilities of each nucleotide of the k-mer.
For each position j, in the k-mer we divide the frequency of this nucleotide by the total number of motifs and then we multiply all these to obtain the final k-mer score.
The final goal is to find the best scoring motif of length k for each sequence.

```
def gibbs_sampler_pwm(sequences, k):
    motifs = []
    for seq in sequences:
        start = random.randint(0, len(seq)-k)
        motif = seq[start:start+k]
        motifs.append(motif)

    best_motifs = motifs.copy()
    while True:
        prof_matrix = []
        for pos in range(k):
            freq = {'A':1, 'C':1, 'T':1, 'G':1}
            for m in range(len(motifs)):
                freq[motifs[m][pos]] +=1
            prof_matrix.append(freq)

        pwm = []
        for freq_dict in prof_matrix:
            total_freq = sum(freq_dict.values())
            pwm.append({base: freq_dict[base] / total_freq for base in 'ACGT'})

        new_motifs = []
        for s in sequences:
            score = 0
            best_motif = ''
            for i in range(len(seq) +1-k):
                kmer = seq[i:i+k]
                kmer_score = 1
```

```
                for j in range(k):
                    kmer_score *= pwm[j][kmer[j]]
                if kmer_score > score:
                    score = kmer_score
                    best_motif = kmer
            new_motifs.append(best_motif)

        new_unique_cols = sum([len(set(col)) for col in zip(*new_motifs)])
        best_unique_cols = sum([len(set(col)) for col in zip(*best_motifs)])

        if new_unique_cols > best_unique_cols:
            best_motifs = new_motifs.copy()
            prof_matrix = []
            for pos in range(k):
                freq = {'A':1, 'C':1, 'T':1, 'G':1}
                for m in range(len(best_motifs)):
                    freq[best_motifs[m][pos]] +=1
                prof_matrix.append(freq)

            pwm = []
            for freq_dict in prof_matrix:
                total_freq = sum(freq_dict.values())
                pwm.append({base: freq_dict[base] / total_freq for base in
'ACGT'})
        if best_motifs == motifs:
            return pwm
        motifs = new_motifs.copy()


k_values = range(3, 8)

for k in k_values:
   print(gibbs_sampler(load_sequence('motifs_in_sequence.fa'), k))
```

OUTPUT

[{'A': 0.25925925925925924, 'C': 0.2037037037037037, 'G': 0.2962962962962963, 'T': 0.24074074074074073},
{'A': 0.18518518518518517, 'C': 0.25925925925925924, 'G': 0.3148148148148148, 'T': 0.24074074074074073},
{'A': 0.2962962962962963, 'C': 0.2222222222222222, 'G': 0.2222222222222222, 'T': 0.25925925925925924}]

[{'A': 0.18518518518518517, 'C': 0.2037037037037037, 'G': 0.2222222222222222, 'T': 0.3888888888888889},
{'A': 0.25925925925925924, 'C': 0.24074074074074073, 'G': 0.18518518518518517, 'T': 0.3148148148148148},
{'A': 0.3333333333333333, 'C': 0.2222222222222222, 'G': 0.2222222222222222, 'T':

0.2222222222222222},
{'A': 0.3148148148148148, 'C': 0.2777777777777778, 'G': 0.14814814814814814, 'T': 0.25925925925925924}]

[{'A': 0.2222222222222222, 'C': 0.3333333333333333, 'G': 0.2222222222222222, 'T': 0.2222222222222222},
{'A': 0.25925925925925924, 'C': 0.24074074074074073, 'G': 0.18518518518518517, 'T': 0.3148148148148148},
{'A': 0.25925925925925924, 'C': 0.3333333333333333, 'G': 0.18518518518518517, 'T': 0.2222222222222222},
{'A': 0.14814814814814814, 'C': 0.24074074074074073, 'G': 0.2962962962962963, 'T': 0.3148148148148148},
{'A': 0.4074074074074074, 'C': 0.18518518518518517, 'G': 0.18518518518518517, 'T': 0.2222222222222222}]

[{'A': 0.3148148148148148, 'C': 0.18518518518518517, 'G': 0.2962962962962963, 'T': 0.2037037037037037},
{'A': 0.25925925925925924, 'C': 0.25925925925925924, 'G': 0.2222222222222222, 'T': 0.25925925925925924},
{'A': 0.2037037037037037, 'C': 0.16666666666666666, 'G': 0.37037037037037035, 'T': 0.25925925925925924},
{'A': 0.24074074074074073, 'C': 0.3333333333333333, 'G': 0.12962962962962962, 'T': 0.2962962962962963},
{'A': 0.25925925925925924, 'C': 0.2777777777777778, 'G': 0.18518518518518517, 'T': 0.2777777777777778},
{'A': 0.24074074074074073, 'C': 0.2777777777777778, 'G': 0.25925925925925924, 'T': 0.2222222222222222}]

[{'A': 0.24074074074074073, 'C': 0.18518518518518517, 'G': 0.2222222222222222, 'T': 0.35185185185185186},
{'A': 0.2962962962962963, 'C': 0.2037037037037037, 'G': 0.25925925925925924, 'T': 0.24074074074074073},
{'A': 0.24074074074074073, 'C': 0.24074074074074073, 'G': 0.2222222222222222, 'T': 0.2962962962962963},
{'A': 0.3333333333333333, 'C': 0.18518518518518517, 'G': 0.25925925925925924, 'T': 0.2222222222222222},
{'A': 0.2037037037037037, 'C': 0.24074074074074073, 'G': 0.35185185185185186, 'T': 0.2037037037037037},
{'A': 0.24074074074074073, 'C': 0.25925925925925924, 'G': 0.3148148148148148, 'T': 0.18518518518518517},
{'A': 0.16666666666666666, 'C': 0.35185185185185186, 'G': 0.16666666666666666, 'T': 0.3148148148148148}
{'A': 0.2037037037037037, 'C': 0.2222222222222222, 'G': 0.24074074074074073, 'T': 0.3333333333333333}]