DNA Motif Search

Felipe colombelli

Implementação

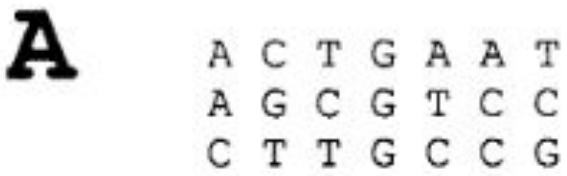
- Baseada no trabalho:
  Identification of consensus patterns in unaligned DNA sequences known to be functionally related
  Gerald Z. Hert et. al. (1990)

- Python

- VS Code

Funcionamento do algoritmo

- Considere as sequências em A
- e motivos de tamanho 6



Funcionamento do algoritmo

A	C	T	G	A	A	T
A	G	C	G	T	C	C
C	T	T	G	C	C	G

- Para a primeira sequência, os motivos possíveis são dados por B

B

	A	C	T	G	A	A
A	1	0	0	0	1	1
C	0	1	0	0	0	0
G	0	0	0	1	0	0
T	0	0	1	0	0	0

I = 12.0

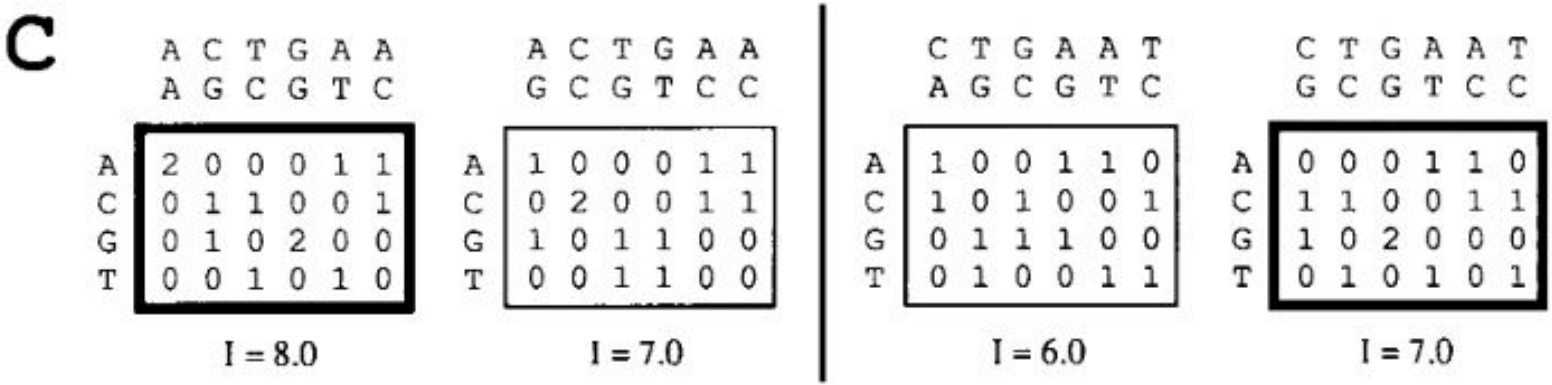
	C	T	G	A	A	T
A	0	0	0	1	1	0
C	1	0	0	0	0	0
G	0	0	1	0	0	0
T	0	1	0	0	0	1

I = 12.0

Funcionamento do algoritmo

```
A C T G A A T
A G C G T C C
C T T G C C G
```

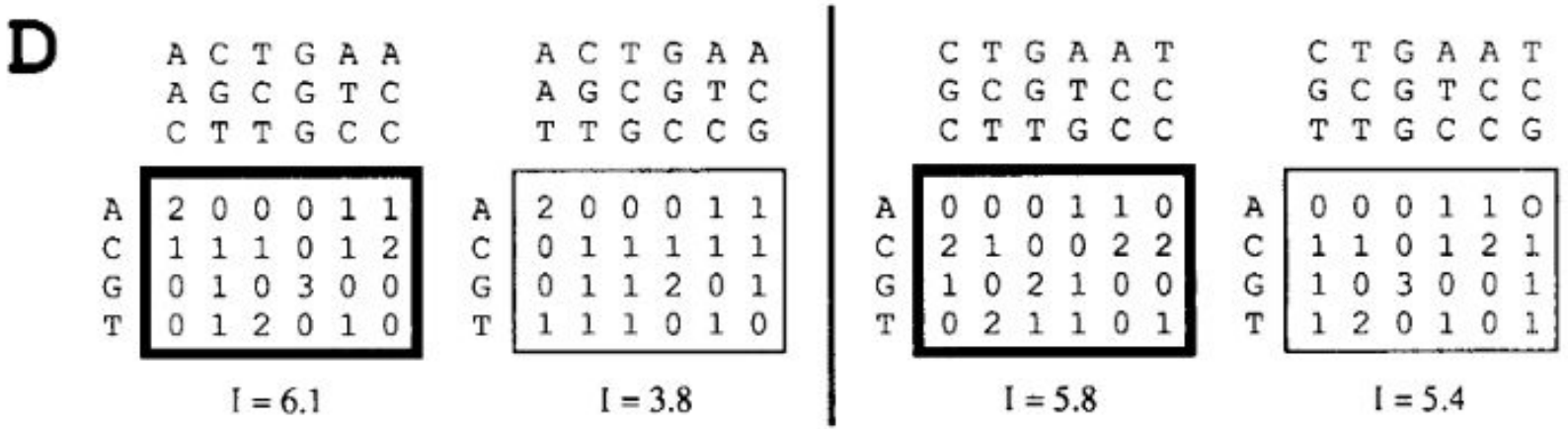
- Motivos possíveis considerando a segunda sequência
- Apenas os com maiores I serão levados adiante



Funcionamento do algoritmo

```
A C T G A A T
A G C G T C C
C T T G C C G
```

- Motivos possíveis considerando a terceira sequência



Funcionamento do algoritmo

A	C	T	G	A	A	T
A	G	C	G	T	C	C
C	T	T	G	C	C	G

- Resultado

E

	A	C	T	G	A	A
	A	G	C	G	T	C
	C	T	T	G	C	C
A	2	0	0	0	1	1
C	1	1	1	0	1	2
G	0	1	0	3	0	0
T	0	1	2	0	1	0

$I = 6.1$

Funcionamento do algoritmo

- Cálculo do termo I

$$I = \sum_{i=1}^{L} \sum_{b=A}^{T} \frac{N_{bi}}{N} \log_2 \frac{N_{bi}/N}{P_b}$$

Investigação do cálculo do termo I

E

	A	C	T	G	A	A
	A	G	C	G	T	C
	C	T	T	G	C	C
A	2	0	0	0	1	1
C	1	1	1	0	1	2
G	0	1	0	3	0	0
T	0	1	2	0	1	0

I = 6.1

```
[3]  PA = 4/18
     PC = 6/18
     PG = 4/18
     PT = 4/18

     (2/3) * log2((2/3) / PA) + \
     (1/3) * log2((1/3) / PA) + \
     (1/3) * log2((1/3) / PA) + \
     (1/3) * log2((1/3) / PC) + \
     (1/3) * log2((1/3) / PC) + \
     (1/3) * log2((1/3) / PC) + \
     (1/3) * log2((1/3) / PC) + \
     (2/3) * log2((2/3) / PC) + \
     (1/3) * log2((1/3) / PG) + \
     (3/3) * log2((3/3) / PG) + \
     (1/3) * log2((1/3) / PT) + \
     (1/3) * log2((1/3) / PT) + \
     (2/3) * log2((2/3) / PT)

     5.92481250360578
```

Implementação - find_motifs

```
40  def find_motifs(sequences, motif_len):
41
42      matrices =  build_first_matrices(sequences[0], motif_len)
43      for sequence in sequences[1:]:
44          matrices =  update_matrices(sequence, matrices, motif_len)
45
46      best_matrix = select_matrix(matrices)
47      return best_matrix
```

build_first_matrices

```
52  def build_first_matrices(sequence, motif_len):
53
54      matrices = []
55      num_matrices = len(sequence) - motif_len + 1
56
57      for i in range(num_matrices):
58          l_mer = get_l_mer_by_iteration(sequence, motif_len, i)
59          matrix = build_matrix([l_mer])
60          matrices.append(matrix)
61
62      return matrices
```

build_matrix

```
73  def build_matrix(l_mers):
74
75      num_cols = len(l_mers[0])
76      matrix = []
77
78      for base in ['a','c','g','t']:
79          row = []
80          for position in range(num_cols):
81              matches = 0
82              for l_mer in l_mers:
83                  if l_mer[position] == base:
84                      matches += 1
85              row.append(matches)
86          matrix.append(row)
87
88      I = compute_I(matrix, l_mers)
89
90      return (l_mers, matrix, I)
```

compute_I

```
93  def compute_I(matrix, l_mers):
94
95      num_sequences = len(l_mers)
96      pa = get_genomic_frequency(l_mers, 'a')
97      pc = get_genomic_frequency(l_mers, 'c')
98      pg = get_genomic_frequency(l_mers, 'g')
99      pt = get_genomic_frequency(l_mers, 't')
100     frequencies = [pa, pc, pg, pt]
101
102     I = 0
103
104     for idx, row in enumerate(matrix):
105         for col in row:
106             N_bi = col
107             if N_bi == 0:
108                 continue
109
110             I += (N_bi / num_sequences) * \
111                 log2((N_bi / num_sequences) / frequencies[idx])
112
113     return I
```

Resultados

- Demo!

Thanks!