Processamento da Informação

Google Meet - 03

http://professor.ufabc.edu.br/~jesus.mena/courses/pi-1q-2020/PI-google-meet/

Tipo de dado composto?

Vimos vários tipos de dados como: int, float

Stings são qualitativamente diferentes dos outros tipos pois são compostas de pedaços menores (caracteres): **Tipo de dado composto**.

```
>>> x = "mensagem"
>>> print( x )
mensagem
```

Strings

String

Uma string (cadeia) é uma sequência de caracteres.

Podemos acessar aos caracteres com o operador colchete

```
>>> fruta = "banana"
>>> letra = fruta[1]
```

Uma string é uma sequência...

A primeira letra (“b”) tem a posição 0.

A segunda letra (“a”) tem a posição 1, ...

```
>>> fruta = "banana"

>>> print( fruta[0] )
 b
>>> print( fruta[1] )
 a
>>> print( fruta[2] )
 n
```

Índices

Uma string é uma sequência...

A primeira letra (“b”) tem a posição 0.

A segunda letra (“a”) tem a posição 1, ...

```
>>> fruta = "banana"

>>> print( fruta[0] )
 b
>>> print( fruta[1] )
 a
>>> print( fruta[2] )
 n
>>> print fruta[1.5]
TypeError: string indices must be integers
```

Índices

No **índice**:
Podemos usar qualquer expressão, incluindo variáveis e operadores,
Entretanto, o valor do índice deve ser inteiro.

Comprimento

A função **len** retorna o número de caracteres de uma string.

```
>>> fruta = "banana"
>>> len(fruta)
6
```

Comprimento

Para pegar a última letra de uma string, podemos tentar realizar a seguinte operação:

```
>>> comprimento = len(fruta)
>>> ultima = fruta[comprimento]
IndexError: string index out of range
```

Índice fora do intervalo

Comprimento

Para pegar a última letra de uma string, podemos tentar realizar a seguinte operação:

```
>>> comprimento = len(fruta)
>>> ultima = fruta[comprimento - 1]
```

Percorrendo uma string com um laço

Várias operações envolvem processamento de strings, considerando um caractere de cada vez.

Exemplo para percorrer e imprimir cada letra da variável fruta:

```
indice = 0
while indice < len(fruta):
    letra = fruta[indice]
    print (letra)
    indice = indice + 1
```

Percorrendo uma string com um laço

Várias operações envolvem processamento de strings, considerando um caractere de cada vez.

Exemplo para percorrer e imprimir cada letra da variável fruta:

```
indice = 0
while indice < len(fruta):
    letra = fruta[indice]
    print (letra)
    indice = indice + 1
```

Se fruta = “banana”

Então o resultado será:

b
a
n
a
n
a

Percorrendo uma string com um laço

Imprimindo de trás para frente

```
indice = len(fruta)-1

while indice >= 0 :
    letra = fruta[indice]
    print (letra)
    indice = indice - 1
```

Se fruta = “banana”

Então o resultado será:
a
n
a
n
a
b

Percorrendo uma string com um laço

Também podemos usar o laço **for** para percorrer uma string.

```
for letra in fruta:
    print (letra)
```

Em cada iteração, o próximo caractere da string é atribuído à variável **letra**. O laço continua até que não reste mas caracteres.

Percorrendo uma string com um laço

Exemplo de concatenação de strings para a geração de uma série abecedário:

```
prefixos = "RSTUVW"
sufixo = "oma"

for letra in prefixos:
    print (letra + sufixo)
```

O símbolo + concatena

Percorrendo uma string com um laço

Exemplo de concatenação de strings para a geração de uma série abecedário:

```
prefixos = "RSTUVW"
sufixo = "oma"

for letra in prefixos:
    print (letra + sufixo)
```

```
Roma
Soma
Toma
Uoma
Voma
Woma
```

Fatias de strings

Um segmento de uma string é chamado de uma fatia (subsequência).

Selecionar uma fatia é similar a selecionar um caractere.

```
>>> s = "Pedro, Paulo e Maria"

>>> print (s[0:5])
Pedro
>>> print (s[7:12])
Paulo
>>> print (s[16:21])
Maria
```

O que faz a seguinte função?

```
def find(cadeia, x):
    i = 0
    while i < len(cadeia):
        if cadeia[i] == x:
            return i
        i = i + 1
    return -1
```

O que faz a seguinte função?

```
def find(cadeia, x):
    i = 0
    while i < len(cadeia):
        if cadeia[i] == x:
            return i
        i = i + 1
    return -1
```

A função procura o índice
do **caractere** na **cadeia**.
Se o caractere não é encontrado,
a função retorna -1.

Exercício

Reverso

Crie uma função que receba duas palavras e retorne **True** se uma das palavras é o reverso da outra:

```
def reverso(s1: str, s2: str) -> bool:
```

Exemplo:

'pots' é reverso de 'stop'

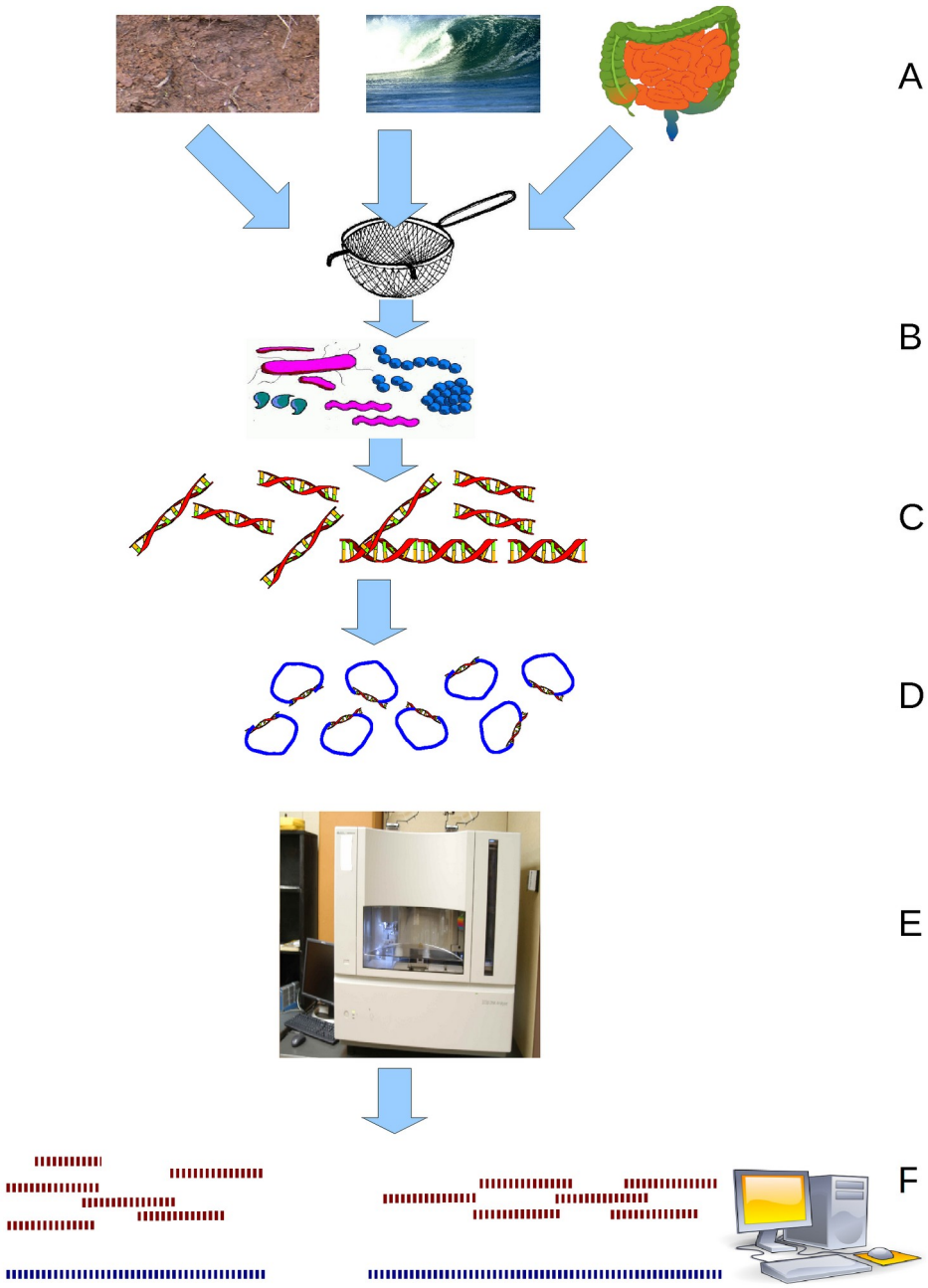
'livres' é reverso de 'servil'

Reverso

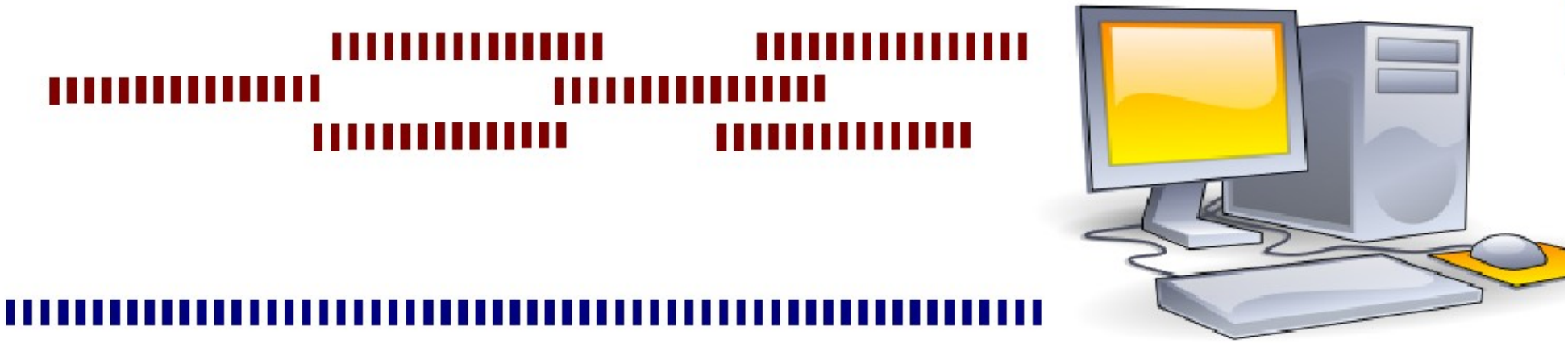
```python
def reverso(s1: str, s2: str) -> bool:
    n1 = len(s1)
    n2 = len(s2)

    if n1!=n2:
        return False

    i = 0
    while i<n1:
        if s1[i] != s2[n1-i-1]:
            return False
        i = i+1

    return True
```

Bioinformática?

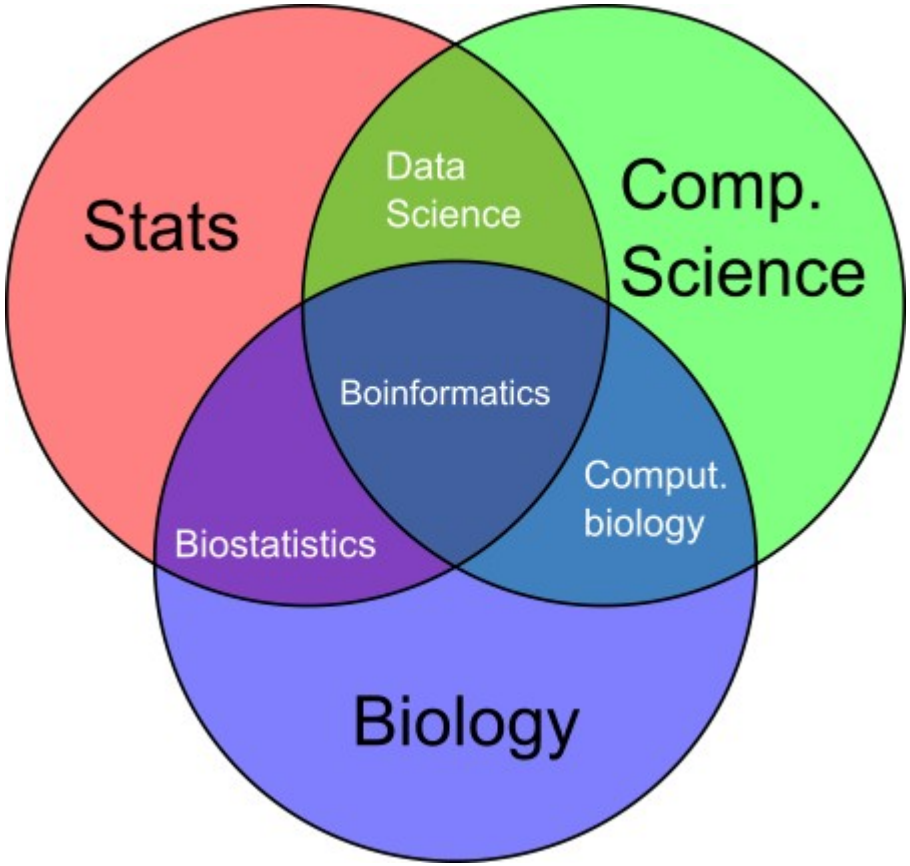
Biologia + Informática



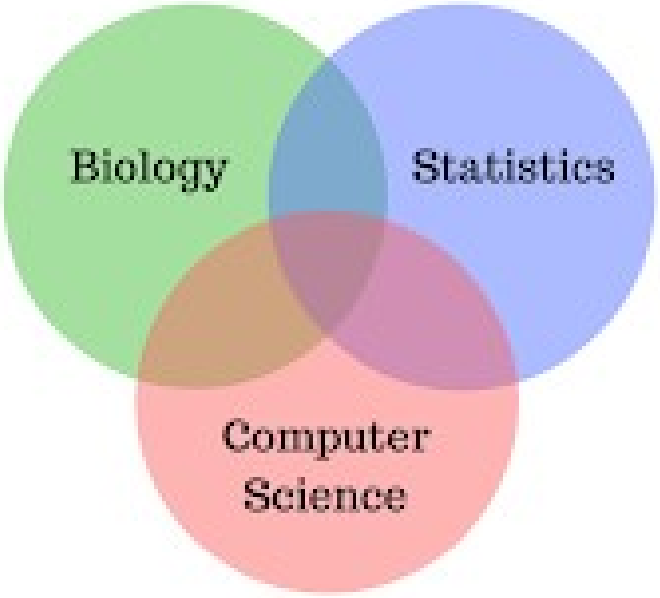
Biologia + Informática



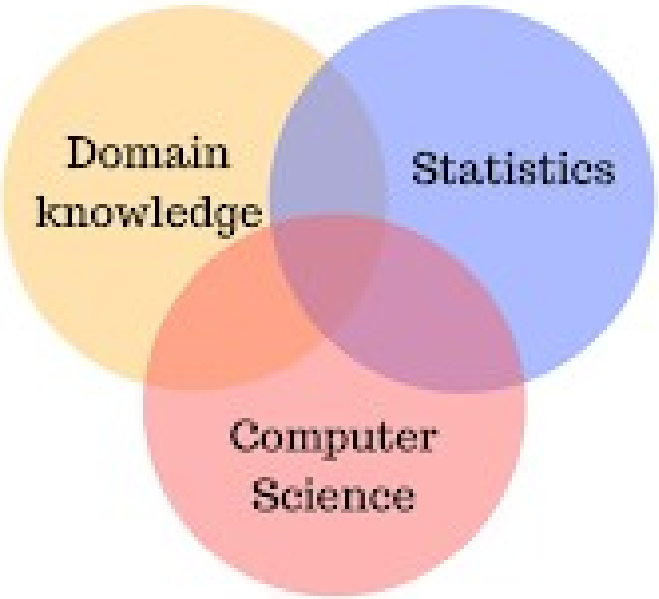
Biologia + Informática



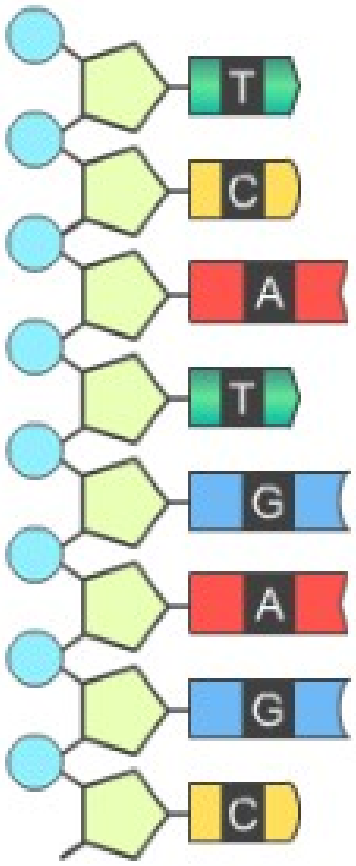
Bioinformatics

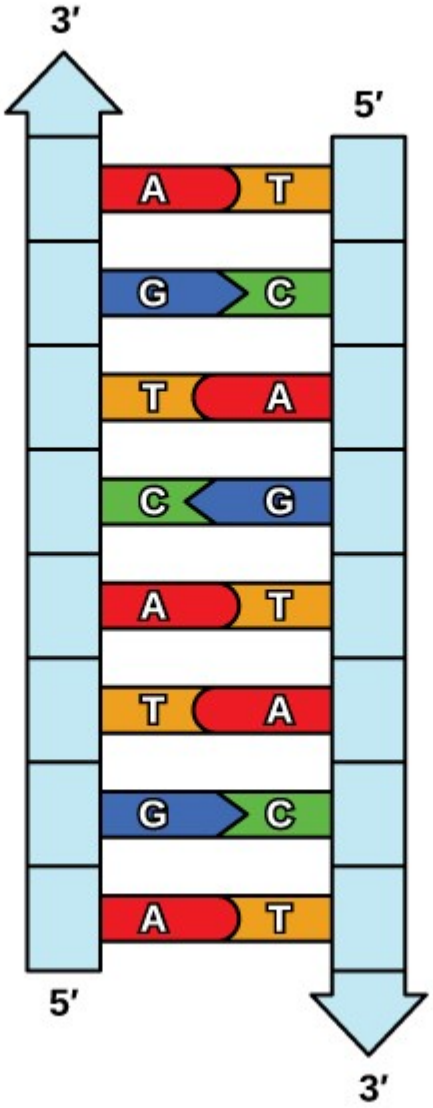


Data Science



Sequência de DNA como uma ‘string’





Uma aplicação: Identificar genes

198 IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, VOL. 5, NO. 2, APRIL-JUNE 2008

Identification of Protein Coding Regions Using the Modified Gabor-Wavelet Transform

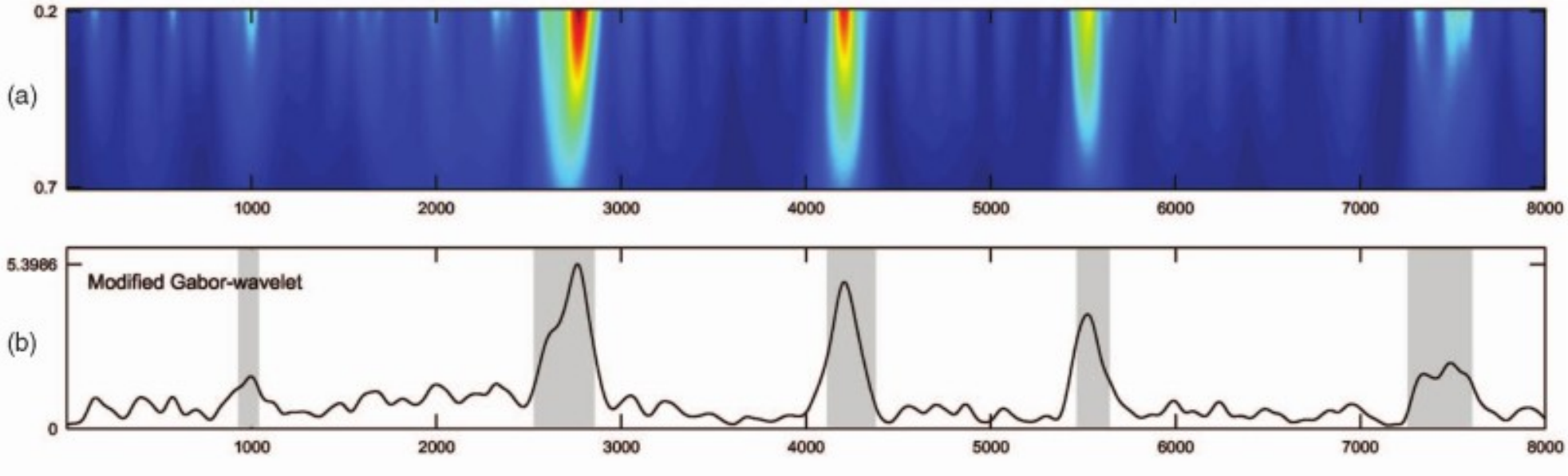
Jesús P. Mena-Chalco, Helaine Carrer, Yossi Zana, and Roberto M. Cesar Jr.

Abstract—An important topic in genomic sequence analysis is the identification of protein coding regions. In this context, several coding DNA model-independent methods based on the occurrence of specific patterns of nucleotides at coding regions have been proposed. Nonetheless, these methods have not been completely suitable due to their dependence on an empirically predefined window length required for a local analysis of a DNA region. We introduce a method based on a modified Gabor-wavelet transform (MGWT) for the identification of protein coding regions. This novel transform is tuned to analyze periodic signal components and presents the advantage of being independent of the window length. We compared the performance of the MGWT with other methods by using eukaryote data sets. The results show that MGWT outperforms all assessed model-independent methods with respect to identification accuracy. These results indicate that the source of at least part of the identification errors produced by the previous methods is the fixed working scale. The new method not only avoids this source of errors but also makes a tool available for detailed exploration of the nucleotide occurrence.

Index Terms—Sequence analysis, wavelet transform, coding regions, signal processing, pattern recognition.

Uma aplicação: Identificar genes

IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, VOL. 5, NO. 2, APRIL-JUNE 2008



Uma aplicação: Identificar genes

International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249 – 8958, Volume-9, Issue-1, October 2019

Modified Gabor Wavelet Transform in Prediction of Cancerous Genes

Lopamudra Das, Anand Kumar, J. K. Das, Sarita Nanda

Abstract*: Cancer is the leading cause of mortality all over the world which in general is the result of some kind of mutation in the genetic sequence. With recent advancements in Digital Signal Processing(DSP) techniques, it has become possible to classify cancerous gene sequences without carrying out extensive biological experiments. In this paper, the Geometric mapping technique along with Modified Gabor wavelet transform (MGWT) has been incorporated to segregate cancerous and non-cancerous gene sequences. This Gabor wavelet based transform technique used in the present research work benefits from the fact that it is independent of the window length which in conjunction with Geometric mapping is used to obtain the spectral components present in the signal accurately and with reduced complexity. This technique has been applied on numerous benchmark datasets and the results obtained prove the performance of the proposed method.

***Keywords:** Deoxyribonucleic Acid (DNA), Genomic Signal Processing(GSP), Modified Gabor Wavelet Transform (MGWT), Geometric mapping, cancer*

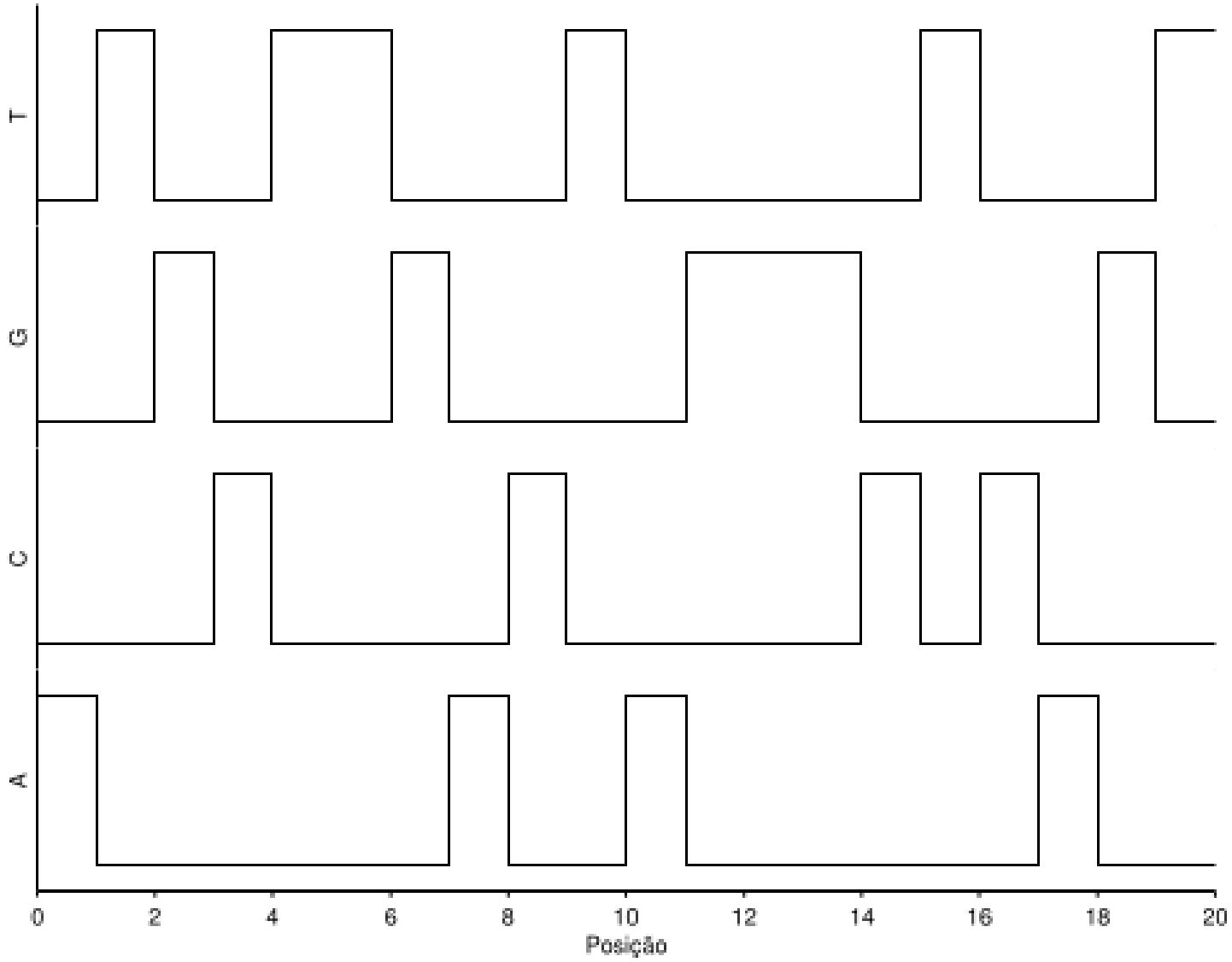
behavior is referred to as "periodicity property" or "three base periodicity". L. Das[6] proposed geometric mapping and compared its performance with several other mapping and found this mapping technique to be more efficient with least exon miss and false exons. Vidyanaath [7]explained the role of signal processing concepts in proteomics and genomics.

Cancer arises due to the alteration of nucleotides in the DNA sequence of the gene called mutation. In-depth analysis has been done in [8] that depicts the occurrence of cancer in an individual's genome. Cancer is a dreadful disease and has increased the mortality rate in recent years. It is grounded by scientists that cancer ascends due to the accretion of mutated cells in pivotal gene locations that changes the normal functioning of cell proliferation, metabolization, etc. Early spotting of cancer cell and identification of protein coding region will play a very crucial role in reducing the death due to cancer. Myriad of works has been done using DSP techniques for cancer prediction[9] Qui[10]. used Genomic

Transformando uma sequência de DNA

Seqüência	A	T	G	C	T	T	G	A	C	T	A	G	G	G	C	T	C	A	G	T
u_A	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
u_C	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0
u_G	0	0	1	0	0	0	1	0	0	0	0	1	1	1	0	0	0	0	1	0
u_T	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1

Transformando uma sequência de DNA



Transformando uma sequência de DNA

```
def obter_vetor_binario(DNA, base):
    vetor = list()

    for b in DNA:
        if b==base:
            vetor.append(1)
        else:
            vetor.append(0)

    return vetor
```

```python
def obter_vetor_binario(DNA, base):
    vetor = list()

    for b in DNA:
        if b==base:
            vetor.append(1)
        else:
            vetor.append(0)

    return vetor



SEQ = "ATCGATGCAAAACCCTTGGGATAG"

vA = obter_vetor_binario(SEQ, "A")
print(vA)
```