

Universidad Distrital Francisco José De Caldas



Lenguaje interpretado/compilado

Ingeniería en Sistemas

Analisis de Sistemas

Integrante:

JHONATHAN DAVID DE LA TORRE GARCIA 20222020033

Bogotá Colombia

Marzo 2024

Taller 1.

En el ejercicio propuesto se provee de información tal que sirva para realizar análisis sobre datos en cadenas de ADN, los detalles se enuncian a continuación:

Imagine you have been hired as data analyst in an important biotechnology company. Your boss, a Science Chief Officer, want to get some patterns in genomic data, sometimes called motifs. Here you will have some tasks in order to complete this workshop: 1. Create a dummy database of genetic sequences composed of nucleotide bases (A, C, G, T), where each sequence must have between 10 and 20 bases. Your database must be composed for 50.000 genetic sequences.. 2) Get the motifs (must repeated sequence) of size 6 and 8 . 3) Use the Shannon Entropy measurement to filter sequences with not a good variance level. 4) Get again the motifs of size 6 and 8 .

Para dar solución a los puntos del 1 al 2 , se usó el código que se muestra a continuación:

```
Click here to ask Blackbox to help you code faster
import random

from collections import Counter
def create_sequence():
    nucleotid_bases = ['A','C','G','T']
    size_sequence = random.randint(10,20)
    new_sequence = [ nucleotid_bases[random.randint(0,3)] for i in range(size_sequence)]
    return "".join(new_sequence)

✓ 0.0s Python

Click here to ask Blackbox to help you code faster
def create_database():
    db_size=50000
    data_base = [create_sequence () for i in range (db_size)] #secuencia que mas se repite motifs
    return data_base

✓ 0.0s Python
```

```

Click here to ask Blackbox to help you code faster
import math
def get_combinations ( n, sequence, bases): # recursividad para combinaciones
    if n==1:
        return [ sequence+bases[i] for sequence in sequence for i in range (len (bases))]
    else:
        sequence_ = [sequence+bases[i] for sequence in sequence for i in range (len(bases))]
        return get_combinations(n-1 , sequence_ , bases)

def count_motif (motif, sequences_db): #contador de motif
    count =0
    for sequence in sequences_db:
        count += sequence.count (motif)
    return count

def get_motif(motif_size , sequences_db): #combinaciones en una funcion, y probar las combinaciones, la ganadora se retorna
    nucleotid_bases= ['A','C','G','T']
    combinations = get_combinations ( motif_size, [""], nucleotid_bases)
    max_counter =0
    motif_winner=""
    for motif_candidate in combinations:
        temp_counter = count_motif ( motif_candidate, sequences_db)
        if temp_counter >max_counter:
            max_counter = temp_counter
            motif_winner = motif_candidate
    return motif_winner , max_counter

✓ 0.0s Python

```

El código señalado anteriormente fue creado y explicado durante el transcurso de la clase, así que procederé a explicar la continuación de dicho ejercicio:

Mi propuesta para calcular la entropía de Shannon es la siguiente:

```

Click here to ask Blackbox to help you code faster
def calculate_shannon_entropy(data_base):
    # Concatenar todas las secuencias en una sola cadena
    all_sequences = "".join(data_base)
    # Contar la frecuencia de cada símbolo en la secuencia completa
    symbol_counts = Counter(all_sequences)
    sequence_length = len(all_sequences)
    # Probabilidad de cada símbolo
    probabilities = [count / sequence_length for count in symbol_counts.values()]
    # Calcular la Entropía de Shannon
    entropy = -sum(p * math.log2(p) for p in probabilities if p != 0)
    return entropy

✓ 0.0s

Click here to ask Blackbox to help you code faster

for size in [6, 8]:
    print(f"\nAfter filter, motifs of size: {size}")
    for i in range(10):
        dataset = create_database()
        dataset = list(filter(filter_shannon, dataset))
        print(f"Dataset size: {len(dataset)}, Motif: {get_motif(size, dataset)}")

Click here to ask Blackbox to help you code faster
print(calculate_shannon_entropy(create_database()))

✓ 0.2s
1.999999061470457

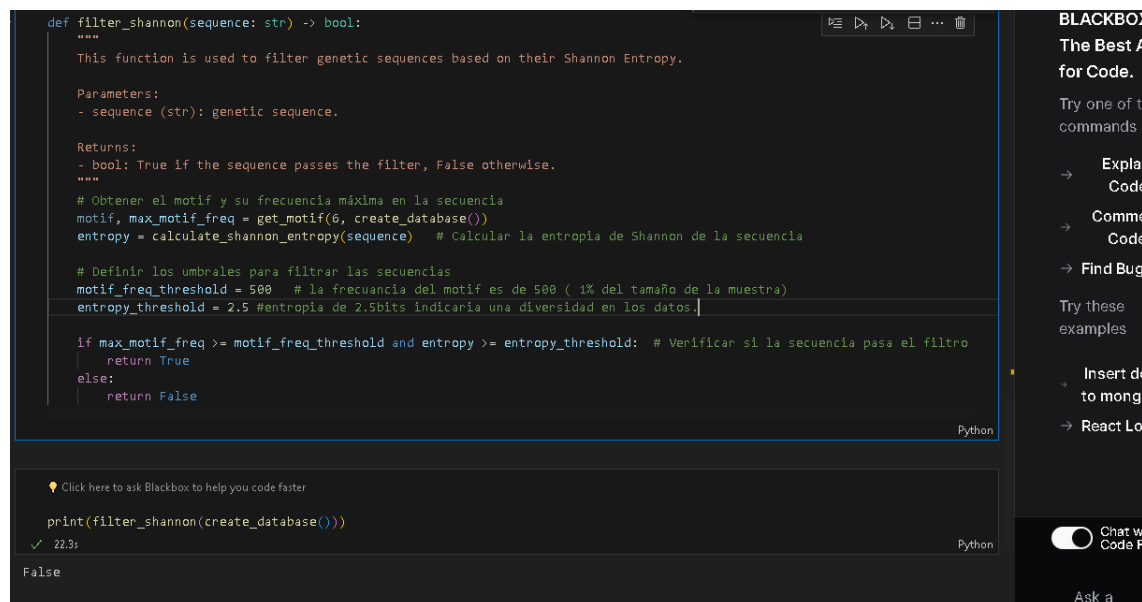
```

El objetivo del código es contar la frecuencia con la que ocurre cada símbolo en la secuencia de la base de datos, para posteriormente calcular la probabilidad de que ocurra un símbolo, ósea se calcula las probabilidades de ocurrencia de cada símbolo en la secuencia usando la relación de la frecuencia de cada símbolo dividida por la longitud total de la secuencia , y con ello aplicamos la fórmula de Shannon , teniendo en cuenta que la variable entropy es una expresión generadora que itera sobre las probabilidades de cada símbolo en la secuencia.

Con este algoritmo nos debe arrojar un numero que nos indicara la cantidad de información (sorpresa) de nuestra secuencia de datos, para el caso particular nos dio un valor cercano a 2, eso quiere decir que nos esta suministrando cerca de 2 bits de información.

Ahora, para aplicar el filtro tuve en cuenta dos condiciones principales, la primera consiste en la frecuencia del Motif, calculado anteriormente, según el conjunto de pruebas que realice un buen numero a tener en cuenta es que si quiero evitar la monogamia esta frecuencia debe ser un numero bastante bajo, así que asumí que la frecuencia no debería ser mayor del 1% del total de los datos.

La segunda condición consiste en darle un valor a la entropía calculada para conocer como se encuentran de repetidos nuestros datos, un valor bajo de entropía no nos da información suficiente, ya que serían datos muy repetidos. Tomando en cuenta lo que aprendí en el pasado, sabiendo que existen 4 símbolos posibles y sus combinaciones, tome un tope máximo de entropía de 4 bits de información, así que tomando un valor intermedio (2 de entropía es suficiente), considere tomar 2.5 para darle un margen mas alto a los valores para su análisis, el código se muestra a continuación:



```
def filter_shannon(sequence: str) -> bool:
    """
    This function is used to filter genetic sequences based on their Shannon Entropy.

    Parameters:
    - sequence (str): genetic sequence.

    Returns:
    - bool: True if the sequence passes the filter, False otherwise.
    """
    # Obtener el motif y su frecuencia máxima en la secuencia
    motif, max_motif_freq = get_motif(6, create_database())
    entropy = calculate_shannon_entropy(sequence) # Calcular la entropia de Shannon de la secuencia

    # Definir los umbrales para filtrar las secuencias
    motif_freq_threshold = 500 # la frecuencia del motif es de 500 ( 1% del tamaño de la muestra)
    entropy_threshold = 2.5 #entropia de 2.5bits indicaria una diversidad en los datos.

    if max_motif_freq >= motif_freq_threshold and entropy >= entropy_threshold: # Verificar si la secuencia pasa el filtro
        return True
    else:
        return False
```

Click here to ask Blackbox to help you code faster

```
print(filter_shannon(create_database()))
```

✓ 22.3s

False

Python

BLACKBOX
The Best A
for Code.
Try one of th
commands
→ Explai
Code
→ Comme
Code
→ Find Bug
Try these
examples
→ Insert do
to mongd
→ React Lo
Chat w
Code F
Ask a

Ahora para la parte final, mi objetivo era tomar una lista filtrada y recalcular la frecuencia del motif, su tamaño y ver como mi condición afecto a los datos, sin embargo no conseguí resultados óptimos por falta de tiempo, así que mi idea queda inconclusa .

Sin embargo, quería ejecutar el código siguiente para analizar dicho filtro:

💡 Click here to ask Blackbox to help you code faster

```
for size in [6, 8]:
    print(f"\nAfter filter, motifs of size: {size}")
    for i in range(10):
        dataset = create_database()
        dataset = list(filter(filter_shannon, dataset))
        print(f"Dataset size: {len(dataset)}, Motif: {get_motif(size, dataset)}")
```

Mi problema radicaba en que estoy retornando un booleano, mas no una secuencia, así que en u futuro cercano espero contar con mas tiempo para realizar pruebas.