Universidad Nacional de San Agustin

Escuela Profesional de
Ciencia de la Computación

COMPUTACIÓN MOLECULAR BIOLÓGICA

Uso del análisis de textura de imágenes para encontrar similitudes en la secuencia de ADN

Alumnos:
Alberto Visa Flores
Sergio Arcos Ponce
Gustavo Leon Paredes
Alfred Guardia Zenteno
Docente:
Mg. Vicente Machaca

18 de agosto de 2020

Índice

1. Introducción

El análisis de similitud de secuencia es la técnica básica para construir árboles filogenéticos, que analizan las funciones de los genes y predicen las estructuras de las proteínas. La alineación de secuencia es la más utilizada y método de análisis de similitud intuitivo. Muchas secuencias de alineación.

2. GLCM

Para calcular eficazmente las propiedades del ADN secuencias y para realizar análisis de similitud, proponemos un método basado en la teoría de la matriz de co-ocurrencia de niveles de gris (GLCM), que es un método estadístico bien conocido y de uso común método en el análisis de la textura de la imagen. Definir y especificar valores de características para cada secuencia es útil para encontrar secuencias similares en bases de datos, especialmente cuando el La base de datos es muy grande y una comparación de secuencia uno por uno es pérdida de tiempo. GLCM puede definir y calcular características relacionadas con cada secuencia; estas características también pueden indicar similitudes entre secuencias. Por lo tanto, las características definidas pueden ser los valores clave en cada secuencia. Al ingresar una secuencia y encontrar sus secuencias similares, todo lo que se requiere es calcular su valor de característica y secuencias de salida que tienen valores de características similares a los de la secuencia de entrada. Esto puede ahorrar mucho tiempo de búsqueda en la base de datos.

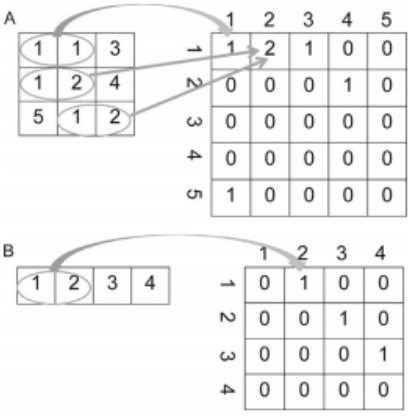

Figura 1: GLCM

3. Transformar una secuencia de ADN en un vector digital

Una secuencia de ADN está representada por una cadena que comprende A (adenina), C (citosina), G (guanina) y T (timina). En este documento, definir un método para transformar cada secuencia de ADN en un vector digital, que luego se puede utilizar para calcular la matriz de co-ocurrencia en la figura1 En el método propuesto, el procedimiento consiste en utilizar primero números enteros 1, 2, 3 y 4 para representar las cuatro bases A, C, G y T respectivamente. En segundo lugar, el número de cada carácter en la secuencia de ADN se suma al valor entero anterior.

4. Características basadas en la matriz de co-ocurrencia

Las características utilizadas fueron entropía, contraste, energía, correlación,y homogeneidad.

- **Entropia:**

$$Entropy = -\sum_{i=1}^{L}\sum_{j=1}^{L} p(i,j)Ln(p(i,j)),$$

- **Contraste:**

$$Contrast = \sum_{i=1}^{L}\sum_{j=1}^{L} (i-j)^2\ p(i,j),$$

- **Energia:**

$$Energy = \sum_{i=1}^{L}\sum_{j=1}^{L} p(i,j)^2,$$

- **Correlacion:**

$$Correlation = \sum_{i=1}^{L}\sum_{j=1}^{L}\left(\frac{(i-\mu_i)(j-\mu_j)p(i,j)}{\sigma_i\sigma_j}\right),$$

- **Homogeneidad:**

$$Homogeneity = \sum_{i=1}^{L}\sum_{j=1}^{L}\left(\frac{p(i,j)}{1+|i-j|}\right),$$

4.1. Codigo

```
from google.colab import drive
import re
import numpy as np
import re
def string_to_array(my_string):
    my_string = my_string.lower()
    my_string = re.sub('[^acgt]', 'z', my_string)
    my_array = np.array(list(my_string))
    return my_array
from sklearn.preprocessing import LabelEncoder
def ordinal_encoder(my_array):
    label_encoder = LabelEncoder()
    label_encoder.fit(np.array(['a','c','g','t','z']))
    integer_encoded = label_encoder.transform(my_array)
    float_encoded = integer_encoded.astype(float)
    float_encoded[float_encoded == 0] = 0 # A
    float_encoded[float_encoded == 1] = 1 # C
    float_encoded[float_encoded == 2] = 2 # G
    float_encoded[float_encoded == 3] = 3 # T
    float_encoded[float_encoded == 4] = 4 # anything else, z
    return float_encoded
from sklearn.preprocessing import OneHotEncoder
def one_hot_encoder(my_array):
    integer_encoded = label_encoder.transform(my_array)
    onehot_encoder = OneHotEncoder(sparse=False, dtype=int, n_values=5)
    integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
    onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
    onehot_encoded = np.delete(onehot_encoded, -1, 1)
    return onehot_encoded
def glcm(m_array):
  a=np.amax(m_array)
  a=int(a)
  template=np.zeros((a+1,a+1))
  for i in range(m_array.shape[0]):
    for j in range(m_array.shape[1]-1):
      if m_array[i][j]!=-1. and m_array[i][j+1]!=-1.:
        template[int(m_array[i][j])][int(m_array[i][j+1])]+=1
```

```
38    return template
39  def glcm_vect(m_array):
40    template=np.zeros((4,4))
41    for i in range(m_array.shape[0]-1):
42      template[int(m_array[i])][int(m_array[i+1])]+=1
43    return template
44
45  def contrast(matriz_g):
46    resultado=0
47    for i in range(matriz_g.shape[0]):
48      for j in range(matriz_g.shape[1]):
49        resultado+=((i-j)**2)*matriz_g[i][j]
50    return resultado
51  def energy(matriz_g):
52    resultado=0
53    for i in range(matriz_g.shape[0]):
54      for j in range(matriz_g.shape[1]):
55        resultado+=matriz_g[i][j]
56    return resultado
57  def entropy(matriz_g):
58    a=np.log(matriz_g)
59    resultado=np.sum(a)
60    return resultado
61
62  from skimage.feature import greycomatrix, greycoprops
63  from multiprocessing import Pool
64  def glcm_props(patch):
65      lf = []
66      props = ['entropy', 'contrast', 'homogeneity', 'energy', '
      correlation']
67
68      #para que vaya a la derecha np.pi/4
69      glcm = greycomatrix(patch, [1], [np.pi/4], 256, symmetric=True,
      normed=True)
70      for f in props:
71          lf.append( greycoprops(glcm, f)[0,0] )
72
73      a=[]
74      for f in props:
75          a.append( greycoprops(glcm, f)[0,0] )
76      return a
77
78  from Bio import SeqIO
79  data=[]
```

```
80 for seq_record in SeqIO.parse('/content/drive/My Drive/molecular/
       example.fa', "fasta"):
81     a=str(seq_record.seq)
82     montar=string_to_array(a)
83     montar=ordinal_encoder(montar)
84     ggg=np.array(montar)
85     m_glcm=glcm_vect(ggg)
86     a=[]
87     if int(ggg.shape[0]%4)!=0:
88       new_cont=(int(ggg.shape[0]//4)+1)*4
89       a=np.zeros(int(new_cont-ggg.shape[0]))
90     ggg=np.append(ggg,a)
91     f=ggg.shape[0]
92     gope =np.array(ggg).reshape(int(f/4),4)
93     a=glcm(gope)
94     a=a.astype(np.uint8)
95     gope=gope.astype(np.uint8)
96     data.append(glcm_props(gope))
97 for seq_record in SeqIO.parse('/content/drive/My Drive/molecular/
       Macaca_fascicularis_chromosome10.fa', "fasta"):
98     a=str(seq_record.seq)
99     montar=string_to_array(a)
100    montar=ordinal_encoder(montar)
101    ggg=np.array(montar)
102    #m_glcm=glcm_vect(ggg)
103    a=[]
104    if int(ggg.shape[0]%4)!=0:
105      new_cont=(int(ggg.shape[0]//4)+1)*4
106      a=np.zeros(int(new_cont-ggg.shape[0]))
107    ggg=np.append(ggg,a)
108    f=ggg.shape[0]
109    gope =np.array(ggg).reshape(int(f/4),4)
110    a=glcm(gope)
111    a=a.astype(np.uint8)
112    gope=gope.astype(np.uint8)
113    data.append(glcm_props(gope))
114 print(data)
115
116 for seq_record in SeqIO.parse('/content/drive/My Drive/molecular/
       Macaca_mulatta_nonchromosomal.fa', "fasta"):
117    a=str(seq_record.seq)
118    montar=string_to_array(a)
119    montar=ordinal_encoder(montar)
120    ggg=np.array(montar)
121    #m_glcm=glcm_vect(ggg)
```

```
122     a=[]
123     if int(ggg.shape[0]%4)!=0:
124       new_cont=(int(ggg.shape[0]//4)+1)*4
125       a=np.zeros(int(new_cont-ggg.shape[0]))
126     ggg=np.append(ggg,a)
127     f=ggg.shape[0]
128     gope =np.array(ggg).reshape(int(f/4),4)
129     a=glcm(gope)
130     a=a.astype(np.uint8)
131     gope=gope.astype(np.uint8)
132     data.append(glcm_props(gope))
133
134 for seq_record in SeqIO.parse('/content/drive/My Drive/molecular/
    Macaca_mulatta_nonchromosomal.fa', "fasta"):
135     a=str(seq_record.seq)
136     montar=string_to_array(a)
137     montar=ordinal_encoder(montar)
138     ggg=np.array(montar)
139     #m_glcm=glcm_vect(ggg)
140     a=[]
141     if int(ggg.shape[0]%4)!=0:
142       new_cont=(int(ggg.shape[0]//4)+1)*4
143       a=np.zeros(int(new_cont-ggg.shape[0]))
144     ggg=np.append(ggg,a)
145     f=ggg.shape[0]
146     gope =np.array(ggg).reshape(int(f/4),4)
147     a=glcm(gope)
148     a=a.astype(np.uint8)
149     gope=gope.astype(np.uint8)
150     data.append(glcm_props(gope))
```

5. Resultados

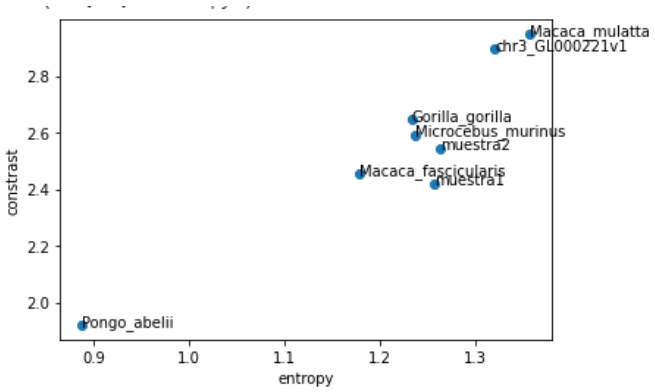

Figura 2: entropy vs contrast

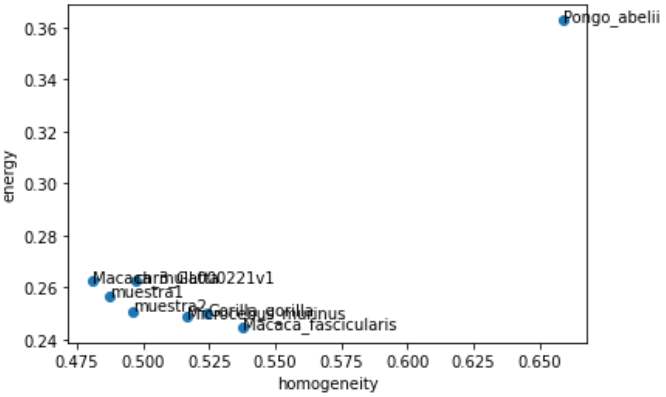

Figura 3: homogeneity vs entropy

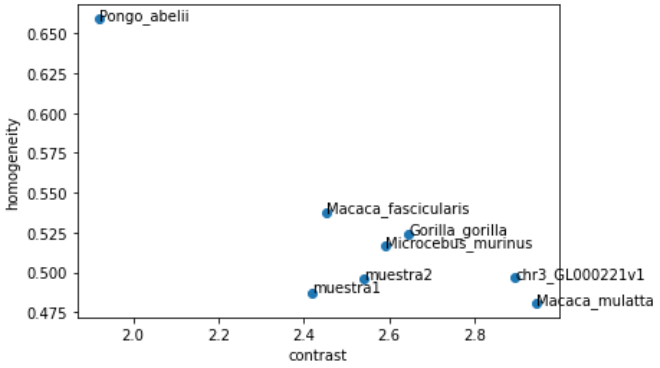

Figura 4: contrast vs homogeneity

6. Conclusiones

- Con este metodo podemos ver de manera grafica la similitud que hay entre una secuencia y otra.
- Se utiliza GLCM para sacar una matriz de texturas para cada secuencia,realizando las operaciones de entropia, contraste, energia, correlación, homogeneidad.

7. Repositorio

https://github.com/widcatd/molecular_glcm

Referencias

[1] Weiyang Chena, Bo Liao , Weiwei Li ,2018, Use of image texture analysis to find DNA sequence similarities,doi:10.1016/j.jtbi.2018.07.001