PROYECTO PREDICCION DE UNA CELULA PROCARIOTA UNICELULARES

Autor Juan Cuevas vasquez

BioInformatico ORCID-NCBI

Objetivos General

El objetivo principal y especifico de este proyecto es dejar estampado cientificamente la importancia de la INTELIGENCIA ARTIFICIAL en uno de los eventos moleculares mas complejos de la ciencia actual les hablo de la forma de como predecir el Comportamiento de una celula en este caso la bacteria **Bacillus subtilis**.

Objetivo Especifico

El objetivo Esecifico es el siguente: Lograr predecir la estructura del operón en bacterias. Un operón es un conjunto de genes adyacentes en la misma hebra de ADN que se transcriben en una sola molécula de ARNm. La traducción de la única molécula de mRNA luego produce las proteínas individuales. Para Bacillus subtilis , cuyos datos usaremos, el número promedio de genes en un operón es de aproximadamente 2,4.

Como primer paso para comprender la regulación génica en bacterias, necesitamos conocer la estructura del operón. Para alrededor del 10% de los genes en Bacillus subtilis , la estructura del operón se conoce a partir de experimentos. Se puede utilizar un método de aprendizaje supervisado para predecir la estructura del operón para el 90% restante de los genes.

Como podemos apreciar solo conocemos el 10% de los genes de esta bacteria para lo cual no enfocaremos al 90% restantes de los genes de la bacteria mostraremos a continuacion una imgen de esta bacteria

imagen

TECNOLOGIA UTILIZADAS EN EL PROYECTO:

LENGUAJES DE PROGRAMCION

PYTHON SERVIDORES: GOOGLE COLAB: PROYECTO PROCESADO EN LA NUBE DE GOOGLE. BASE DE DATOS: NCBI CENTRO INTERNACIONAL DE BIOTECNOLOGIA. PROYECTO ALMACENADO EN ORCID CENTRO DE INVESTIGAORES CINTIFICOS QUE APORTAN DATOS RELEVANTE A LA CIENCIA MUNDIAL.EN LA CUAL ACTUALMENETE ESTOY REPRESENTANDO A CHILE COMO CIENTIFICO DE DATOS.

LINK DE ORCID: https://orcid.org/my-orcid?orcid=0000-0002-2778-1922

METODOLOGIAS MATEMATICAS

Para tal enfoque de aprendizaje supervisado, necesitamos elegir algunas variables predictoras x ique se pueden medir fácilmente y están relacionados de alguna manera con la estructura del operón. Una variable predictora podría ser la distancia en pares de bases entre genes. Los genes adyacentes que pertenecen al mismo operón tienden a estar separados por una distancia relativamente corta, mientras que los genes adyacentes en diferentes operones tienden a tener un espacio más grande entre ellos para permitir secuencias promotoras y terminadoras. Otra variable predictora se basa en mediciones de expresión génica. Por definición, los genes que pertenecen al mismo operón tienen perfiles de expresión de genes iguales, mientras que se espera que los genes en diferentes operones tengan diferentes perfiles de expresión. En la práctica, los perfiles de expresión medidos de genes en el mismo operón no son del todo idénticos debido a la presencia de errores de medición. Para evaluar la similitud en los perfiles de expresión génica,

Ahora tenemos dos variables predictoras que podemos usar para predecir si dos genes adyacentes en la misma hebra de ADN pertenecen al mismo operón:

x 1 : el número de pares de bases entre ellos; x 2 : su similitud en el perfil de expresión. En un modelo de regresión logística, usamos una suma ponderada de estos dos predictores para calcular una puntuación conjunta S :

S = β 0 + β 1 X 1 + β 2 X 2 .(16.1) El modelo de regresión logística nos da valores apropiados para los parámetros β 0 , β 1 , β 2 utilizando dos conjuntos de genes de ejemplo:

OP: genes adyacentes, en la misma hebra de ADN, que se sabe que pertenecen al mismo operón; NOP: genes adyacentes, en la misma hebra de ADN, que se sabe que pertenecen a diferentes operones. En el modelo de regresión logística, la probabilidad de pertenecer a una clase depende de la puntuación a través de la función logística. Para las dos clases OP y NOP, podemos escribir esto como

Pr(PO | x 1 , x 2) =

Exp(β 0 + β 1 x 1 + β 2 x 2) 1+exp(β 0 + β 1 x 1 + β 2 x 2) (16.2) Pr(NOP | x 1 , x 2) =

1 1+exp(β 0 + β 1 x 1 + β 2 x 2) (16.3)

ALGORITMO DE INTELIGENCIA ARTIFICIAL UTILIZADO

Modelo de Regresión Logística

```
!pip install Bio
from Bio import LogisticRegression
```

```
Collecting Bio
  Downloading bio-1.3.8-py3-none-any.whl (269 kB)
     |████████████████████████████████| 269 kB 4.2 MB/s
Collecting mygene
  Downloading mygene-3.2.2-py2.py3-none-any.whl (5.4 kB)
Collecting biopython>=1.79
  Downloading biopython-1.79-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (2.3
     |████████████████████████████████| 2.3 MB 52.1 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from Bio)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from bio
Collecting biothings-client>=0.2.6
  Downloading biothings_client-0.2.6-py2.py3-none-any.whl (37 kB)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib
Installing collected packages: biothings-client, mygene, biopython, Bio
Successfully installed Bio-1.3.8 biopython-1.79 biothings-client-0.2.6 mygene-3.2.2
```

hemos importado las librerias necesarias para el proyecto.

```
from Bio import LogisticRegression
```

ahora crearemos una matriz bidimensional con las variables x para lograr predecir el comportamiento de la bacteria.

```
x_moli_cat = [[-53, -200.78],
     [117, -267.14],
      [57, -163.47],
      [16, -190.30],
       [11, -220.94],
       [85, -193.94],
       [16, -182.71],
       [15, -180.41],
       [-26, -181.73],
       [58, -259.87],
      [126, -414.53],
      [191, -249.57],
       [113, -265.28],
       [145, -312.99],
      [154, -213.83],
       [147, -380.85],
       [93, -291.13]]


y_moli_cat = [1,
       1,
      1,
```

```
 1,
1,
1,
1,
1,
 1,
 1,
 0,
0,
 0,
 0,
 0,
 0,
 0]
```

```
model = LogisticRegression.train(x_moli_cat, y_moli_cat)
```

Aquí, x_moli_cat y_moli_cat están los datos de entrenamiento: x_moli_cat contiene las variables predictoras para cada par de genes y ysespecifica si el par de genes pertenece al mismo operón (1, clase OP) oa diferentes operones (0, clase NOP). El modelo de regresión logística resultante se almacena en model, que contiene los pesos β 0 , β 1 y β 2 :

```
model.beta
```

```
[8.98302901571447, -0.035968960444850887, 0.021813956629835197]
```

Tenga en cuenta que β 1 es negativo, ya que los pares de genes con una distancia intergénica más corta tienen una mayor probabilidad de pertenecer al mismo operón (clase OP). Por otro lado, β 2 es positivo, ya que los pares de genes que pertenecen al mismo operón suelen tener una mayor puntuación de similitud de sus perfiles de expresión génica. El parámetro β 0 es positivo debido a la mayor prevalencia de pares de genes de operón que de pares de genes que no son de operón en los datos de entrenamiento.

ahora crearemos una funcion llamada mostar progreso

```
def mostrar_progres(iteratión, loglikelihood):
        print("Iteration:", iteratión, "Log-likelihood function:", lolikelihood)
```

```
model = LogisticRegression.train(x_moli_cat, y_moli_cat, update_fn=mostrar_progreso)
```

```
Iteration: <built-in function iter> Log-likelihood function: -11.78350206951907
Iteration: <built-in function iter> Log-likelihood function: -7.158867676721056
Iteration: <built-in function iter> Log-likelihood function: -5.768772098679432
```

```
Iteration: <built-in function iter> Log-likelihood function: -5.113622943382592
Iteration: <built-in function iter> Log-likelihood function: -4.748706424325652
Iteration: <built-in function iter> Log-likelihood function: -4.50026077146048
Iteration: <built-in function iter> Log-likelihood function: -4.311277737371034
Iteration: <built-in function iter> Log-likelihood function: -4.1601504339559465
Iteration: <built-in function iter> Log-likelihood function: -4.035617197847367
Iteration: <built-in function iter> Log-likelihood function: -3.93073282192017
Iteration: <built-in function iter> Log-likelihood function: -3.8408766092914273
Iteration: <built-in function iter> Log-likelihood function: -3.762825606050504
Iteration: <built-in function iter> Log-likelihood function: -3.69425027154435
Iteration: <built-in function iter> Log-likelihood function: -3.63341786019592
Iteration: <built-in function iter> Log-likelihood function: -3.579008558366153
Iteration: <built-in function iter> Log-likelihood function: -3.529996713864589
Iteration: <built-in function iter> Log-likelihood function: -3.4855714516343337
Iteration: <built-in function iter> Log-likelihood function: -3.4450820613930997
Iteration: <built-in function iter> Log-likelihood function: -3.4079994844651083
Iteration: <built-in function iter> Log-likelihood function: -3.3738885623997366
Iteration: <built-in function iter> Log-likelihood function: -3.3423876581020284
Iteration: <built-in function iter> Log-likelihood function: -3.3131934376911234
Iteration: <built-in function iter> Log-likelihood function: -3.286049334600595
Iteration: <built-in function iter> Log-likelihood function: -3.2607366863005254
Iteration: <built-in function iter> Log-likelihood function: -3.2370678409148406
Iteration: <built-in function iter> Log-likelihood function: -3.214880736138568
Iteration: <built-in function iter> Log-likelihood function: -3.194034592585724
Iteration: <built-in function iter> Log-likelihood function: -3.1744064605163884
Iteration: <built-in function iter> Log-likelihood function: -3.1558884270318726
Iteration: <built-in function iter> Log-likelihood function: -3.138385339473235
Iteration: <built-in function iter> Log-likelihood function: -3.121812935946028
Iteration: <built-in function iter> Log-likelihood function: -3.1060962996567087
Iteration: <built-in function iter> Log-likelihood function: -3.0911685728203993
Iteration: <built-in function iter> Log-likelihood function: -3.076969880170042
Iteration: <built-in function iter> Log-likelihood function: -3.0634464228770586
Iteration: <built-in function iter> Log-likelihood function: -3.050549711911315
Iteration: <built-in function iter> Log-likelihood function: -3.0382359161858656
Iteration: <built-in function iter> Log-likelihood function: -3.026465305727712
Iteration: <built-in function iter> Log-likelihood function: -3.015201773938085
Iteration: <built-in function iter> Log-likelihood function: -3.0044124260112417
Iteration: <built-in function iter> Log-likelihood function: -2.994067222959573
Iteration: <built-in function iter> Log-likelihood function: -2.9841386725875947
```

La iteración se detiene una vez que el aumento en la función de probabilidad logarítmica es inferior a 0,01. Si no se alcanza la convergencia después de 500 iteraciones, la trainfunción regresa con un AssertionError.

Uso del modelo de regresión logística para la clasificación La clasificación se realiza llamando a la classifyfunción. Dado un modelo de regresión logística y los valores de x 1 y x 2 (por ejemplo, para un par de genes de estructura de operón desconocida), la classifyfunción devuelve 1o 0, correspondiente a la clase OP y la clase NOP, respectivamente. Por ejemplo, consideremos los pares de genes yxcE , yxcD y yxiB , yxiA :

```
print("yxcE, yxcD:", LogisticRegression.classify(modelo, [6, -173.143442352]))
```

```
yxcE, yxcD: 1
```

```
print("yxiB, yxiA:", LogisticRegressio.classify(model, [ 309, -271.005880394]))
```

```
yxiB, yxiA: 0
```

en este evento podemos visualizar que el modelo ya aprendio a diferencia a los genes de un mismo operon o a los de diferente operon osea un gen desconocido que puede crear ARNm nuevos y peligrosos y nocivos para la salud humana.

olvidaba mencionar que esto concuerda con la literatura biologica sobre la bacteria Bacillus subtilis. en donde laboratorios no virttuales ya se encontraron mutaciones de la bacteria.esta es un modelo creado sobre datos reales.

TESTEO DEL ALGORITMO

COMPROBAMOS LA EXACVTITUD DEL ALGORITMO CON LA FUNCION://CALCULATEFUNCION

para las clases OP y NOP. para yxcE, yxcD we find 99% DE EXACTITUD

```
q, p = LogisticRegression.calculate(model, [6, -173.143442352])
print("clase OP: probabilidad =", p, "clase NOP: probabilidad =", q)
```

```
clase OP: probabilidad = 0.9932421635025626 clase NOP: probabilidad = 0.006757836497437
```

```
q, p = LogisticRegression.calculate(model, [309, -271.005880394])
print("clase OP: probabilidad =", p, "clase NOP: probabilidad =", q)
```

```
clase OP: probabilidad = 0.00032121125181733316 clase NOP: probabilidad = 0.99967878874
```

Y PARA LA PRECISON DE BUSCAR UN GEN DESCONOCIDO Y PELIGROSO ES DE UN 99.9%

la precisión de la predicción del modelo de regresión logística, podemos aplicarlo a los datos de entrenamiento:

```
for i in range(len(y_moli_cat)):
        print("True:", y_moli_cat[i], "Predicted:", LogisticRegression.classify(modelo, x_mol
```

```
True: 1 Predicted: 1
True: 1 Predicted: 0
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
```

IMPORTANTE

NOTA QUE: mostrando que la predicción es correcta para todos menos uno de los pares de genes. Se puede encontrar una estimación más confiable de la precisión de la predicción a partir de un análisis de exclusión, en el que el modelo se vuelve a calcular a partir de los datos de entrenamiento después de eliminar el gen que se va a predecir:

```
for i in range(len(y_moli_cat)):
        model = LogisticRegression.train(x_moli_cat[:i]+x_moli_cat[i+1:], y_moli_cat[:i]+y_mo
        print( "Verdadero:", y_moli_cat[i], "Predicho por algoritmo la gata moli o moli_cat:"
```

```
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 0
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 1 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 0 Predicho por algoritmo la gata moli o moli_cat: 0
Verdadero: 0 Predicho por algoritmo la gata moli o moli_cat: 0
Verdadero: 0 Predicho por algoritmo la gata moli o moli_cat: 0
Verdadero: 0 Predicho por algoritmo la gata moli o moli_cat: 0
Verdadero: 0 Predicho por algoritmo la gata moli o moli_cat: 1
Verdadero: 0 Predicho por algoritmo la gata moli o moli_cat: 0
Verdadero: 0 Predicho por algoritmo la gata moli o moli_cat: 0
```

en la lista de verdadero es el valor real del gen de la bacteria. y en la

El análisis de exclusión muestra que la predicción del modelo de regresión logística es incorrecta solo para dos de los pares de genes, lo que corresponde a una precisión de predicción del 88 %.

precision alcanzada con algoritmo CAT MOLI ES DE UN 88%

VERSION 1.0

CREADO POR JUAN ALEJANDRO CUEVAS VASQUEZ CHILE CIENTIFICO DE DATOS DE ORCID

https://orcid.org/my-orcid?orcid=0000-0002-2778-1922

validacion de proyecto mediante codigo QR

Agradecimientos a mi esposa fanny gomez la cual me ha apoyado a lo largo de mi carrera este proyecto esta dedicado a ella y mi padre.