MÜHENDİSLİK ve DOĞA BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Hesaplamalı Biyolojiye Giriş

ÖDEV 1

NC_001416 Canlısının Genom Diziliminin İncelenmesi

Fatih ATEŞ 19360859074

Dr. Öğr. Üyesi Ergün GÜMÜŞ

2021

1. TANIM

NC_001416 kimliğine sahip canlının poisson süreci ile belirlenen beklenti değerleri ve sekanslarından elde edilen gözlem değerlerinin X^2 testine göre değerlendirilmesi.

2. PROGRAM GEREKSİNİMLERİ

Betik yazımında Python Programlama Dilinin 3.8.5 versiyonu kullanılmıştır. Dolayısıyla bilgisayarınızda Python 3.8.5 versiyonu veya üzeri bir daha yeni versiyonunun kurulu olması gerekir.

Python 3 indirmek için: https://www.python.org/downloads/ .
Python 3 kurulum için: https://realpython.com/installing-python/

Python 3 kurulumu tamamlandıktan sonra kurulması gereken paketler için aşağıdaki komutu çalıştırabilirsiniz.

> pip3 install requirements.txt

Gerekli paketlerin kurulumu tamamlandıktan sonra aşağıdaki komut iler programı çalıştırabilirsiniz.

> python3 main.py

3. NASIL ÇALIŞIR ?

Öncelikle aşağıdaki komutlar yardımıyla kullanılacak olan paketler betiğin içerisine dahil edilir.

- **String içerisinde arama işlemleriyle uğraştığımız için ‘re’ kütüphanesi,**
- **Exponential değer, ondalık sayı yuvarlama gibi işlemlerde yardım alacağımız ‘math’ kütüphanesi,**
- **‘fasta’ gibi dosya türlerinden sekansları okumamızı sağlayan ‘Bio’ kütüphanesi,**
- **İntegral almada kullanacağımız ‘scipy’ kütüphanesi,**
- **Konsol ekranında formatlı bir şekilde tablo oluşturmamızı sağlayan ‘terminaltables’ kütüphanesi**

dahil ediliyor.

```
import re
import math
from Bio import SeqIO
```

```
from scipy.integrate import quad
from terminaltables import AsciiTable
```

GenBank sitesinden elde ettiğimiz fasta formatındaki dosyamızı SeqIO sınıfının parse metoduyla bir SeqIO iteatörüne çeviriyoruz. SeqIO nesnesi bir fasta dosya içerisinden birden fazla sekans okuyabildiği için bize bir iteratör döndürüyor. Dolayısıyla ilk elemana yani bizim NC_001416 canlımızın sekansına ulaşabilmek için next() metoduyla bir ilerideki sekansı elde etmemiz gerekiyor.

```
pathToFile = "NC_001416.fasta"

fasta_sequences = SeqIO.parse(open(pathToFile), 'fasta')
sequence_obj = next(fasta_sequences)

```

next() metoduyla elde ettiğimiz nesnemizin içerisinden id(NC_001416) ve DNA sekansını sırasıyla id ve sequence değişkenlerine atıyoruz. Daha sonrasında bizden istenen HpaII enziminin sekansını kendi ismiyle bir değişken olarak tanımlıyor ve bu sekansın canlının DNA sekansının içerisinde bulunduğu yerleri RegEx yardımıyla liste tamamlama yöntemi kullanarak ‘h’ adında bir değişkene atıyoruz.

> **Not:** re.finditer metodu ‘sequence’ değişkeni içerisinde ‘HpaII’ enzimini ve her bulduğu yerde re.Match nesnesi döndürür. re.Match nesnesi sekansın başladığı ve bittiği noktaları tutan tuple türünde bir değişken içerir. re.Match nesnesinin sahip olduğu start metoduyla sekansın başladığı değerlere erişilebilir. Kısacası re.finditer metodu, re.match metodu gibi davranan bir iteratordür. Aralarındaki fark re.match tek bir nesne bulur ve geriye döndürür, re.finditer katardaki tüm nesneleri geriye döndürür.

```
id, sequence = sequence_obj.id, str(sequence_obj.seq)
HpaII = 'CCGG'
h = [m.start() for m in re.finditer(HpaII, sequence)]
```

Enzimin parçaladığı noktaların aralarında kalan yerlere fragman denir. Bu fragmanları bulabilmek için yukarıda tanımladığımız ‘h’ dizisinde başlangıç noktalarını belirledik. Enzim, h dizisinin uzunluğu kadar bölümleme yapacaktır. Dolayısıyla h dizisinin uzunluğu kadar bir tekrar söz konusudur. Ancak en son sekansta taşma yaşacağımızdan dolayı bir eksiği kadar döngü tekrarlanır ve döngü bitiminde en son değer için bir ifade daha eklenir.

```
```

```
fragments = ["" for x in range(len(h))]
for i in range(len(h) - 1):
    fragments[i] = sequence[h[i]:h[i+1] - 1]
fragments[-1] = sequence[h[-1]:]
```

Poisson süreci için p-value değerinin hesaplanması gerekir. Burada p-value değerimiz HpaII enziminin algılayabildiği ‘CCGG’ sekansının tüm sekansta ne kadar hangi oranda bulunabileceğinin hesaplanmasıdır. Bunun için önce p(C) ve p(G) değerleri hesaplanır. ‘CC’ sekansının p(CC) değeri, p(C)^2 değerine eşittir. Aynı şekilde G bazı ve GG dimeri içinde geçerlidir. En sonunda da CCGG sekansının oranının elde etmek için elde edilen P(C)^2 ve P(G)^2 değerleri çarpılır ve p-value elde edilir.

```
p_C2 = (len([i.start() for i in re.finditer('C', sequence)]) / (len(sequence)))**2
p_G2 = (len([i.start() for i in re.finditer('G', sequence)]) / (len(sequence)))**2
p = p_C2*p_G2
```

Poisson sürecinin fonksiyonunun betik içerisinde tanımlanması gerekiyor. Kısa bir fonksiyon olduğu için lambda fonksiyonu olarak tanımlamak işimizi görecektir. İstenen aralıklar [0,100), [100,200)...[600,48502) aralıklarıdır. Bu aralıklar da range fonksiyonu ve len fonksiyonu yardımlarıyla tanımlanabilir.

```
poisson = lambda x: p*math.exp(-p*x)
bins = list(range(0, 601, 100)) + [len(sequence)]
```

Beklenti değerlerimizin hesaplanması için yukarıda aralıklarımızı hesapladık. Bins dizimizde yedi değer var dolayısıyla eksiği bize aralık sayısını verecektir. Her aralık için poisson sürecine göre belirli integral değeri yani tüm sekansa oranla o aralığa dahil olacak uzunlukta kaç adet fragman bulunacağı hesaplanır. Hesaplanan değer toplam fragman sayısı ile çarpılarak o aralık için beklenti değeri elde edilir.

```
expecteds = []
for i in range(len(bins) - 1):
    res, err = quad(poisson, bins[i], bins[i+1])
    res = round(res, 3)
    expecteds.append(len(fragments)*res)
```

Daha önceden hesaplamış olduğumuz fragmanlar yardımıyla her aralığa denk gelen gözlem değerlerimizi hesaplamamız gerekir. ‘observations’ dizisi de expecteds ve bins gibi 6 değerli bir dizi olacaktır. Her fragman için dahil olduğu aralık karar yapıları yardımıyla belirlenir. Ve dahil olduğu aralık bulunduğunda, o aralığa denk gelen hücre bir arttırılır ve içerideki döngü kırılarak dışarıda döngüye çıkılır.

```
observations = [0 for i in range(len(bins) - 1)]
for i in fragments:
    for j in range(len(bins) - 1):
        if len(i) >= bins[j] and len(i) < bins[j+1]:
            observations[j] += 1
            break
```

Genom dağılımının beklenti değerlerine yakınlığını ölçmek ve beklentiyle gözlem arasındaki benzerliğini ortaya koyabilmek için X^2 testi kullanacağız. Dolayısıyla her beklenti ve gözlem çifti için X^2 değeri hesaplanmalıdır.

```
chi_square_test = []
for expected, observation in zip(expecteds, observations):
    chi_square_test.append((observation - expected) ** 2 / expected)
```

Terminaltables kütüphanesi ile konsola tablo formatında veri yazdırmak için diziler formata uygun hale getirilmelidir. Uygun hale getirilen diziler zip metoduyla birleştirilir ve bir AsciiTable nesnesi oluşturulur. Son olarak da oluşturulan tablo ekrana yazdırılır.

```
bins = ['Aralıklar'] + [str(bins[i]) + ' - ' + str(bins[i+1]) for i in range(len(bins) - 1  )] +
['------'] + ['Toplam']
observations = ['Gözlemler'] + observations + ['------'] + [round(sum(observations),
3)]
expecteds = ['Beklenen'] + [round(i, 3) for i in expecteds] + ['------'] +
[round(sum(expecteds), 3)]
chi_square_test = ['Ki-kare Test'] + [round(i, 3) for i in chi_square_test] + ['------'] +
[round(sum(chi_square_test), 3)]

table_data = [[a,b,c,d] for a,b,c,d in zip(bins, observations, expecteds,
chi_square_test)]
table = AsciiTable(table_data)
print(table.table)
```

4.