عنوان پروژه : شناسایی و بررسی ژن های استخراج شده از داده های متاژنوم

دانشجو: علی یزدی زاده خرازی - 610798032

استاد راهنما: خانم دکتر وفایی

مقدمه:

روش‌های پیش‌بینی ژن در بیوانفورماتیک ابزارهایی هستند که برای شناسایی و مشخص کردن ژن‌ها در توالی‌های DNA استفاده می‌شوند. به منظور پیش‌بینی دقیق ژن‌ها، این روش‌ها از الگوریتم‌ها و پایگاه‌های اطلاعاتی مختلفی برای تجزیه و تحلیل مناطق کد کننده و غیر کد کننده ژنوم استفاده می‌کنند. استفاده از روش‌های پیش‌بینی ژن برای داده‌های متاژنومیکس برای درک حضور و فراوانی عناصر ژنتیکی مختلف در یک نمونه مشخص مهم است. متاژنومیکس مطالعه مواد ژنتیکی است که مستقیماً از نمونه های محیطی بازیافت می شود. با استفاده از روش‌های پیش‌بینی ژن، محققان می‌توانند ترکیب ژنتیکی یک نمونه را کشف کنند، که سپس می‌توان از آن برای استنباط وجود ارگانیسم‌های خاص، عملکرد ژن‌ها و سایر فرآیندهای بیولوژیکی استفاده کرد.

در این پروژه سعی در استخراج های ژن ها از DNA متاژنومی کرده و پس از ارزیابی آن ها توسط مقایسه با پایگاه داده های ژنی (BLAST)، به بررسی رابطه آنها با پپتیدهای ضد میکروبی می کنیم.

توضیح پکیج های استفاده شده:

در این پروژه از پکیج های زیر به منظور پردازش و تحلیل داده و کار با API ها استفاده شده است.

BIOPYTHON: خواندن و نوشتن توالی ها

GGET: Send BLAST Request

PANDAS: خواندن داده های نتیجه و کار با دیتا فریم

RE: شناسایی الگو ژن ها توسط Regex

OS: کار با فایل ها و سیستم عامل

توضیح توابع تعریف شده:

در این بخش به توضیح توابع تعریف شده در این پروژه می پردازیم. هر کدام از این توابع یک عملکرد مشخص را انجام میدهد که در بخش بعدی با استفاده از این توابع شناسایی ژن ها را انجام می دهیم.

1. تابع find_protein

این تابع یک رشته DNA دریافت می کند و تمامی آن توالی را (تمامی 3 فریم ها) را برای یک الگو مشخص که متشکل از یک کدون شروع (ATG) و یک کدون پایان (TAA|TAG|TGA) که حداقل 30 کدون باهم فاصله دارند، جست و جو میکند و نتایج را برمیگرداند.

این حداقل فاصله باعث میشوند تا توالی های بسیار کوتاه که احتمال پایینی برای ژن بودن دارند را حذف کنیم.

```python
def find_proteins(dna_sequence, min_length=30):
    # regex for finding proteins with ATG as start codon and TAA, TAG or TGA as stop codon excluding the stop codon
    protein_regex = "(ATG)([ATGC]{3}){1,}?(TAA|TAG|TGA)"

    matches = re.finditer(protein_regex, dna_sequence)
    proteins = []
    proteins_index = []
    for match in matches:
        span = match.span()
        # exclude proteins shorter than 100 amino acids
        if span[1] - span[0] > min_length * 3:
            proteins.append(match.group()[:-3])
            proteins_index.append(match.span())
    return proteins, proteins_index
```

2. تابع reverse_complement

این تابع یک رشته DNA دریافت می کند و رشته مکمل آن را برمیگرداند.

```python
def reverse_complement(dna_sequence):
    complement = {"A": "T", "C": "G", "G": "C", "T": "A"}
    return "".join([complement[base] for base in
dna_sequence[::-1]])
```

3. تابع translate_dna_to_protein

این تابع یک رشته DNA دریافت می کند و با استفاده از یک جدول کدون آن را به آمینواسید ترجمه می کند.

```python
def translate_dna_to_protein(dna_sequence):
    codon_table = {
        "TTT": "F",
        "TTC": "F",
        "TTA": "L",
           .
           .
           .
        "GGG": "G",
    }
    protein = ""
    for i in range(0, len(dna_sequence), 3):
        codon = dna_sequence[i : i + 3]
        if codon in codon_table:
            protein += codon_table[codon]
        else:
            protein += "X"
    return protein
```

4. تابع run_blast

این تابع یک رشته آمینواسیدی دریافت می کند و آن را BLAST میکند و نتیجه را در یک فایل CSV ذخیره میکند. در صورت پیدا نکردن نتیجه چاپ می کند که نتیجه ای پیدا نشد و آن توالی یک ژن نبوده.

```python
def run_blast(sequence, seq_id):
    print("Running blast for sequence {}".format(sequence))
    result_handle = gget.blast(sequence, "blastp", "nr")
    print(f"Finished blast for sequence {sequence} with result
{result_handle}")
    # if the result is a dataframe, save it to a file
    if isinstance(result_handle, pd.DataFrame):
        result_handle.to_csv(f"results/{seq_id}.txt",
index=False)
        print(f"Saved result to file results/{seq_id}.txt")
    else:
        print(f"No results found for sequence {seq_id}")
```

مراحل اجرا و نتایج:

مرحله 1: خواندن داده و حذف کردن توالی های کوتاه تر از 40kb

*** این مرحله به خاطر حجم زیاد فایل ها در Google Colab اجرا شده است**

در این مرحله ابتدا پکیج های مورد نیاز را نصب می کنیم:

```unset
#bash
pip install --upgrade --no-cache-dir gdown biopython gget
gdown "14PGwsGuL2ouY-_fv0yrzijGnBMSjREU6"
unrar x /content/y5.final.contigs.rar
```

سپس توالی ها را دانلود و اکسترکت میکنیم:

```python
# load the fasta file using BioPython
from Bio import SeqIO
records = list(SeqIO.parse("/content/y5.final.contigs.fa",
"fasta"))
```

سپس تمامی توالی های کمتر از 40000 باز را حذف میکنیم و بقیه را در یک فایل fasta ذخیره میکنیم:

```python
# filter any sequences that are shorter than 40000 bp and save
the remaining sequences to a new fasta file
filtered_records = [record for record in records if
len(record.seq) > 40000]
SeqIO.write(filtered_records, "contigs_over40kb.fa", "fasta")
```

مرحله 2: استخراج ژن های احتمالی از توالی DNA و توالی مکمل آن

در این مرحله که برای هر کانتیگ تکرار می شود، ژن های احتمالی را از هر رشته و رشته مکمل آن استخراج کرده و در یک لیست میریزیم.

```python
proteins, proteins_index = find_proteins(str(record.seq),
min_length=100)
proteins = [translate_dna_to_protein(protein) for protein in
proteins]
proteins_ids = [f"{record.id}_{i[0]}-forward" for i in
proteins_index]
```

```
reverse_complement_proteins, reverse_complement_proteins_index =
find_proteins(reverse_complement(str(record.seq)),
min_length=100)
reverse_complement_proteins = [translate_dna_to_protein(protein)
for protein in reverse_complement_proteins]
reverse_complement_proteins_ids = [f"{record.id}_{i[0]}-reverse"
for i in reverse_complement_proteins_index]

proteins.extend(reverse_complement_proteins)
proteins_ids.extend(reverse_complement_proteins_ids)
```

مرحله 3: بررسی ژن های احتمالی توسط BLAST

در این مرحله توالی بدست آمده از روش قبل را برای BLAST ارسال میکنیم. برای این هدف از پکیج gget و تابع run_blast استفاده می کنیم.

- یکی از مشکلات این پروژه در این است که API ارائه شده توسط NCBI محدودیت های زیادی برای ارسال درخواست های زیاد دارد و برای ارسال درخواست های موردی طراحی شده در نتیجه ارسال همزمان درخواست (مثلا با استفاده از پکیج asyncio) باعث مسدود شدن ip شما توسط NCBI می شود در نتیجه باید درخواست ها را یکی یکی ارسال کنیم که بسیار زمان بر است. در حالت ایده آل این درخواست ها به یک سرور ایجاد شده توسط خودمان که محدودیتی ندارد ارسال می شود.

Python
```
for protein, protein_id in zip(proteins, proteins_ids):
    run_blast(protein, protein_id)
```

مرحله 4: تحلیل نتایج

در این نتایج BLAST را که در فایل ذخیره کردیم را بررسی میکنیم و تنها ژن هایی را نگه می داریم که حداقل یک نتیجه با e-value کمتر از 0.001 داشته اند.

نمونه نتیجه BLAST:

	Description	Scientific Name	E value	Per. Ident	Acc. Len	Accession
0	transposase [Prevotella sp.]	Prevotella sp.	0.000000e+00	85.44%	340	MBR2185107.1
1	transposase [Prevotella sp.]	Prevotella sp.	0.000000e+00	84.49%	339	MBR6936975.1
2	transposase [Prevotella sp.]	Prevotella sp.	0.000000e+00	83.86%	317	MBQ2216408.1

نمونه فایل fasta نهایی حاوی ژن های تایید شده:

```
Unset
>k141_3522370_3613-forward|0.0|transposase [Prevotella
sp.]|Prevotella sp.
MVVAGRRPVFGEIVGSTKEGGEKPHLKPSPLGKAVLEKEIPKIHHYYPQVDIWQVCLMPD
HLHLIVRINQPLPKGKHLGIIIGAFKGGVSRAWWKLETTVSAASATDPRSPLFEPNYNDH
ILMRDGQLENWKRYLRDNPLRYMMRREYPYLFQRALCVTIGGTRYSAFGNMLLLRQPEKH
HVFFHRRTKGVPTENTEFWQTERQRLISAAQSGDVLVTPGVSECEKRIKKMALEQRLRLI
HVQGEPIGRYWKPERSRFEACAYGSLLILAPWPEDMPTFTSAYERFHYLNRLAAIVCEIS
HTTNVTVQGLRLSNGG
```

بخش دوم (اختیاری) :

شناسایی پپتید هایی که میتوانند هدف Anti Microbial Peptides ها قرار بگیرند.

پپتیدهای ضد میکروبی گروهی از پپتید های کوتاه و غالبا با بار مثبت زیاد هستند که میتوانند باعث مرگ میکروب ها شوند بدون اینکه آسیبی به سلول های بدن ما وارد کنند. در نتیجه با توجه به گسترش مقاومت آنتی بیوتیکی در باکتری ها، توجه ویژه ای به این پپتید ها به عنوان جایگزینی برای آنتی بیوتیک ها می شود. این موضوع از این نظر مهم است که این پپتید ها اکثرا اهداف عمومی در سلول دارند مثلا غشا پلاسمایی را بهم میزنند یا کمپلکس ها درون سلولی را تخریب می کنند، در نتیجه احتمال ایجاد مقاومت به آن ها در باکتری ها به شدت کمتر از آنتی بیوتیک است.

روش مورد استفاده:

طبق بررسی های من توافق مشخصی در این مورد که چه پپتید هایی مورد هدف AMP ها قرار می گیرند وجود ندارد و این پپتید طیف وسیعی از مولکول ها را هدف میگیرند. اما با توجه به اینکه این پپتید ها غالبا بار مثبت زیادی دارند، محتمل است که به پپتید های دارای بار منفی متصل شوند. به این منظور کد زیر پس از خواندن توالی پروتئین های استخراج شده در مرحله قبل، پروتئین های با PI پایین تر از 4 (با بار منفی زیاد در pH سلول) را جدا و در یک فایل دیگر ذخیره می کند.

Python

```python
from Bio import SeqIO
from Bio.SeqUtils.IsoelectricPoint import IsoelectricPoint as IP

# load validated genes
records = list(SeqIO.parse("./results/validated_genes.fa",
"fasta"))

# calculate isoelectric point for each gene
records_pi = [IP(record.seq).pi() for record in records]

# keep genes with high negative charge
records_pi_filtered = [record for record, pi in zip(records,
records_pi) if pi < 5]
print(len(records_pi_filtered))

# save filtered genes
SeqIO.write(records_pi_filtered,
"./results/validated_genes_potential_AMP_targets.fa", "fasta")
```