Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

Курсов проект

Линукс и езици за програмиране в биоинформатиката

Тема: RESTful приложение, което оперира върху данни от ensembl

Изготвил: Янита Докова

ФН: 26198, специалност: БМИ

София

2021

Съдържание

1. Увод

Проектът представлява RESTful приложение на Python, което оперира върху данни от Ensembl. Има три основни възможности, които са свързани с обработката на данни от секвенции. Първата възможност е да връща цялата секвенция на ген по подадено id и всички негови екзони, включително информация за тях. Може също да намери GC content на секвенция и да я върне с разменени нуклеинови киселини аденин и тимин. Също така, по id на ген може да конвертира секвенцията във формати като fasta, x-fasta и multi-fasta.

2. Материали и методи

Приложението е написано на Python версия 3.8 и работи на порт 8080. За Web framework се използва Flask в development режим.

Използвани модули:

- requests – библиотека за правене на HTTP заявки

Приложението комуникира с Ensembl REST API

3. Резултати

Има 2 API endpoints, към които може да се направят HTTP GET заявки

3.1. Намиране на цялата секвенция на гена и всички екзони

Краен резултат:

```
{
    "seq":"GTAGTTGATCTCCGGAGTTTCGCCATGCGGAACTTGGGGGCTTTCGCGGCCCGCGTCGGTGCGGAGTAGCTGCTTT",
    "exons":[
        {"start":38327340,"end":38327509,"id":"ENSE00003819676","seq":"TCCGGAGTTTC"},
        {"start":38727340,"end":38327509,"id":"ENSE00003819676","seq":"GGTGCGGAGTAGC"},
        {"start":38318829,"end":38318987,"id":"ENSE00001025712","seq":"GAACTTGGGGGCTTTCGCGGCCCGCGTCGGTGC"}
    ]
}
```

За получаване на резултата се прави GET заявка към приложението с endpoint `/v1/sequence/gene/<string:id>/`, където `<string:id>` е id на гена.

Правят се две заявки към Ensemble API:

- `/sequence/id/:id` – по подадено id връща json, съдържащ цялата секвенция
- `/lookup/id/:id/?expand=1` – по подадено id връща json, съдържащ всички екзони

За изпращане на http request се използва функцията `get_ensembl_response(path, id, attr)`, която приема като параметри пътя до API endpoint-а, id и атрибути на заявката. Ако заявката е неуспешна се връща подходящ Exception.

```
def get_ensembl_response(path, id, attr):
    ext = path + id + attr
    r = requests.get(server + ext, headers = { "Content-
Type": "application/json"})
    if not r.ok:
        r.raise_for_status()

    return r.json()
```

Резултатът от първата заявка за взимане на секвенцията се записва в променлива `decoded_seq`, а от втората в `decoded_exons`.

Според json отговора, екзоните могат да се вземат от списък с транскрипти или директно. За целта се проверява дали вече съществува такъв списък с транскрипти и ако съществува се обхожда за всеки елемент.

```
    exons = []
    if "Transcript" in decoded_exons:
        for transcript in decoded_exons["Transcript"]:
            exons += get_exons(transcript, decoded_seq["seq"])
    else:
        exons += get_exons(decoded_exons, decoded_seq["seq"])

    result = {
        "seq": decoded_seq["seq"],
        "exons": exons
    }
```

Екзоните се записват в списък exons, а крайният резултат се записва в dictionary result, което съдържа елементи “seq“ и “exons”

За взимането на екзоните се използва отделна функция `get_exons(transcript, seq)`, която приема като параметри транскрипта и секвенцията

```
def get_exons(transcript, seq):
    result = []
    for exon in transcript["Exon"]:
        result.append(
            {
                "start": exon["start"],
                "end": exon["end"],
                "id": exon["id"],
                "seq": seq[exon["start"] - transcript["start"] :
                                        exon["end"] - transcript["end"]]
            }
        )
```

```
    return result
```

Функцията минава по всеки екзон и го добавя към общ списък с екзони. Добавя се dictionary, съдържащ елементите - начало, край, id и секвенция. Секвенцията на екзона се изчислява чрез изваждане на старта и края на екзона от старта и края на транскрипта и взимането на получената част от цялата секвенция. Върнатият резултат от функцията е списък от dictionary елементи, съдържащи информация за всеки екзон.

3.2. Намиране на цялата секвенция, GC content и секвенцията с разменени нуклеинови киселини аденин и тимин

Крен резултат:

```
{
  "gc_content": 41,
  "seq":
"AGGATGGAAGACTTTTTATGTGGGGTGACAATTCCGAAGGGCAAATTGGTTTAAAAAATGTAAGTAATGTCTGTGTCCCTCAGCAAGTGACCATTGGG
AAACCTGTCTCCTGGATCTCTTGTGGATATTACCATTCAGCTTTTGTAACAA",
  "swap_sequence":
"TGGTAGGTTGTCAAAAATAGAGGGGAGTCTTAACCGTTGGGCTTTAAGGAAATTTTTTAGATTGATTAGACAGAGACCCACTGCTTGAGTCCTAAGGG
TTTCCAGACACCAGGTACACAAGAGGTATAATCCTAACTGCAAAAGATTCTT"
}
```

За получаване на резултата се прави GET заявка към приложението с endpoint `/v1/sequence/gene/<string:id>/?gc_content=true&swap=A:T`, където `<string:id>` е id на гена.

Прави се заявка към Ensemble API:

- `/sequence/id/:id` – по подадено id връща json, съдържащ цялата секвенция

```
    gc_content_arg = request.args.get('gc_content')
    swap_arg = request.args.get('swap')
    if gc_content_arg == "true" and swap_arg == "A:T":
        result = {
            "seq": decoded_seq["seq"],
            "gc_content": gc_content(decoded_seq["seq"]),
            "swap_sequence": swap_a_t(decoded_seq["seq"])
        }

        return result
```

Прави се проверка дали атрибутите gc_content и swap са подадени в заявката. Ако са подадени се извикват помощни функции за пресмятане на процентното съдържание на гуанин и цитозин и намирането на секвенцията с разменените аденин и тимин. Върнатият резултат се записва в dictionary.

Функцията за намиране на процентното съдържание на цитозин и гуанин `gc_content(seq)` приема като параметър секвенцията и връща сборът на броя бази цитозин и гуанин, разделен на общия брой бази и умножен по 100.

```
def gc_content(seq):
    return round((seq.count('C') + seq.count('G')) / len(seq) * 100)
```

Функцията за връщане на секвенция с разменени аденин и тимин `swap_a_t(seq)` обхожда цялата секвенция и проверява стойността на текущата база. Ако базата е тимин добавя аденин в лист с резултата, ако е аденин добавя тимин, иначе запазва текущата стойност. Крайният резултат се връща под формата на стринг.

```
def swap_a_t(seq):
    new_seq = []
    for base in seq:
        if base == 'A':
            new_seq.append('T')
        elif base == 'T':
            new_seq.append('A')
        else:
            new_seq.append(base)

    return ''.join(new_seq)
```

3.3. Конвертиране на секвенция във fasta, x-fasta или multi-fasta формат

Краен резултат:

```
{
  "id": ">ENSE00001025715.1 chromosome:GRCh38:X:38317316:38317465:-1",
  "seq":
"AGGATGGAAGACTTTTTATGTGGGGTGACAATTCCGAAGGGCAAATTGGTTTAAAAAATGTAAGTAATGTCTGTGTCCCTCAGCAAGTGACCATTG
GGAAACCTGTCTCCTGGATCTCTTGTGGATATTACCATTCAGCTTTTGTAACAA"
}
```

За получаване на резултата се прави GET заявка към приложението с endpoint `/v1/sequence/<string:id>/`, където `<string:id>` е id на гена.

Прави се заявка към Ensemble API:

- `/sequence/id/:id` – по подадено id връща json, съдържащ цялата секвенция

Взима се стойността на аргумента content-type и се проверява дали е fasta, x-fasta или multi-fasta. Ако не е някоя от тези стойности се връща съобщение.

Конвертиране във fasta формат – използва се функция `convert_to_fasta(decoded_seq)`, която приема като аргумент резултатът от заявката към Ensembl API. Крайният резултат е dictionary, което съдържа id и seq като id се създава от конкатенация на началния символ “>”, id на секвенцията, версия и описание.

```
def convert_to_fasta(decoded_seq):
    desc = ">" + decoded_seq["id"] + "." + str(decoded_seq["version"]) +
           " " + decoded_seq["desc"]
    result = {
        "id": desc,
        "seq": decoded_seq["seq"]
        }

    return result
```

Конвертиране в x-fasta формат – използва се функция `convert_to_xfasta(decoded_seq)`, която работи подобно на горната с единствената разлика, че стойността id съдържа “>” и id на секвенцията

```
def convert_to_xfasta(decoded_seq):
    desc = ">" + decoded_seq["id"]
    result = {
        "id": desc,
        "seq": decoded_seq["seq"]
        }

    return result
```

Конвертиране в x-fasta формат – използва се функция `convert_to_multifasta(decoded_seq)`, която работи подобно на останалите 2, но с разликата, че обхожда множество секвенции и ги обединява в един лист от fasta формати

```
def convert_to_multifasta(decoded_seq):
    result = []

    for seq in decoded_seq["seq"]:
        result += convert_to_fasta(seq)

    return result
```

4. Заключение

RESTful приложението има предимството, че може да се пусне на която и да е машина и да се използва от множество клиенти, без да се налага те да инсталират нещо допълнително при тях. За използването му е достатъчно да имат само браузър.

5. Източници

https://rest.ensembl.org/

https://rest.ensembl.org/documentation/info/sequence_id

https://rest.ensembl.org/documentation/info/lookup

https://flask.palletsprojects.com/en/1.1.x/