Projet 2 - modélisation de matrices de comptage (PFM)

L'objectif du second projet est de modéliser des classes permettant de manipuler des PFM : https://en.wikipedia.org/wiki/Positio

Ces matrices permettent de représenter un motif sur une séquence. Elles ont été particulièrement inventées pour modéliser des sites de fixation de facteurs de transcription. Il existe des bases de données pour ces matrices : https://jaspar.genereg.net/

A partir d'un ensemble de sites connus pour un motif, c'est-à-dire de séquences d'ADN toutes de même longueur (disons l) qui sont connues pour être ce motif, on transforme ces séquences en un objet condensé où pour chaque position (de 1 à l) on trouve 4 nombres représentants la propension de chaque nucléotide à être présent à cette position. L'article Wikipedia cité plus haut explique le processus de transformation qui part des sites pour arriver à une PFM puis une PSSM puis une PWM, mais **ce n'est pas ce qui nous intéresse ici**.

Nous allons simplement nous intéresser aux matrices de comptage, c'est-à-dire compter pour chaque position (de 1 à l) le nombre d'occurrences de chaque nucléotide d'un ensemble de sites connus.

Prenons un exemple. Imaginons avoir en entrée les sites suivants :

```
ATTGCGGTC
ATTGCGGTT
TTTGCGGTC
ATTGTGGTT
ACTGCGGTT
TTTGTGGTC
CTTGTGGTC
ACTGTGGTT
CCTGCGGTT
ATTGCGGTA
TATGCGGTT
GCTGCGGTT
AATGTGGTA
TTTGTGGAC
AAAGTGGTT
TTCTTGGTT
TCAGTGGGT
```

la matrice de comptage associée sera (si les lignes sont supposées indexées par A, T, C et G) :

```
 8  3  2  0  0  0  0  1  2
 6  9 14  1  9  0  0 15 10
 2  5  1  0  8  0  0  0  5
 1  0  0 16  0 17 17  1  0
```

indiquant qu'en colonne 1 il y a 8 A, 6 T, 2 C et 1 G. On vérifie bien qu'il y a 17 nucléotides dans chaque colonne puisqu'il y a 17 sites ici, et qu'on a 9 colonnes, puisque chaque site comporte 9 nucléotides.

On souhaite pour ce projet créer une modélisation des PFM dans une classe.

Format d'entrée

Les sites seront fournis dans un fichier au format multifasta, c'est-à-dire la concaténation de fichiers fasta. Ici on supposera que chaque séquence est toujours donnée sur une ligne.

Pour l'exemple :

```
> site1
ATTGCGGTC
> site2
ATTGCGGTT
> site3
TTTGCGGTC
> site4
ATTGTGGTT
> site5
ACTGCGGTT
```

```
> site6
TTTGTGGTC
> site7
CTTGTGGTC
> site8
ACTGTGGTT
> site9
CCTGCGGTT
> site10
ATTGCGGTA
> site11
TATGCGGTT
> site12
GCTGCGGTT
> site13
AATGTGGTA
> site14
TTTGTGGAC
> site15
AAAGTGGTT
> site16
TTCTTGGTT
> site17
TCAGTGGGT
```

Les séquences pourront être écrites en majuscules ou minuscules, peu importe. Le nom des sites ne nous intéresse pas.

On se dotera d'une classe `Multifasta` qui crée une représentation d'un ensemble de sites stockés au format multifasta et founit une méthode `sequences` qui retourne la liste des séquences sous forme de chaînes de caractères (voir exemple plus bas).

Calculs avec une PFM

On souhaite disposer de fonctionnalités nous permettant de manipuler les PFM. Une PFM pourra être construite à partir d'une liste de sites, chaque site étant une chaîne de caractère représentant une séquence d'ADN. Toutes les chaînes devront bien entendu être de la même longueur.

Avoir des informations basiques

Calculer la longueur, obtenir une représentation sous forme de chaîne de caractères, tester l'égalité de deux PFM (méthode `__eq__`) : deux PFM seront égales si elles ont exactement les mêmes quantités sur chaque colonne.

Avoir des informations sur les colonnes

Savoir si une colonne est conservée (toujours la même base), avoir la liste des colonnes conservées, avoir le nucleotide le plus frequent d'une colonne.

Créer un consensus

Obtenir une séquence de la même longueur que les sites qui contienne à chaque position le nucléotide le plus fréquent. Obtenir un consensus mou : comme le consensus ne sait pas gérer les ambiguités, on s'interesse aussi à extraire un consensus qui permet d'indiquer ce qu'il est possible de trouver à chaque position. On adopte donc la notation entre [ ] pour indiquer les différentes possibilités à chaque position (voir exemple).

Modifier une PFM

Ajouter un nouveau site (condition, bien sûr le nouveau site doit avoir la bonne longueur), ceci a pour effet de modifier la PFM. Concaténer deux PFM (condition : les deux doivent avoir le même nombre de sites), ceci a pour effet de créer une nouvelle PFM (on pourra utiliser `__add__` qui est reconnu comme l'opérateur d'addition).

Recherche avec une PFM

Dans le `main` qui sera produit, on écrira une fonction `search` qui prend en paramètre une PFM et une séquence d'ADN et recherche les positon où le consensus mou de la PFM peut apparaître.

Exemple d'utilisation

Multifasta

Le code suivant :

```
f = Multifasta('mes_sites.fasta')
print(f.sequences())
```

produira l'affichage suivant :

```
[ 'ATTGCGGTC', 'ATTGCGGTT', 'TTTGCGGTC', 'ATTGTGGTT', 'ACTGCGGTT', 'TTTGTGGTC', 'CTTGTGGTC', 'ACTGTGGTT
```

PFM

Le code suivant :

```
l = [ 'ATTGCGGTC', 'ATTGCGGTT', 'TTTGCGGTC', 'ATTGTGGTT', 'ACTGCGGTT', 'TTTGTGGTC', 'CTTGTGGTC', 'ACTGT

m = PFM(l)

print(m)
print(len(m))
print(m.is_conserved(5))
print(m.is_conserved(6))
print(m.is_conserved(7))
print(m.is_conserved(8))
print(m.conserved_columns())
print(m.most_frequent_base(1))
print(m.consensus())
print(m.weak_consensus())
m.append("ATTAGGATA")
print(m)
print(m.conserved_columns())

print(m+m)
```

produira l'affichage suivant :

```
A  8  3  2  0  0  0  0  1  2
T  6  9 14  1  9  0  0 15 10
C  2  5  1  0  8  0  0  0  5
G  1  0  0 16  0 17 17  1  0
9
False
True
True
False
[6, 7]
A
ATTGTGGTT
[ATCG][ATC][ATC][TG][TC]GG[ATG][ATC]
A  9  3  2  1  0  0  1  1  3
T  6 10 15  1  9  0  0 16 10
C  2  5  1  0  8  0  0  0  5
G  1  0  0 16  1 18 17  1  0
[6]
A  9  3  2  1  0  0  1  1  3  9  3  2  1  0  0  1  1  3
T  6 10 15  1  9  0  0 16 10  6 10 15  1  9  0  0 16 10
C  2  5  1  0  8  0  0  0  5  2  5  1  0  8  0  0  0  5
```

```
G  1  0  0 16  1 18 17  1  0  1  0  0 16  1 18 17  1  0
```

Fonction `search`

Le code suivant :

```
print(search(m,"GGGAACTTGGTCAT"))
print(search(m,"GCCAGCGGAAGGAACTTGCTCAT"))
print(search(m,"GGGAACTTGATCAT"))
```

produira l'affichage suivant :

```
[ 3 ]
[ 1, 12]
[ ]
```

Bonus

Dans un premier temps je vous conseille de réaliser ce projet en admettant que tout se passe toujours bien, i.e. les paramètres fournis aux méthodes sont corrects, les fichiers mutlifasta sont bien formatés, etc.

Si vous avez le temps, vous pourrez ensuite ajouter des vérifications dans les méthodes.

Par exemple vérifier que lorsqu'on passe un numéro de colonne dans une méthode, le numéro est valide. On pourra faire cela grâce aux instructions `assert` qui permettent de vérifier qu'une expression booléenne est vraie. Si l'expression est vraie, il ne se passe rien, sinon un message d'erreur est affiché et le programme s'interrompt.

Par exemple:

```
num = 2
assert num < 3, "{} n'est pas inférieur à 3".format(num)
# rien ne se passe
```

```
num = 5
assert num < 3, "{} n'est pas inférieur à 3".format(num)
# une erreur est déclenchée
```

```
---------------------------------------------------------------------------

AssertionError                            Traceback (most recent call last)

/var/folders/ms/6jqfpr_d6zg553c5n__mw6wc0000gn/T/ipykernel_80504/1967759600.py in <module>
      1 num = 5
----> 2 assert num < 3, "{} n'est pas inférieur à 3".format(num)
      3 # une erreur est déclenchée


AssertionError: 5 n'est pas inférieur à 3
```

Rendu attendu

Dépôt sur PROF (https://prof.fil.univ-lille1.fr/) pour le 25/11/2021 à 13h une archive zip contentant dans un dossier portant votre nom de famille :

- un script Python nommé `main_PFM.py`
- une classe `Multifasta` et une classe `PFM` respectivement dans les fichiers `multifasta.py` et `PFM.py`
- un fichier README.txt
- un fichier exemple.fasta contenant un ensemble de sites test

Le fichier README.txt indique :

- ce qui est fonctionnel,
- quelques mots expliquant le principe général de votre programme,
- comment lancer le programme (i.e. des lignes de commande que je peux copier-coller directement)
- un exemple et un commentaire du résultat

Le fichier `main_PFM.py`contiendra : - la fonction `search` - la partie main qui prendra en entrée un nom de fichier contenant un les sites d'une PFM au format multifasta, un nom de fichier contenant une séquence au

format fasta, et produira un affichage de la PFM, des sites conservés, du consensus mou et la liste des positions d'occurrence de la PFM dans la séquence fournie

Il faudra pour l'ensemble des méthodes créées donner des commentaires *docstrings* respectant la norme PEP257 décrite dans le cours en ligne chapitre 15.

Le code exemple donné plus haut doit produire exactement les mêmes sorties.

Pour ce travail, il est rigoureusement interdit d'utiliser du code existant par ailleurs.