Traitement de données génomiques en Perl Orienté Objet et BioPerl

IUT Génie Biologique, Option Bioinformatique (2e année)

Bérénice Batut

Février 2016

Introduction

Les séances de TP vont s'organiser autour d'un mini-projet qui vous permettra de vous familiariser avec le traitement de séquences en utilisant des classes, des références et BioPerl.

Principe du projet

L'objectif du projet est traiter très simplement un jeu de données métagénomiques, c'est-à-dire un jeu de données contenant de nombreuses séquences issus d'organismes différents. Les traitements, effectués dans le cadre de ce projet, passent par un contrôle de la qualité des séquences.

Ce projet vous permettra de vous familiariser avec ce type de données pour mieux les utiliser dans le projet du cours de Métagénomique et d'avoir une vue rapide de ce qu'il y a derrière les outils de traitement de ce type de données.

Organisation

Pour ce projet, vous vous mettrez par binôme. Les deux membres du binôme doivent participer au projet, pour les codes mais aussi la rédaction du compte-rendu. D'ailleurs, il serait bien qu'au moins un des membres du binôme amène son ordinateur personnel. De plus, la personne qui "code" et la personne qui "rédige" ne doivent pas être toujours les même.

L'ensemble du projet devra être envoyé par mail (berenice.batut@udamail.fr) sous forme d'une archive (`zip` ou `tar.gz`), au plus tard le 01/03/2016 à 20h00.

La note du projet portera sur le compte-rendu, les codes fournis et le suivi des instructions (2 points).

Instructions

L'organisation est importante tout le temps, même en informatique et en bioinformatique. En particulier dans un projet bioinformatique, il est important de bien centraliser dans un seul dossier tous les fichiers concernant le projet afin de garder une trace. De plus, les fichiers au sein de ce dossier doivent être organisé de façon méthodique afin de faciliter votre travail mais aussi celui de vos collaborateurs.

Ainsi, pour ce projet, tous les fichiers sont rassemblés dans un dossier comprenant :

- Un dossier `data` pour toutes les données
- Un dossier `results` pour les résultats des analyses (graphiques, fichiers générés, ...)

- Un dossier `src` pour le script principal (nommé `sequence_processing.pl`) et les modules Perl‘ développés
- Un dossier `doc` pour les notes relatives au projet, en particulier le compte-rendu

Le compte-rendu, dans le cadre de ce projet, correspond à un “cahier de notes” avec les différentes étapes de l'analyse (graphiques, résultats, échecs, ...). En effet, quand on travaille sur un projet, tenir un tel “cahier” permet de garder une trace de ce qui a été fait, ce qui a marché pour pouvoir expliquer la démarche, faciliter la reprise du travail par quelqu'un, ... Dans ce “cahier”, vous noterez les différentes étapes, les grandes lignes des codes, les méthodes particulières utilisées (et pourquoi), les graphiques, les liens vers les fichiers. Pour faciliter la rédaction, un modèle de cahier en `markdown` doit être utilisé. Il est disponible sur bebatut-edu.github.io.

L'organisation et la collaboration passent aussi par le rédaction de codes “propres”. Un code “propre” permet de faciliter la prise en main du code par tout nouvel arrivant, mais aussi par vous plus tard. Ecrire un code “propre” implique de suivre quelques règles

- Documenter le code (avec POD en Perl)
- Faire attention à l'indentation
- Ne pas mettre trop de caractères par ligne (généralement, moins de 80)

Deux modèles sont disponibles sur bebatut-edu.github.io : un modèle pour les classes et un modèle pour les scripts. Ils doivent être utilisé dans ce projet. Ces modèles intègrent de la documentation qui doit être mise à jour régulièrement (visualisable avec `perldoc`).

Pour conserver l'enchainement des analyses et les paramètres choisis, toutes les commandes utilisées seront exécutées depuis un script `Perl` principal nommé `sequence_processing.pl`. Cependant, pour éviter de réexécuter toutes les commandes à chaque exécution, il faut penser à mettre des conditions de test pour chaque étape.

Récupération et première exploration des données

Les données à utiliser par chaque binôme sont disponibles sur bebatut-edu.github.io. Cependant, pour garder une trace des données téléchargées, toutes les commandes utilisées pour récupérer et transformer les données sont exécutées depuis le script principal.

Quelques pistes pour la récupération des données :

- Le téléchargement de données se fait avec la commande `wget <link>` (`<link>` correspondant au lien récupéré en faisant `Clic droit->Copier l'adresse du lien` dans votre navigateur sur le lien à télécharger)
- Les données doivent être déplacées dans le répertoire `data`
- L'extraction des données avec l'extension `gz` se fait en utilisant la commande `gunzip path/to/file`
- L'exécution de commandes systèmes dans `Perl` se fait en les invoquant avec `system "command"`

Nous souhaitons avoir quelques informations sur le fichier et son contenu (la taille, le nombre de ligne dans le fichier, ...). Ces informations doivent être affichées à l'écran lors de l'exécution du script principal. Pour la taille d'un fichier, vous devez utiliser la classe `File::stat` avec la méthode `size`, qui renvoie la taille en octets. Pour le nombre de ligne, vous écrivez une méthode `get_line_number` qui ouvre le fichier et lit ligne par ligne jusqu'à la fin en comptant les lignes. Vous pouvez vérifier le bon fonctionnement de votre méthode en exécutant `wc -l path/to/file` depuis un terminal.

Le fichier contenant les séquences est au format `fastq`. Chaque séquence est ainsi représentée sur 4 lignes. En utilisant la méthode `get_line_number`, on souhaiterait obtenir le nombre de séquences dans le fichier, grâce à une méthode `get_sequence_number`.

Création d'une classe `Sequence`

Pour gérer les séquences contenues dans le fichier de données, nous souhaitons créer une classe `Sequence`, en s'inspirant de la classe `Bio::Seq`. Ainsi, dans un premier temps, les objets `Sequence` ont comme propriétés:

- Sa séquence à proprement parler (nommé `SEQUENCE`)
- Son identifiant (nommé `ID`)
- Sa longueur (nommé `LENGTH`)

Le constructeur de la classe `Sequence` prend en argument un objet `Bio::Seq` et extrait les informations utiles pour remplir les propriétés d'un objet `Sequence`. Vous pouvez tester ce constructeur sur la première séquence du fichier de données (lue grâce à `Bio::SeqIO`), en utilisant `Data::Dumper`.

Une classe fonctionne mieux quand elle a des accesseurs et des modifieurs, c'est-à-dire des méthodes permettant d'accéder aux propriétés pour les récupérer (accesseurs) ou les modifier (modifieurs). Vous ajoutez donc à votre classe `Sequence` les méthodes nécessaires (dont les noms démarrent par `get` pour les accesseurs et par `set` pour les modifieurs) pour chaque propriété et vous les testez dans le script principal. Que penser d'un modifieur pour la propriété de longueur? Quelles sont les différentes possibilités?

Parcours d'un fichier de séquences

Nous souhaitons maintenant parcourir le fichier avec les séquences et enregistrer les séquences dans des objets `Sequence`, qui seront stockés dans un objet `Sequences`. Les objets de la classe `Sequences` ont deux propriétés : le tableau des objets `Sequence` (nommé `SEQUENCES`) et le nombre de séquences enregistrées (nommé `SEQUENCE_NUMBER`).

Dans un premier temps, nous aimerions savoir si le nombre de séquences calculé précédemment correspond bien au nombre de séquences dans le fichier. Pour cela, vous écrivez une méthode `get_real_sequence_number` qui prend en argument le chemin vers le fichier, parcourre ce fichier avec un objet `Bio::SeqIO` en comptant le nombre de séquences (dans le script principal).

Dans un second temps, nous aimerions construire la classe `Sequences`. Le constructeur de la classe prend en argument le chemin vers un fichier de séquences. Il initialise un tableau vide, parcourt le fichier avec un objet `Bio::SeqIO` et pour chaque séquence, créé un objet `Sequence` et enregistre cet objet dans le tableau. Ainsi, le tableau contiendra autant de cases qu'il y a de séquences. L'autre propriété de la classe est le nombre de séquences, qui est compté pendant le parcours du fichier. Vous devez construire cette classe, en pensant aussi aux accesseurs (pas aux modifieurs ici). Pensez bien à tester la classe et les différentes méthodes.

Le fichier de séquences est au format `fastq` et contient donc, pour chaque séquence, sa séquence nucléotidique mais aussi la séquence des scores de qualité pour chaque nucléotide. Jusqu'à présent, nous avons considéré les séquences comme des objets `Bio::Seq`. Or ce type d'objet ne peut contenir de score de qualité. Afin de pouvoir récupérer correctement la séquence des scores de qualité, il faut déjà vérifier quel type d'objet est généré pour chaque séquence par `Bio::SeqIO`. Pour cela, vous récupérez la première séquence du fichier avec `Bio::SeqIO` et vous affichez l'objet avec `Data::Dumper`. Comment récupérer la séquence des scores de qualité? Quel est le type de données correspondant? Pourquoi?

Pour conserver le score qualité, vous complétez la classe `Sequence` en ajoutant une nouvelle propriété pour le score de qualité, nommé `QUALITY` (penser à l'accesseur).

Contrôle de la qualité des séquences

Lorsque des données brutes sont reçues, elles ont souvent besoin d'un traitement de qualité, à la fois pour éliminer les petites séquences qui portent peu d'information utile et aussi pour éliminer les portions des séquences où les scores de qualité sont faibles.

Contrôle de la longueur des séquences

La première chose à vérifier est la longueur des séquences. En particulier, vérifier si toutes les séquences ont la même longueur. Pour vérifier cela, nous souhaitons enregistrer les longueurs des séquences dans un fichier texte ainsi que l'identifiant de la séquence.

Dans la classe `Sequences`, vous écrivez une méthode `write_sequence_length` qui prend en argument le chemin vers un fichier `txt` vierge (nommé `seq_length_before_quality_treatment.txt` dans le dossier `data`) et qui écrit dedans, pour chaque séquence dans le tableau `SEQUENCES` (1 séquence par ligne), l'identifiant et la longueur de la séquence (séparés par une tabulation `\t`):

```
SRR029687.1     229
SRR029687.2     239
SRR029687.3     254
SRR029687.4     275
SRR029687.5     261
```

En ouvrant le fichier généré, que pouvez-vous dire de la longueur des séquences?

Il est difficile de ressortir une tendance en étudiant directement le fichier généré. Une technique pour visualiser les longueurs des séquences est de générer un histogramme. Pour cela, vous pouvez utiliser le script `R plot_sequence_length.R` disponible sur bebatut-edu.github.io en l'exécutant avec la commande (depuis le script principal)

```
Rscript path/to/plot_sequence_length.R
path/to/seq_length_before_quality_treatment.txt
path/to/results/seq_length_before_quality_treatment.pdf
```

Que pouvez-vous dire des longueurs des séquences? Que pensez-vous des séquences avec moins de 60 paires de bases? Que pensez-vous qu'il faudrait faire?

Contrôle des scores de qualité

Le score de qualité d'une base représente la probabilité d'erreur d'identification de cette base. Lorsque le score d'une base est faible, la probabilité d'erreur est forte et nous pouvons considérer que la base concernée ait été potentiellement mal identifiée.

Les scores de qualité ne sont pas homogènes le long de séquences. Nous souhaitons vérifier si cela est vrai aussi sur nos séquences. Pour cela, nous avons besoin d'un tableau avec une séquence par ligne (sans l'identifiant) et un score de qualité de base par colonne. Ainsi, la cellule à la colonne y et à la ligne x contient le score de qualité de la séquence x à la position y.

Cependant, comme observé précédemment, toutes les séquences n'ont pas la même longueur. Le nombre de colonne dans le fichier correspondra alors à la longueur maximale observée dans les séquences. Pour récupérer cette valeur, vous écrivez une méthode `compute_max_sequence_length` dans la classe `Sequences` qui parcoure toutes les séquences et retourne la longueur maximale des séquences.

Dans la classe `Sequences`, vous écrivez ensuite une méthode `write_quality_scores`, qui prend en argument le chemin vers un fichier `txt` de sortie. Cette méthode récupère la longueur maximale des séquences, parcoure les séquences. Pour chaque séquence, la longueur est récupérée et pour chaque base, le score de qualité est écrit dans le fichier (avec une tabulation pour séparer les colonnes). Les dernières colonnes (entre la longueur de la séquence et la longueur maximale des séquences) sont remplies avec "NA":

```
27 28 28 34 27 27 27 28 27 36 32 13 26 34 28 33 25 37 33 15 28 28 28
29 9 33 25 35 28 27 27 27 35 31 11 27 27 27 34 27 28 27 34 27 NA NA
```

Depuis le script principal, vous générez un graphique en utilisant le script `R plot_quality_scores.R` (disponible sur bebatut-edu.github.io). Que pensez-vous des scores de qualité? Sont-ils homogènes le long des séquences? Quel seuil de score de qualité choisisseriez-vous?

Nous choisissons de fixer de conserver les bases dont le score de qualité est supérieur à 15 en coupant les séquences à droite lorsque le score est inférieur au seuil fixé. A quel probabilité le seuil fixé correspond-il?

Pour couper les séquences en fonction du seuil fixé, vous créez une méthode `eliminate_too_bad_bases` dans la classe `Sequence`. Cette méthode prend en argument le seuil de score des bases à conserver. Cette méthode créé une séquence vide, un tableau vide pour les scores de qualité à conserver et initie une nouvelle longueur. Pour remplir la séquence et le tableau des scores de qualité, la méthode parcoure les scores de qualité initiaux. Tant que le score de qualité est supérieur au seuil, les bases sont ajoutées à la séquence, leur score au tableau et la longueur incrémentée. Lorsque le score d'une base est inférieure au seuil défini, les bases suivantes ne sont pas ajoutées (la boucle est arrêtée). Les nouvelles séquences, scores de qualité et longueurs remplacent les précédents.

Pour appliquer cette méthode à toutes les séquences, vous écrivez une méthode dans la classe `Sequences` qui parcoure les séquences en appliquant la méthode `eliminate_too_bad_bases` avec le seuil de score en argument.

Il faut ensuite vérifier les résultats de ce traitement en générant un nouveau fichier avec les scores de qualité, ainsi que le graphique généré par `plot_quality_scores.R`.

Contrôle de la longueur des séquences (2)

Après ce traitement de séquences en les coupant en fonction du score de qualité, les longueurs et leur distribution ont changé. Il faut regéner le graphique précédant après avoir généré le fichier `txt seq_length_after_quality_treatment.txt`.

Que pouvez-vous dire par rapport au graphique précédent?

Nous souhaiterions garder que les séquences dont la taille est supérieur à 60 paires de bases. Pour cela, dans la classe `Sequences`, vous écrirez une méthode `eliminate_too_small_squences` qui prend en argument la taille minimum des séquences à conserver. La méthode créé un tableau vide, le remplit avec les séquences à conserver (car elles respectent la taille minimale des séquences) et ensuite remplace la propriété `SEQUENCES` par le nouveau tableau et met à jour `SEQUENCE_NUMBER`.

Une fois ce traitement effectué, vous pouvez vérifier que toutes les séquences ont une taille supérieure à 60 paires de bases en générant un nouveau fichier `seq_length_after_length_elimination.txt` et un nouveau graphique `seq_length_after_length_elimination.pdf`.

Extraction des séquences obtenues

Pour de futurs traitements des séquences, nous souhaitons exporter les séquences conservées dans un fichier de séquences au format `fasta`.

Dans la classe `Sequences`, vous écrivez une méthode `extract_sequences` qui prend en argument le chemin vers un fichier de sortie et qui remplit ce fichier avec les séquences conservées et leur identifiant.

Discussion

A la fin d'un projet, il est intéressant de prendre du recul vis à vis de ce qui a été fait. Pour cela, faites un schéma récapitulatif des différents traitements effectué.

Que pensez-vous de tout ce qui a été fait pendant ce projet? Auriez-vous fait les choses différemment? Auriez-vous choisi des paramètres différents? Quels autres traitements auriez-vous effectués sur les séquences?

Pour ce projet, nous avons utilisé différentes classes et un script principal. Faites un schéma récapitulatif des différentes classes, avec les différentes méthodes implémentées et les liens entre les classes mais aussi les liens avec le script principal.