Rapport d'intégration de données hétérogènes

Drosophila melanogaster

Par Berlin Tristan, le 21/10/2019

UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université de Toulouse

Introduction

Dans le cadre de l'UE Intégration de Données Hétérogènes, il nous a été demandé de réaliser un projet en autonomie. Celui-ci a pour objectif de nous former à la création de processus automatique en autonomie par le biais de la thématique de la recherche d'enrichissement. En effet, grâce à des moyens tels que la Gene Ontology (GO), que 'on a déjà vu l'année précédente, il est aisé de pouvoir annoter des gènes par rapport à leur fonction : les GO-Terms.

Concrètement les objectifs du projet étaient de :

1) Créer un script capable de donner les annotations directs et implicites d'un organisme par la GO
2) Modifier le script *blastset.py* pour y intégrer deux nouvelles fonctions de calculs
3) Comparer les différentes méthodes de calculs du script
4) Faire une comparaison des fonctions de regroupement

Le plan suivra donc ces différents objectifs : leur problématique, les solutions envisagées, la méthode utilisée et enfin le résultat.

Enfin, quelques mots sur l'organisme choisi pour le projet : *Drosophila melanogaster* (ou la mouche du vinaigre/fruit fly). Petit insecte très reconnaissable pour quiconque a fait de la biologie, cette mouche est un modèle de référence dans l'étude de la génétique et de la biologie du développement (et dans la science en général) car les effets phénotypiques des mutations de gènes se retrouvent très facilement sur cet organisme. En effet, elle ne possède que quatre paires de chromosomes (dont une sexuelle), son cycle de génération est relativement court (deux semaines) et des techniques de transformations génétique ont été rapidement mis en place. Venant d'un cursus avec une forte composante de génétique et biologie du développement, c'est naturellement que j'ai choisi cet organisme avec lequel j'ai longtemps travaillé.

Intégration et préparation des données

Dans cette partie, nous allons nous concentrer sur la réalisation du premier objectif. Celui-ci était de réaliser un script capable de donner les annotations directs et implicites : une liste des gènes annotés par un même terme (ou un de ses descendants pour l'implicite). Les sorties de ce script seront utilisées dans la suite du projet.

Avant de voir la solution concrètement apportée, une petite visualisation des données utilisées. J'ai récupéré le fichier *17.D_melanogaster.goa* (dossier Données) dans la base de données de l'EBI *GO Annotation* (version Gaf : 2.1). Celui-ci ne possède pas moins de 238273 *gene products*. Cependant, une grande partie de cette liste est le même gène pour des termes Go différent. Au final, il y a 19770 gene products.

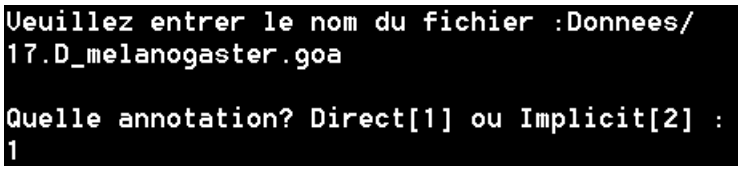
Pour le script, j'ai choisi le langage de programmation Python (version 3) qui permet le traitement de fichier facilement et avec un temps d'exécution relativement rapide (bien que cela ne soit influencé par la programmation elle-même. Dès le début, j'ai choisi d'utiliser la librairie que nous avions élaboré l'an passé pour le traitement de la Gene Ontology (*GeneOntology.py*). Cependant, je voulais d'abord exploiter les fonctions geneProducts de celle-ci, mais je passais plus de temps à faire des aller-retours entre les scripts que j'ai choisis de partir du départ. Le script final est *annotation.py.*

La méthode utilisée est celle-ci :

1) Choisir le fichier .goa que l'on veut annoter (Input)
2) Choisir si on veut faire du direct ou de l'implicite
3) Charger la Gene Ontology inversé (load_OBOMD) pour avoir les relations de descendances
4) Charger le .goa sur la GO (load_GOAMD)
5) Dans un fichier externe, pour chaque sommets « GOTerm » du graph, parcourir ses arrêtes direct et si le sommet relié est un « geneProduct » le garder dans un dictionnaire
6) Optionnel pour l'implicite : faire ça pour chaque nouveau nœuds descendant « GOTerm » rencontré
7) Chaque ligne d'un nouveau fichier indique le GOTerm, sa fonction et les geneproducts associés (directement ou implicitement)

Exemple d'utilisation du programme :

>*python3 annotation.py*



On peut choisir 1 pour Direct, 2 pour Implicit ou directement écrire Direct ou Implicit

En output, on obtient un fichier *.sets* nommé par rapport au fichier entré en input (voir les fichiers dans le dossiers Donnees). Le script fonctionne aussi avec le .goa de *P. falciparum*.

Pour le fichier des annotations directs, on a obtenu 8848 termes GO annotés sur un total de 44 733 GO actuellement. Pour le fichier des annotations indirects, on a obtenu 12321 termes GO annotés sur 44 733. Lors des annotations, on avait 533 erreurs d'impossibilité à annoter sur la GO. On a donc une bonne annotation de la GO pour un organisme tel que la *melanogaster*

Ajout de fonctionnalités à blastset.py

Le second objectif était d'apporter des modifications sur le script blastset.py. Les modifications sont :

1) Ajout de la fonction de Coverage :

Le but est de rajouter une fonction non statistique (naïve) pour une autre mesure de dissemblance. Dans celle-ci, on fait un simple calcule de « coverage » en comparant les fractions des deux ensembles qui se correspondent

```
elif param.measure=='coverage': #Coverage
    if c == 0:
        pval = 1
    elif c==q and c==t:
        pval = 0
    else:
        pval = 1 - ((c / q) * (c / t))
```

Pour l'utiliser :

./blastsetmodifie –sets EcolA.biocyc.sets --query 'ALAS ARGS ASNS ASPS CYSS GLTX GLYQ GLYS HISS ILES' -m coverage

2) Ajout de la function chi2

On voulait aussi ajouter un chi2 d'indépendance via une table de contingence. Pour cela, on a ajouté la fonction chi2_contingency de la librairie scipy déjà utilisé par le programme. Pour l'utiliser :

./blastsetmodifie -sets EcolA.biocyc.sets --query 'ALAS ARGS ASNS ASPS CYSS GLTX GLYQ GLYS HISS ILES' -m chi2

3) Ajout d'une function random

Pour la suite du projet, j'ai ajouté une partie qui permet de produire des query aléatoires et de pouvoir faire cela un certain nombre de fois. Cela se passe au travers de trois arguments optionnels (qui ne sont pas à renseigner si on utilise une query prédéfini). Le premier argument est --random (-r) qui est le nombre de gène que l'on veut dans la query. Le second est –round(-R) qui est le nombre de fois que l'on veut faire les essais. Le troisième, c'est d'écrire random pour l'argument –query.

Exemple d'utilisation pour faire 5 essais de chi2 sur 10 gènes aléatoires sur le fichier .sets des annotations direct de melanogaster:

./blastsetmodifie –sets 17.D_melanogaster.goa_direct.sets –query random -m chi2 -r 10 -R 5

On obtiendra ainsi les pvalues des 5 essais pour chaque bonne correspondance d'annotation.

Comparaison des mesures ajoutées

Pour faire une comparaison des différentes mesures ajoutées, différentes méthodes me sont venues à l'idée. Tout d'abord, comparer les pvalues que l'on a entre chacune des méthodes en fonction des gènes que l'on recherche.

Dans cette optique, j'ai généré pour chaque mesure (coverage, chi2, hypergeometric et binomial) trois fichiers qui font des recherches aléatoires de liste de gènes petites (3), moyennes (7) et grandes (12) pour 50 fois. Le but étant ensuite de comparer les pvalue

récupérés. A première vue, la technique de coverage n'est pas efficace. Je n'ai obtenu aucune pvalue intéressante, certainement du au fait que l'on fait une simple comparaison des termes que l'on obtient : c'est une méthode qui ne fonctionne pas quand on ne sait pas ce que l'on recherche. Au contraire, la méthode du chi2 est plus efficace que celles déjà implémenté. Problème : elle est beaucoup plus longue à être exécuté. Je n'ai malheureusement pas fait les traitement sous R pour avoir des graphiques et chiffre qui le montre.

Dans les perspectives d'amélioration : on pourrait ajuster la FDR en fonction des requêtes que l'on émet par exemple, car une gêne aura plus de facilité à être intégrer dans des GOTerm qu'une liste de 10, il faudrait donc ajuster la spécificité.

Bilan personnel sur cette partie du projet

Le projet est intéressant dans le cadre du développement du script pour faire les annotations et aussi pour la modification du blastset. En effet, j'ai aimé replonger dans le code de l'année dernière et pouvoir appliquer autrement la Gene Ontology car c'est un concept que j'ai encore du mal à comprendre. Malheureusement, je n'ai pas eu le temps de plus me pencher sur les comparaisons des méthodes.