Master 1 Bioinformatique

Object-oriented programming

Hierarchical clustering

Olivier Dameron
olivier.dameron@univ-rennes1.fr

February 9, 2018

Version 1.4

1 Objective

This project's goal is to classify a set of students according to their grades, and to generate the corresponding dendrogram.

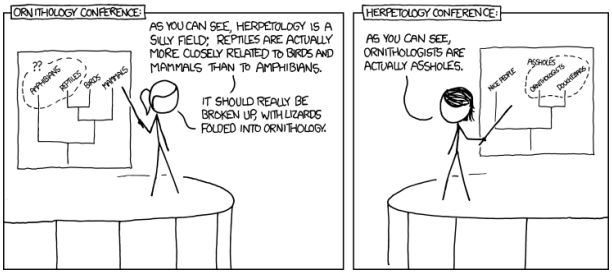

Figure 1: xkcd 867 (`http://xkcd.com/867/`)

2 Representing a class of students

2.1 Class `Student`: methods overload

Step 1 *Create a class* `Student` *that represents the set of students. Each student has an (assumed) unique identifier (a string) and a grade (a double).*

Step 2 *Add a first constructor having for parameters an identifier and a grade. Then add a second constructor having an identifier as single parameter. This is an excellent opportunity to use method overload...*

Step 3 *Add the methods* `getIdent()`, `getGrade()` *et* `setGrade(double newGrade)`.

Step 4 *In the* `main(...)` *method, create the following instances and check that the methods from step 3 still work correctly:*

```
Student riri = new Student("riri", 12.5);
Student fifi = new Student("fifi", 14.0);
Student loulou = new Student("loulou", 18.5);
Student geo = new Student("geo", 19.5);
Student donald = new Student("donald", 10.5);
```

2.2 Class `Promo`: inheritance and static methods

Step 5 *Create a class* `Promo` *that represents a set of students.* `Promo` *is a sub-class of* `java.util.ArrayList`[1]. *Make sure to read the documentation for* `ArrayList`*, you will need it soon. Please note that* `ArrayList` *is a generic class, whereas all the elements of a* `Promo` *instance are composed of instances of* `Student`*, so you will need to state that* `Promo` *is a subclass of* `ArrayList<Student>`.

Step 6 *In the* `Promo`*'s* `main(...)` *method, create an instance* `m1bioinfo` *of* `Promo`*, and add the members* `riri`, `fifi`, `geo`, `donald` *and* `loulou` *(respect this order so that the highest and lowest grades are in the middle of the list. The idea here is to avoid having the students almost sorted for the clustering).*

Step 7 *Add the methods* `getMinGrade()`, `getMaxGrade()` *et* `getAverageGrade()` `to the class Promo`. *The class* `java.lang.Math`[2] *has several useful methods. For iterating over all the students that compose a promotion, you can seek inspiration from the article "Traversing collections"*[3]. *Feel glad to have declared* `Promo` *as a subclass of* `ArrayList`.

Step 8 *In the* `main(...)` *method of* `Promo`*, add the code for printing the lowest grade, the highest and the average for* `m1bioinfo`.

Step 9 *The* `PromoLoader` *class (**FIXME**: fournie sur l'ENT) has a method* `loadTsvFile(...)` *that takes as argument a text file (one student per line; its identifier, a tabulation, its grade) for creating an instance of* `Promo`. *Why is the method* `loadTsvFile(...)` *declared as "*`static`*"? Draw the parallel with the methods* `min(...)`, `max(...)` *and* `abs(...)` *from* `java.lang.Math`.

3 Hierarchical clustering

3.1 Principle

3.1.1 Agglomerative vs. divisive approaches

Classifying consists in organizing a set of elements in groups based on the elements' similarities or differences.

[1] `http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html`
[2] `http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html`
[3] `http://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html`

Hierarchical clustering consists in organizing the sets of elements into subsets included in to each others in a tree-like structure. There are two main approaches for determining this organization:

- the **agglomerative approach** (also called ascending) starts by creating one (atomic) cluster for each element, and then iteratively generates new clusters composed of the most similar two, until there only remains one cluster;
- the **divisive approach** (also called descending) starts by gathering all the elements into a single cluster, and then iteratively decompose the clusters into subclusters until each of them is only composed of a single element.

The divisive approach requires more operations than the agglomerative one and is therefore longer... except when we only need the most general clusters (e.g. to separate a sample into two groups).

3.1.2 Distance measures between elements and between clusters

For both the agglomerative and ther divisive approaches, clustering depends on two main parameters:

- a **distance measure between elements** (also simply called *distance*). There are several classical ones: euclidian distance, Manhattan distance... In our case, we will consider that the distance between two students is the absolute value of the difference of their grades;
- a **distance measure between clusters** (also called *linkage*) that relies on the *distance* between elements of the two clusters. There are several classical linkage measures: the average of the distances between all the combinations of elements, their maximum, their minimum... In our case, we will consider that the distance between two clusters of students is the average of the distances between all the elements of the first cluster and all the elements of the second cluster.

3.2 Classe `ClusterOfStudents`

This section aims at implementing the `ClusterOfStudents` class for representing a cluster of `Student` instances. A simple cluster is composed of a single instance of `Student`. A complex cluster is composed of several sub-clusters which can themselves be either simple or complex clusters. A complex cluster has a tree-like structure where all the leaves are simple clusters.

Initially, a complex cluster is only composed of simple clusters (Fig. 2). After clustering, a complex cluster is composed of sub-clusters that are intermediate complex clusters (Fig. 3)

3.2.1 Initialisation

Step 10 *Créez la classe* `ClusterOfStudents` *avec un attribut* `subclusters` *qui représente la liste des sous-clusters. Pour simplifier l'étape de clustering où il est nécessaire de parcourir la liste des étudiants composant un cluster, on ajoute l'attribut* `students` *qui représente la liste des étudiants constituant les feuilles du cluster.*

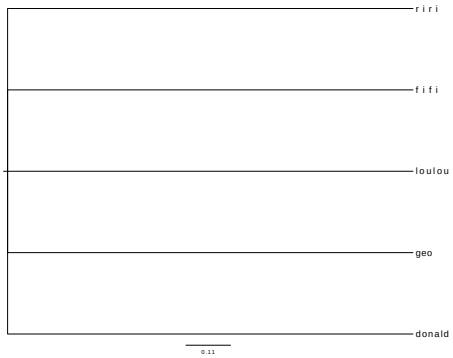

Figure 2: Complex cluster in its initial state: it is composed of five simple sub-clusters, each composed of a student.

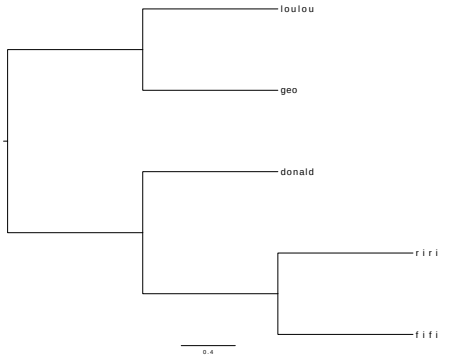

Figure 3: Complex cluster after clustering. It is composed of two intermediate complex sub-clusters. The firt is itself composed of two simple clusters (geo and loulou). The second is composed of a smple sub-cluster (donald) and a complex sub-cluster composed of two simple sub-clusters (riri and fifi).

Step 11 *Ajoutez un constructeur par défaut* `ClusterOfStudents()` *qui crée un cluster vide (on ne devrait pas en avoir besoin, mais c'est plus propre), un constructeur* `ClusterOfStudents(Student aStudent)` *qui crée un cluster simple, et un constructeur* `ClusterOfStudents(Promo aPromo)` *qui crée un cluster complexe dans l'état initial.*

Step 12 *Dans le* `main(...)` *de* `ClusterOfStudents`*, créez une instance de cluster simple* `geoCluster` *initialisée avec* `geo`*, et une instance de cluster complexe* `bigCluster` *initialisée avec* `m1big`*.*

3.2.2 Visualisation

Le format Newick[4] permet une représentation simple des arbres et des dendrogrammes, qui est compris par de nombreux outils de visualisation. Vous

[4] `http://evolution.genetics.washington.edu/phylip/newicktree.html`

pourrez par exemple utiliser le serveur web Tree Viewer[5] ou T-REX[6] ou les programmes FigTree[7] dendroscope[8] (utilisation académique libre, il n'est pas nécessaire d'obtenir une licence pour les fonctions de base). FigTree est celui qui semble donner les meilleurs résultats.

Le dendrogramme de la figure 2 peut être représenté par
`(riri,fifi,loulou,geo,donald);`
et celui de la figure 3 par
`((loulou,geo),(donald,(riri,fifi)));`.

NB : pour la visualisation, on aurait aussi pu utiliser les fonctions de R via le binding Java–R, mais c'est plus compliqué, et puis l'exercice de générer des fichiers au format Newick était intéressant.

Step 13 *Ajoutez une méthode* `getNewick()` *qui renvoie une chaîne de caractères représentant le dendrogramme au format Newick. À cause du point-virgule final, vous aurez peut-être besoin d'introduire une fonction intermédiaire* `getNewickIntermediate()`*. Puisqu'il s'agit d'un parcours d'arbre, vous vous simplifierez la vie en adoptant une approche récursive. Quelle devra être la visibilité de ces méthodes (publique, protégée, privée) ?*

Step 14 *Générez la représentation de* `bigCluster` *au format Newick et vérifiez (par exemple avec T-REX ou dendroscope) qui vous obtenez quelque chose de similaire à la figure 2.*

3.2.3 Clustering

Step 15 *Ajoutez une méthode* `linkage(ClusterOfStudents anotherCluster)` *qui renvoie la distance entre deux clusters. Le plus simple est sans doute de choisir la moyenne des différences des notes entre toutes les combinaisons d'étudiants du cluster courant et de* `anotherCluster`*.*

Step 16 *Dans le* `main(...)` *de* `ClusterOfStudents`*, créez les clusters simples* `loulouCluster` *et* `donaldCluster` *et vérifiez que les distances entre* `geoCluster`*,* `loulouCluster` *et* `donaldCluster` *correspondent à ce que vous attendez (pensez à vérifier les six combinaisons).*

Step 17 *Dans le* `main(...)` *de* `ClusterOfStudents`*, créez le cluster complexe* `geoLoulouCluster` *et vérifiez que sa distance avec* `donaldCluster` *correspond à ce que vous attendez. Vérifiez également la distance entre* `donaldCluster` *et* `geoLoulouCluster`*.*

Step 18 *Ajoutez la méthode* `clusterizeAgglomerative()`*. Clusterisez* `bigCluster` *et affichez le résultat au format Newick.*

La figure 4 montre le résultat de la classification pour un ensemble d'étudiants. Observez que puisque toutes les branches ont une longueur constante, on a l'impression de distinguer deux clusters principaux. La figure 5 montre qu'en rendant la longueur des branches proportionnelle à la distance séparant les deux clusters que l'on fusionne, ce sont en fait trois clusters principaux qui apparaissent (voir section 4.1).

[5] `http://www.proweb.org/treeviewer/`
[6] `http://www.trex.uqam.ca/`
[7] `http://tree.bio.ed.ac.uk/software/figtree/`
[8] `http://ab.inf.uni-tuebingen.de/software/dendroscope/`

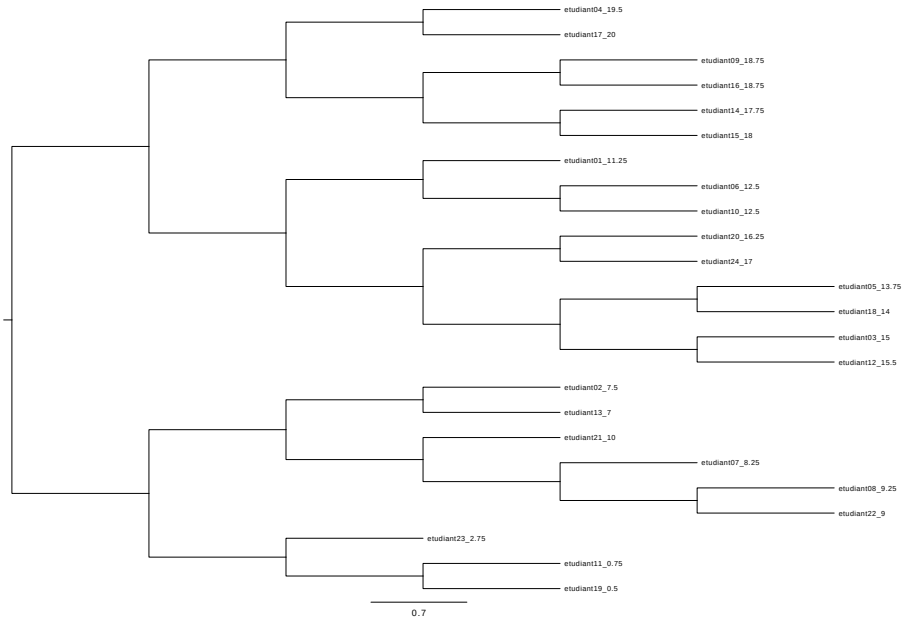

Figure 4: Cluster complexe après clustering.

4 Extensions facultatives

4.1 Amélioration du dendrogramme

Le format Newick permet d'indiquer la longueur de chacun des branches. Utilisez plutôt l'application FigTree[9] ou le site de Tree Viewer[10] pour la visualisation, les autres outils semblent avoir des problèmes.

Step 19 *Améliorez la méthode* `getNewick()` *pour que toutes les feuilles soient au même niveau.*

Step 20 *Améliorez la méthode* `getNewick()` *pour que toutes les feuilles soient au même niveau et que la longueur des branche soit proportionnelle à l'écart des notes.*

4.2 Approche divisive

Step 21 *Ajoutez une méthode* `clusterizeDivisive()` *à la classe* `ClusterOfStudents`*. Clusterisez* `bigCluster` *et comparez avec l'approche ascendante.*

4.3 Modélisation de `ClusterOfStudents`

Dans la classe `ClusterOfStudents`, chaque instance d'étudiant apparaît deux fois :

[9] `http://tree.bio.ed.ac.uk/software/figtree/`
[10] `http://www.proweb.org/treeviewer/`

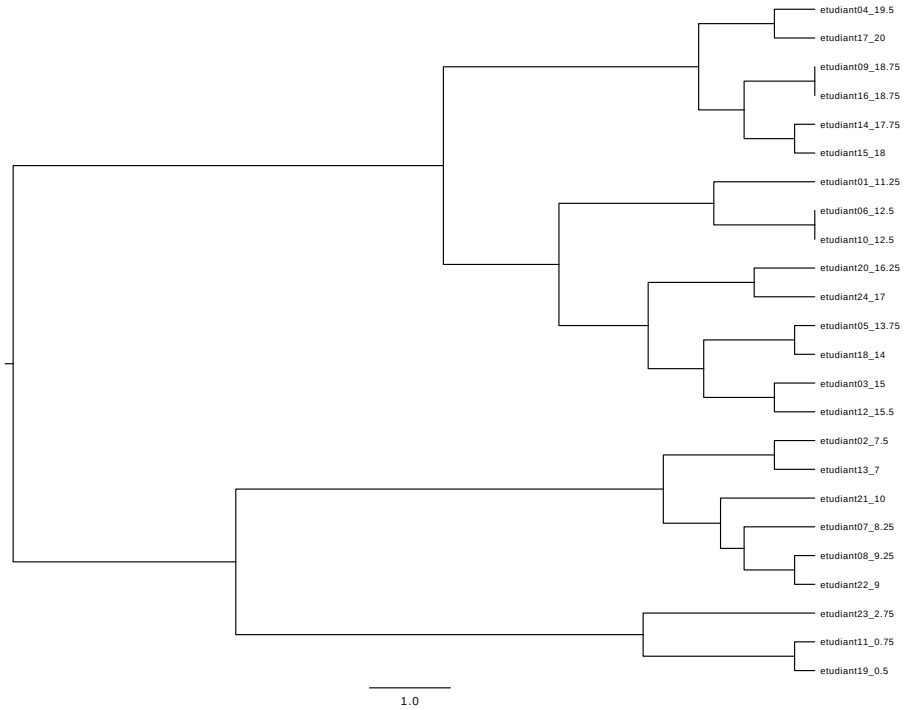

Figure 5: Cluster complexe après clustering. La longueur des branches est proportionnelle à la distance séparant les deux clusters fusionnés.

- dans l'attribut `subClusters` puisque le dendrogramme a autant de sous-clusters feuilles que d'étudiants ;
- dans l'attribut `students` qui permet un parcours plus simple de la liste des étudiants d'un cluster en évitant de devoir parcourir récursivement tout le dendrogramme à chaque fois.

On pourrait penser que cela occupe donc deux fois plus de place en mémoire que nécessaire, même si dans notre cas ce surcoût est acceptable dans la mesure où chaque instance occupe peu de place mémoire et qu'il y a relativement peu d'étudiants. Néanmoins, Java ne duplique évidemment pas les instances de `Student` dans les deux attributs. Chaque attribut ne contient en fait que les adresses des instances de `Student` (on appelle ça un *passage d'objet par référence*). Ainsi, si vous modifiez la note d'un étudiant dans `students`, cette modification apparaîtra également dans `subClusters`, et inversement.

Au final, le fait d'utiliser deux attributs qui semblent redondants parce qu'ils contiennent les mêmes objets :

- a l'avantage principal d'améliorer le temps de traitement en évitant un parcours de l'arborescence du dendrogramme chaque fois que l'on souhaite parcourir les étudiants (et cela arrive souvent durant l'étape de clustering) ;
- a l'avantage secondaire de simplifier l'écriture de la classe en vous dispensant justement d'écrire la fonction de parcours de l'arborescence du

dendrogramme ;

- a l'inconvénient d'augmenter légèrement la consommation de la mémoire.

Step 22 *Écrivez une classe* `ClusterOfStudentsBis` *qui ne contient que l'attribut* `subClusters`*. Comparez les temps de clusterisation de* `ClusterOfStudents` *et de* `ClusterOfStudentsBis`*.*