Master 1 Bioinformatique

Object-oriented programming

Hierarchical clustering

Emmanuelle Becker, Olivier Dameron, Marine Louarn

February 13, 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.

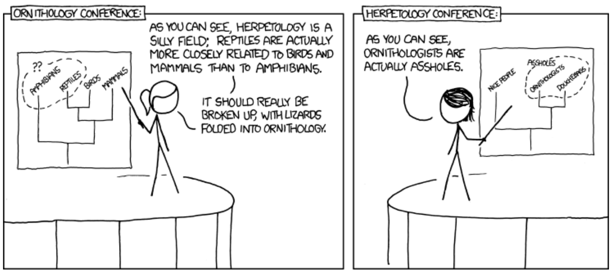

Figure 1: xkcd 867 (`http://xkcd.com/867/`)

2 Set up your environment

2.1 Cloning the project

The project's description, the Java source files and some example datasets can be retrieved from its git repository[1].

FIXME: We should definitely do a fork instead

Step 1 *Set your working environment up:*

1. *create an empty local directory;*

[1] `https://gitlab.com/odameron/javaHierarchicalClustering`

2. *move to this directory;*
3. *clone te project with* `git clone <gitURL>` *(you have to retrieve* `gitURL` *from the web page);*
4. *create you own branch with* `git branch -b devLastnameFirstname` *(obviously, adapt Lastname and Firstname). Reminder: you might want to read the section on branches from a git tutorial*[2]

From now on, you are strongly encouraged to use git profusely and commit at least at each step.

2.2 Configure your editor

FIXME: instructions for importing in Eclipse or writing an ant file

FIXME: instructions for runing javadoc in the `doc` directory

3 Representing a class of students

3.1 Class `Student`: methods overload

Step 2 *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 3 *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 4 *Add the methods* `getIdent()`, `getGrade()` *et* `setGrade(double newGrade)`*.*

Step 5 *In the* `main(...)` *method, create the following instances and check that the methods from step 4 still work correctly:*

```java
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);
```

Step 6 ***FIXME****: run javadoc and commit*

3.2 Class `StudentGroup`: inheritance and static methods

Step 7 *Create a class* `StudentGroup` *that represents a set of students.* `StudentGroup` *is a sub-class of* `java.util.ArrayList`[3]*. 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* `StudentGroup` *instance are composed of instances of* `Student`*, so you will need to state that* `StudentGroup` *is a subclass of* `ArrayList<Student>`*.*

[2] https://bioinfo-fr.net/git-usage-collaboratif

[3] http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

Step 8 *In the `StudentGroup`'s `main(...)` method, create an instance `m1bioinfo` of `StudentGroup`, 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 9 *Add the methods `getMinGrade()`, `getMaxGrade()` et `getAverageGrade()` to the class `StudentGroup`. The class `java.lang.Math`[4] has several useful methods. For iterating over all the students that compose a promotion, you can seek inspiration from the article "Traversing collections"[5]. Feel glad to have declared `StudentGroup` as a subclass of `ArrayList`.*

Step 10 *In the `main(...)` method of `StudentGroup`, add the code for printing the lowest grade, the highest and the average for `m1bioinfo`.*

Step 11 *The `StudentGroupLoader` 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 `StudentGroup`. Why is the method `loadTsvFile(...)` declared as "`static`"? Draw the parallel with the methods `min(...)`, `max(...)` and `abs(...)` from `java.lang.Math`.*

Step 12 ***FIXME**: run javadoc and commit*

4 Hierarchical clustering

4.1 Principle

4.1.1 Agglomerative vs. divisive approaches

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

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).

[4] `http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html`

[5] `http://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html`

4.1.2 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.

4.2 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)

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

4.2.1 Initialization

Step 13 *Create a `ClusterOfStudents` with a `subclusters` attribute that represents the list of its sub-clusters. For simplifying the clustering step (when marshalling the students composing the cluster), add an attribute `students` that represents the set of students constituting the leaves of the cluster.*

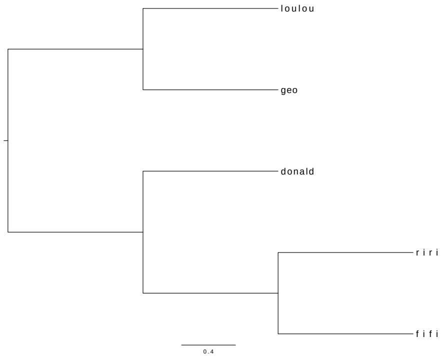

Figure 3: Complex cluster after clustering. It is composed of two intermediate complex sub-clusters. The firt is itself composed of two simple clusters (geo and loulou). The second is composed of a smple sub-cluster (donald) and a complex sub-cluster composed of two simple sub-clusters (riri and fifi).

Step 14 *Add the following constructors:*

- *a default constructor* `ClusterOfStudents()` *that creates an empty cluster (we do not reaaly need it, but I find it cleaner to have a default constructor);*
- *a constructor for simple clusters* `ClusterOfStudents(Student aStudent);`
- *a constructor for complex clusters before clustering* `ClusterOfStudents(StudentGroup aStudentGroup)`*.*

Step 15 *In* `ClusterOfStudents`*'* `main(...)` *main method, create an instance of a simple cluster* `geoCluster` *initialized with* `geo`*, and an instance of complex cluster* `bioCluster` *initialized with* `m1bioinfo`*.*

Step 16 ***FIXME****: run javadoc and commit*

4.2.2 Visualization

The Newick format[6] provides a straightforward representation of trees and dendrograms, and is supported by most visualization tools. You can use the Tree Viewer web server[7] or T-REX[8] or the dedicated softwares FigTree[9], dendroscope[10] (free use in an academic context; getting a licence is not required for the basic functions). FigTree seems to give the best results.

The dendrogram from Fig. 2 can be represented by `((loulou,geo),(donald,(riri,fifi)));`.

NB : for visualizing dendrograms, we could as well have used the R functions via the Java–R binding, but it is more complicated, and writting Newick files makes for an interesting exercice anyway.

[6] `http://evolution.genetics.washington.edu/phylip/newicktree.html`
[7] `http://www.proweb.org/treeviewer/`
[8] `http://www.trex.uqam.ca/`
[9] `http://tree.bio.ed.ac.uk/software/figtree/`
[10] `http://ab.inf.uni-tuebingen.de/software/dendroscope/`

Step 17 *Add a `getNewick()` method to the class `ClusterOfStudents` that returns a string representing the dendrogram in the Newick format. Because of the final semicolon, you may need to introduce an intermediate function (aptly named `getNewickIntermediate()`). For marshalling the tree, you will make your life easier by considering a recursive approach (but this is not mandatory). Should these methods' visibility be public, protected or private?*

Step 18 *Generate a Newick representation of `bioCluster` and check (for example with T-REX or dendroscope) whether you get something similar to Fig. 2.*

Step 19 ***FIXME**: run javadoc and commit*

4.2.3 Clustering

Step 20 *Add a method `linkage(ClusterOfStudents anotherCluster)` that returns the distance between the current cluster and `anotherCluster`. Choosing the average of the absolute value of the grade differences for each combination of students from each cluster is probably the easiest solution.*

Step 21 *In `ClusterOfStudents`'s `main(...)` method, create two simple clusters `loulouCluster` and `donaldCluster` and check whether the distance between `geoCluster`, `loulouCluster` and `donaldCluster` are what you expect them to be (check the six combinations).*

Step 22 *In `ClusterOfStudents`'s `main(...)` method, create the complex cluster `geoLoulouCluster` and check whether its distance with `donaldCluster` and `geoLoulouCluster` (and conversely).*

Step 23 *Add a method `clusterizeAgglomerative()`. Perform clustering on `bigCluster` and display the result as a Newick string.*

Step 24 ***FIXME**: run javadoc and commit*

Figure 4 shows the classification result for a set of students. Notice that because all the branches have the same length, the dendrogram seems to display two main clusters. Figure 5 shows that by making the length of each branch proportional to the distance separating the two clusters it joins, the dendrogram reveals three main clusters (cf. section 5.1).

5 Optional extensions

5.1 Dendrogram improvement

Le format Newick permet d'indiquer la longueur de chacun des branches. Utilisez plutôt l'application FigTree[11] ou le site de Tree Viewer[12] pour la visualisation, les autres outils semblent avoir des problèmes.

Step 25 *Améliorez la méthode `getNewick()` pour que toutes les feuilles soient au même niveau.*

Step 26 *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.*

[11] `http://tree.bio.ed.ac.uk/software/figtree/`

[12] `http://www.proweb.org/treeviewer/`

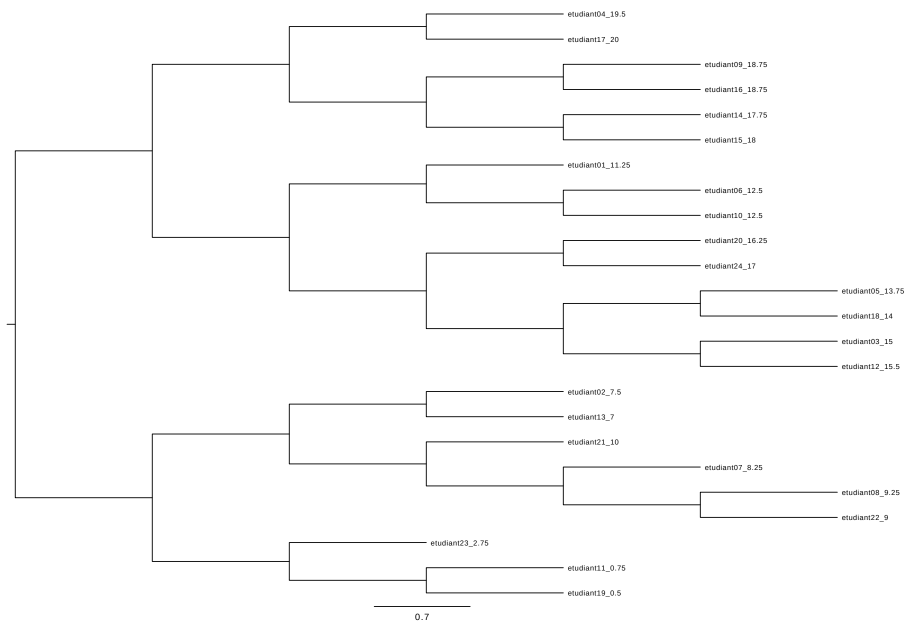

Figure 4: Complex cluster after clustering. The length of each branch is constant.

5.2 Approche divisive

Step 27 *Ajoutez une méthode `clusterizeDivisive()` à la classe `ClusterOfStudents`. Clusterisez `bigCluster` et comparez avec l'approche ascendante.*

5.3 Modélisation de `ClusterOfStudents`

Step 28 *Should we have declared `ClusterOfStudents` as a subclass of `StudentGroup`? Are theses classes' internal structures compatible? Are there `StudentGroup`'s attribute or methods for which such an inheritance would make sense?*

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

- 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*

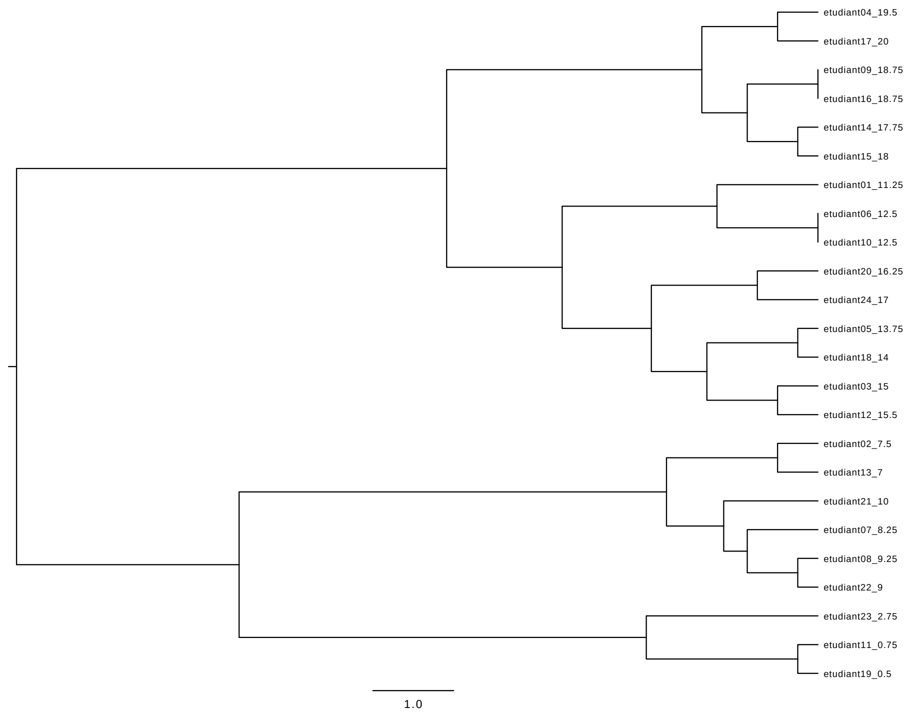

Figure 5: Complex cluster after clustering. The length of each branch is proportional to the distance between the two clusters it unites.

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