Compared analysis of taxonomic trees

C. REDA

supervised by M. NIKOLSKI and M. RAFFINOT
CBIB, Bordeaux

August, 21 2016

Abstract

Taxonomic trees are trees that represent the classification of species into groups according to their evolutionnary past. In metagenomics, that is the study of genetic material from samples taken from a natural environment (opposite to labs), taxonomic trees allow to identify the species matching the samples, and thus to determine the composition of the microbial population endemic to the environment.

Interest in metagenomics has recently plummeted in bioinformatics, as a consequence of the decreasing price of DNA sequencing, and also for studying environmental samples has unveiled the existence of many bacterias which could not be cultivated in labs. Metagenomics has many applications in biology and medicine. The subject of this internship is to improve the analysis of metagenomic data in medical applications.

Contents

List of Figures

List of Tables

Chapter 1

Scientific context

First we explain what metagenomics is, how taxonomic trees intervene in this scientific field, and what are the most burning issues today in metagenomics. Then we introduce the test database.

1.1 Definition and goals of metagenomics

Metagenomics is the study of the genetic material of samples, often directly taken from a natural environment. It can be parts of DNA, proteins or RNA.

We only focus here on bacteria's DNAs. Indeed, bacteria used to be cultivated in laboratories before having their DNA sequenced and annoted. However a great number of bacteria cannot be raised in an artificial environment, such as some bacteria in animal guts or living in fragile marine ecosystems. Thus metagenomics helps to study more and unknown species of bacteria. Along with the decreasing price of DNA sequencing, this explains the development and the growing interest in metagenomics today.

The whole pipeline to turn raw genetic material into reliable data comprises these following steps:

- After removing samples of genetic data from the environment, one extracts the DNA from the sample by a chemical reaction.
- Then one sequences the resulting DNA, that is obtains strings of characters that correspond to the primary structure (one-dimensioned) of the DNA, i.e. the order of (set of) the base pairs constructed with cytosine ($\{C\}$), adenine ($\{A\}$), guanine ($\{G\}$), and thymine ($\{T\}$). Note that there exist other set of bases such as $N = \{A, C, T, G\}$ or X, and this can increase greatly the complexity of sequencing. There are two main ways to sequence DNA: the first method, called *Sanger sequencing*, is the slowest, but gives

good quality sequences. The second one NGS (that stands for Next-Generation Sequencing [31]) is faster, however resulting sequences can contain errors. NGS produces only parts of the DNA, of length 32 to 1,000 base pairs. As a general rule, these two methods are mixed to make a good tradeoff between quality and price of the sequencing.

To distinguish one bacteria from another, sequencing is performed on genes, called *16S*, that are present in all bacteria. These genes contain highly conserved regions (they are the same in all bacteria) and highly variable regions *HVR*. The sequencing therefore focuses on the *HVR* to determine to which bacteria a given sequence belongs. It is particularly hard to correctly determine the bounds of these *HVR*, and it is one of the causes of errors in sequencing.

After sequencing by NGS, one obtains *reads* -bits of DNA sequences of length 32 to 1,000 base pairs: e.g. this (partial) read is associated with the *16S* gene of species *Hydrothermal vent clone VC2.1 Arc13*:

```
AGGCCACTGCTATCGGGCTCCGACTAAGCCATGCGAGTCTAG
GGGCTCCTTCGGGAGCACCGGCGGACGGCTCAGTAACACGTGGACAA
CCTACCCTCGGGTGGGGGATAACCTCGGGAAACTGAGGCTAATACCC...
```

Once the sequencing is done, reads are obtained, and then they are usually matched with the reference sequences found in databases such as NCBI, GreenGenes or RDP: for each read, one looks for the best alignment with one of the reference sequences, that is the alignment that preserves most the order of the bases (see figure 1.1). After this step, in the case of human bacteria (unfortunately it is not a general rule), to one read corresponds one or more sequences, that is, one or more species.

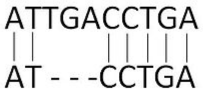


Figure 1.1: An example of alignment of DNA sequence, from *Gap penalty* article in Wikipedia. The upper read is the reference to which the lower read is aligned

The main objective of metagenomics is to determine to which species belong the sequencing reads by matching to the reference sequences.

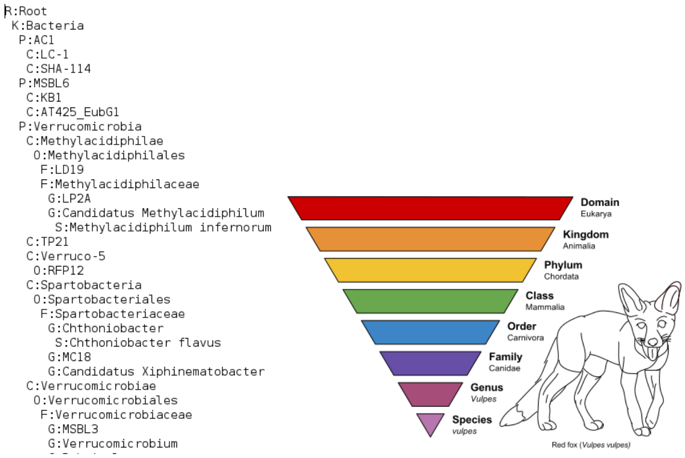
1.2 Assignment/identification of reads to a certain species of bacteria

There exists no unique common phylogeny. A handful of taxonomic trees are used in bioinformatics, coming from different databases such as, for instance, NCBI, GreenGenes or RDP.

A taxonomic tree (see figure 1.2) is of bounded height, which value can vary according to the database. In GreenGenes's taxonomic tree, that is the one used for our tests, there are eight ranks, that is, eight levels/generations in the tree in growing precision (see figure 1.2):

- 'D' (Domain) or 'R' (Reign): this rank is basically divided into three nodes: '*Bacteria*', '*Archae*' and '*Eucarya*'.
- 'P' (Phylum): '*Animalia*', '*Plantae*', '*Fungi*', ...
- 'C' (Class): '*Mammalia*', ...
- 'O' (Order): '*Carnivora*', ...
- 'F' (Family): '*Carnidae*', ...
- 'G' (Genus): '*Vulpes*', ...
- 'S' (Species): '*Vulpes*', ... (corresponds to a single species)

Note that it exists also another group called OTU, which stands for Operational Taxonomic Unit. An OTU is a set of leaves and internal nodes of the taxonomic tree that are thought to be really close. The OTU are disjoint.



```
R:Root
 K:Bacteria
  P:AC1
   C:LC-1
   C:SHA-114
  P:MSBL6
   C:KB1
   C:AT425_EubG1
  P:Verrucomicrobia
   C:Methylacidiphilae
    O:Methylacidiphilales
     F:LD19
     F:Methylacidiphilaceae
      G:LP2A
      G:Candidatus Methylacidiphilum
       S:Methylacidiphilum infernorum
   C:TP21
   C:Verruco-5
    O:RFP12
   C:Spartobacteria
    O:Spartobacteriales
     F:Spartobacteriaceae
      G:Chthoniobacter
       S:Chthoniobacter flavus
      G:MC18
      G:Candidatus Xiphinematobacter
   C:Verrucomicrobiae
    O:Verrucomicrobiales
     F:Verrucomicrobiaceae
      G:MSBL3
      G:Verrucomicrobium
```

Figure 1.2: A partial taxonomic tree from GreenGenes and the taxonomy of the fox, from *Taxonomic Rank* article in Wikipedia

It is worth noticing that the nearest nodes to the root have the lowest degree, and that a taxonomic tree has a great proportion of leaves (S-ranked nodes) compared to the number of internal nodes. GreenGenes's taxonomic trees for the domains '*Bacteria*' and '*Archae*' used in TaxoTree (see in the next chapters) approximately owns $9,065$ nodes, and among them $5,565$ leaves.

Most of the time, in spite of the fact that one read can match many species -that is leaves of the taxonomic tree, a given read is assigned to only one node of the taxonomic tree. Generally speaking, a read is assigned to the *Least Common Ancestor* (LCA) [9] of the set of matched sequences/species/S-ranked nodes. The LCA of a set of leaves L is the last common node in the paths from the root to each leaf of L (see figure 1.3).

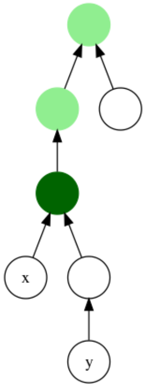


Figure 1.3: The dark green node is the LCA of x and y, from *Lowest Common Ancestor* article in Wikipedia

One can seldomly associate one read to a single species, because of errors in the sequencing. The deeper in the tree the read is assigned, better is the precision of the assignment. Nevertheless, some algorithms to improve the accuracy of assignment of reads have been designed [4] [2]. Please note eventually that a node can match several reads from a same sample, and thus several reads can be assigned to this node in a single sample.

1.3 Topics in metagenomics

There are already algorithms that enhance the assignment of a read to a node of the taxonomic tree [4], and measures on phylogenies that quantify the relevancy of the tree according to a certain distance matrix between reads, or according to a certain other taxonomic tree [21] [10]. But there is a lack of efficient tools [7] when it comes to compare several trees/forests by quantifiying their similarities and also their differences, and above all to draw biological conclusions on the presence or the absence of certain nodes, provided some information that cannot be given by the taxonomic tree. This sort of information is called in this report *metadata*, that is, information that cast light on the natural environment in which samples were taken.

For instance, given a group of patients afflicted with cystic fibrosis, one can extract samples from their guts and may want to evaluate the influence of the age or the treatment over the population of *E. Coli* bacteria in their patient's guts. Statistical methods, using e.g. WILCOXON [32] [13] or MC NEMAR's [15] tests, are nowadays used to solve these questions, but their interpretation needs a manual overlap of the analysed data. To guide the diagnosis or the evaluation of the efficiency of a treatment, practitionners are in need of a semi-automatic processing of the data.

1.4 The test database

The test database used to design our algorithms is a study which took place at the Children's Hospital of Bordeaux, from November, 2015 to May, 2016 [7]. All patients were afflicted with cystic fibrosis, and some of them needed a heavy antibacterial treatment by intraveinous injection (denoted ATB-IV) to cure their chronic bacterial infections. The aim of the study was to underline the influence of antibacterials on the microbial population of the patient's gut.

Patients were divided into witnesses (W) and treated patients (T). Those in the W group saw the paediatrician twice, on Day 0 and Day 90, whereas those in the T group saw him thrice, on Day 0, Day 45 and Day 90, and were treated from Day 0 to Day 45. There are in total 47 samples, with 21 patients.

Each time patients saw the doctor, they gave them samples of their stool and a filled survey over their quality of life. Samples were then sequenced. Later on, results were filtered and normalized this way:

- Variables that will not likely be useful (e.g. a node with a number of assignments too small) are identified and removed: for instance, if the number of assignments falls below the first quartile, the variable associated is discarded.
- Assignment numbers have been normalized relatively to the total number of reads assigned in the taxonomic tree.
- Then these numbers have been mean-centered and reduced.

Chapter 2

Subject of the internship

Our research aimed at solving two quite similar problems, that mainly target at *clustering* patients, that is separating them into groups which maximize the resemblance between two patients of a same group or *cluster*, and minimize it for a pair of patients from different *clusters*. For all these problems, a common numbering of samples, nodes in the taxonomic tree, and non-taxonomic information (*metadata*) have been chosen. Furthermore, we use for input a reference taxonomic tree denoted T (from GreenGenes's in our test database), and a *data/information matrix* denoted INFOM (see Figure 2.2), such as INFOM[i][j] is the value of metadatum j in sample i (being 0 or 1 for boolean values).

2.1 Problem of most different pairs (MDPairs)

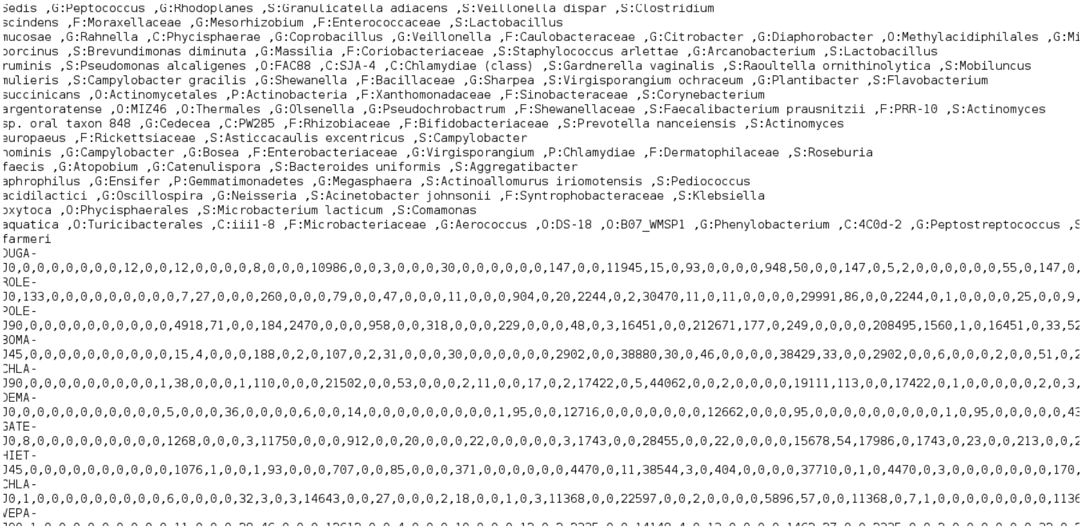
MDPAIRS is:

SUPPLEMENTARY INPUT:

- A matrix OCCM (called *occurrence matrix*) such as OCCM[i][j] (see Figure 2.1) is the number of assignments to node j in sample i.
- A set of groups of samples S.

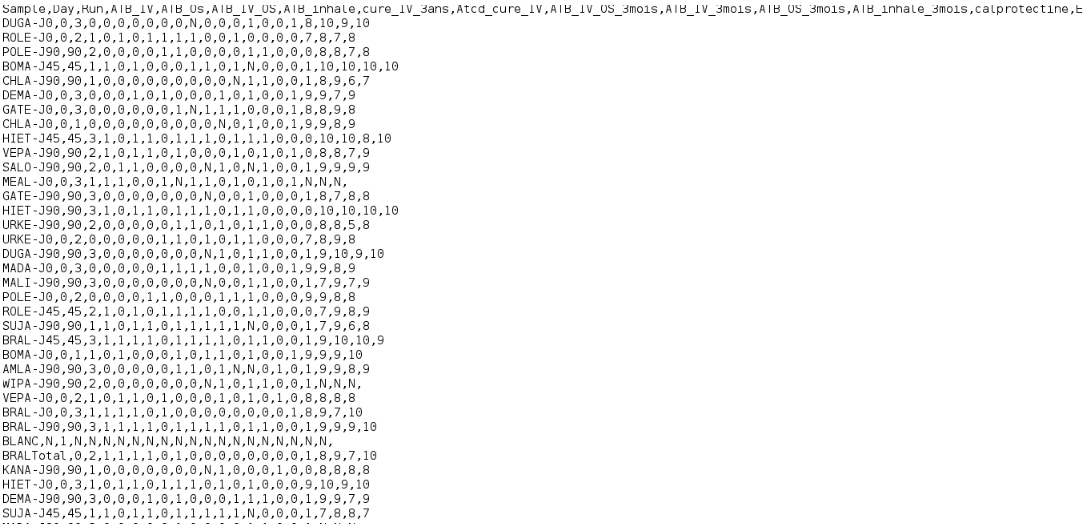
OUTPUT:

- The list of the pairs in S that are the most different.



```
Sedis ,G:Peptococcus ,G:Rhodoplanes ,S:Granulicatella adiacens ,S:Veillonella dispar ,S:Clostridium
scindens ,F:Moraxellaceae ,G:Mesorhizobium ,F:Enterococcaceae ,S:Lactobacillus
mucosae ,G:Rahnella ,C:Phycisphaerae ,G:Coprobacillus ,G:Veillonella ,F:Caulobacteraceae ,G:Citrobacter ,G:Diaphorobacter ,O:Methylacidiphilales ,G:Mi
porcinus ,S:Brevundimonas diminuta ,G:Massilia ,F:Coriobacteriaceae ,S:Staphylococcus arlettae ,G:Arcanobacterium ,S:Lactobacillus
ruminis ,S:Pseudomonas alcaligenes ,O:FAC88 ,C:SJA-4 ,C:Chlamydiae (class) ,S:Gardnerella vaginalis ,S:Raoultella ornithinolytica ,S:Mobiluncus
mulieris ,S:Campylobacter gracilis ,G:Shewanella ,F:Bacillaceae ,G:Sharpea ,S:Virgisporangium ochraceum ,G:Plantibacter ,S:Flavobacterium
succinicans ,O:Actinomycetales ,P:Actinobacteria ,F:Xanthomonadaceae ,F:Sinobacteraceae ,S:Corynebacterium
argentoratense ,O:MIZ46 ,O:Thermales ,G:Olsenella ,G:Pseudochrobactrum ,F:Shewanellaceae ,S:Faecalibacterium prausnitzii ,F:PRR-10 ,S:Actinomyces
sp. oral taxon 848 ,G:Cedecea ,C:PW285 ,F:Rhizobiaceae ,F:Bifidobacteriaceae ,S:Prevotella nanceiensis ,S:Actinomyces
europaeus ,F:Rickettsiaceae ,S:Asticcacaulis excentricus ,S:Campylobacter
hominis ,G:Campylobacter ,G:Bosea ,F:Enterobacteriaceae ,G:Virgisporangium ,P:Chlamydiae ,F:Dermatophilaceae ,S:Roseburia
faecis ,G:Atopobium ,G:Catenulispora ,S:Bacteroides uniformis ,S:Aggregatibacter
aphrophilus ,G:Ensifer ,P:Gemmatimonadetes ,G:Megasphaera ,S:Actinoallomurus iriomotensis ,S:Pediococcus
acidilactici ,G:Oscillospira ,G:Neisseria ,S:Acinetobacter johnsonii ,F:Syntrophobacteraceae ,S:Klebsiella
oxytoca ,O:Phycisphaerales ,S:Microbacterium lacticum ,S:Comamonas
aquatica ,O:Turicibacterales ,C:iii1-8 ,F:Microbacteriaceae ,G:Aerococcus ,O:DS-18 ,O:B07_WMSP1 ,G:Phenylobacterium ,C:4C0d-2 ,G:Peptostreptococcus ,S
farmeri
DUGA-
J0,0,0,0,0,0,0,0,0,12,0,0,12,0,0,0,0,8,0,0,0,10986,0,0,3,0,0,0,30,0,0,0,0,0,0,147,0,0,11945,15,0,93,0,0,0,0,948,50,0,0,147,0,5,2,0,0,0,0,0,0,55,0,147,0,
ROLE-
J0,133,0,0,0,0,0,0,0,0,0,7,27,0,0,0,260,0,0,0,79,0,0,47,0,0,0,11,0,0,0,904,0,20,2244,0,2,30470,11,0,11,0,0,0,0,29991,86,0,0,2244,0,1,0,0,0,0,25,0,0,9,
POLE-
J90,0,0,0,0,0,0,0,0,0,0,4918,71,0,0,184,2470,0,0,0,958,0,0,318,0,0,0,229,0,0,0,48,0,3,16451,0,0,212671,177,0,249,0,0,0,0,208495,1560,1,0,16451,0,33,52
BOMA-
J45,0,0,0,0,0,0,0,0,0,0,15,4,0,0,0,188,0,2,0,107,0,2,31,0,0,0,30,0,0,0,0,0,0,2902,0,0,38880,30,0,46,0,0,0,0,38429,33,0,0,2902,0,0,6,0,0,0,2,0,0,51,0,2
CHLA-
J90,0,0,0,0,0,0,0,0,0,1,38,0,0,0,1,110,0,0,0,21502,0,0,53,0,0,0,2,11,0,0,17,0,2,17422,0,5,44062,0,0,2,0,0,0,0,19111,113,0,0,17422,0,1,0,0,0,0,0,2,0,3,
DEMA-
J0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,36,0,0,0,0,6,0,0,14,0,0,0,0,0,0,0,0,0,1,95,0,0,12716,0,0,0,0,0,0,0,12662,0,0,0,95,0,0,0,0,0,0,0,0,0,1,0,95,0,0,0,0,0,43
GATE-
J0,8,0,0,0,0,0,0,0,0,0,1268,0,0,0,3,11750,0,0,0,912,0,0,20,0,0,0,22,0,0,0,0,0,3,1743,0,0,28455,0,0,22,0,0,0,0,15678,54,17986,0,1743,0,23,0,0,213,0,0,2
HIET-
J45,0,0,0,0,0,0,0,0,0,0,1076,1,0,0,1,93,0,0,0,707,0,0,85,0,0,0,371,0,0,0,0,0,0,4470,0,11,38544,3,0,404,0,0,0,0,37710,0,1,0,4470,0,3,0,0,0,0,0,0,0,0,170,
CHLA-
J0,1,0,0,0,0,0,0,0,0,0,6,0,0,0,0,32,3,0,3,14643,0,0,27,0,0,0,2,18,0,0,1,0,3,11368,0,0,22597,0,0,2,0,0,0,0,5896,57,0,0,11368,0,7,1,0,0,0,0,0,0,0,0,1136
VEPA-
```

Figure 2.1: A screenshot of the occurrence matrix from the test database



```
Sample,Day,Run,AIB_IV,AIB_Os,AIB_IV_OS,AIB_inhale,cure_IV_3ans,Atcd_cure_IV,AIB_IV_OS_3mois,AIB_IV_3mois,AIB_OS_3mois,AIB_inhale_3mois,calprotectine,E
DUGA-J0,0,3,0,0,0,0,0,0,0,N,0,0,0,1,0,0,1,8,10,9,10
ROLE-J0,0,2,1,0,1,0,1,1,1,1,0,0,1,0,0,0,0,7,8,7,8
POLE-J90,90,2,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,8,8,7,8
BOMA-J45,45,1,1,0,1,0,0,0,1,1,0,1,N,0,0,0,1,10,10,10,10
CHLA-J90,90,1,0,0,0,0,0,0,0,0,0,N,1,1,0,0,1,8,9,6,7
DEMA-J0,0,3,0,0,0,1,0,1,0,0,0,1,0,1,0,0,1,9,9,7,9
GATE-J0,0,3,0,0,0,0,0,0,1,N,1,1,1,0,0,0,1,8,8,9,8
CHLA-J0,0,1,0,0,0,0,0,0,0,0,0,N,0,1,0,0,1,9,9,8,9
HIET-J45,45,3,1,0,1,1,0,1,1,1,0,1,1,1,0,0,0,10,10,8,10
VEPA-J90,90,2,1,0,1,1,0,1,0,0,0,1,0,1,0,1,0,8,8,7,9
SALO-J90,90,2,0,1,1,0,0,0,0,N,1,0,N,1,0,0,1,9,9,9,9
MEAL-J0,0,3,1,1,1,0,0,1,N,1,1,0,1,0,1,0,1,N,N,N,
GATE-J90,90,3,0,0,0,0,0,0,0,N,0,0,1,0,0,0,1,8,7,8,8
HIET-J90,90,3,1,0,1,1,0,1,1,1,0,1,1,0,0,0,0,10,10,10,10
URKE-J90,90,2,0,0,0,0,0,1,1,0,1,0,1,1,0,0,0,8,8,5,8
URKE-J0,0,2,0,0,0,0,0,1,1,0,1,0,1,1,0,0,0,7,8,9,8
DUGA-J90,90,3,0,0,0,0,0,0,0,N,1,0,1,1,0,0,1,9,10,9,10
MADA-J0,0,3,0,0,0,0,0,1,1,1,1,0,0,1,0,0,1,9,9,8,9
MALI-J90,90,3,0,0,0,0,0,0,0,N,0,0,1,1,0,0,1,7,9,7,9
POLE-J0,0,2,0,0,0,0,1,1,0,0,0,1,1,1,0,0,0,9,9,8,8
ROLE-J45,45,2,1,0,1,0,1,1,1,1,0,0,1,1,0,0,0,7,9,8,9
SUJA-J90,90,1,1,0,1,1,0,1,1,1,1,1,N,0,0,0,1,7,9,6,8
BRAL-J45,45,3,1,1,1,1,0,1,1,1,1,0,1,1,0,0,1,9,10,10,9
BOMA-J0,0,1,1,0,1,0,0,0,1,0,1,1,0,1,0,0,1,9,9,9,10
AMLA-J90,90,3,0,0,0,0,0,1,1,0,1,N,N,0,1,0,1,9,9,8,9
WIPA-J90,90,2,0,0,0,0,0,0,0,N,1,0,1,1,0,0,1,N,N,N,
VEPA-J0,0,2,1,0,1,1,0,1,0,0,0,1,0,1,0,1,0,8,8,8,8
BRAL-J0,0,3,1,1,1,1,0,1,0,0,0,0,0,0,0,0,1,8,9,7,10
BRAL-J90,90,3,1,1,1,1,0,1,1,1,1,0,1,1,0,0,1,9,9,9,10
BLANC,N,1,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
BRALTotal,0,2,1,1,1,1,0,1,0,0,0,0,0,0,0,0,1,8,9,7,10
KANA-J90,90,1,0,0,0,0,0,0,0,N,1,0,0,0,1,0,0,8,8,8,8
HIET-J0,0,3,1,0,1,1,0,1,1,1,0,1,0,1,0,0,0,9,10,9,10
DEMA-J90,90,3,0,0,0,1,0,1,0,0,0,1,1,1,0,0,1,9,9,7,9
SUJA-J45,45,1,1,0,1,1,0,1,1,1,1,1,N,0,0,0,1,7,8,8,7
```

Figure 2.2: A screenshot of the info matrix from the test database

2.2 Problems of clustering (CLust)

The following two problems deal with the clustering of the groups of samples by their respective microbial populations, and the comparison of such a clustering with the values of metadata for these groups, in order to find a potential correspondance between bacteria and metadata.

2.2.1 Problem of compatibility (CL.Comp)

CL.Comp is:

Supplementary input:

- The set of lists of nodes nodesS (called *matchingNodes*) such as nodesS[i] is the list of nodes matched in at least one of the reads in sample i.
- A subset N of the set of nodes in the taxonomic tree.
- A set of metadata M.

Output:

- Decide if the clustering of samples according to the microbial population restricted to N is compatible with the clustering of samples according to their values of metadata in M.

2.2.2 Problem of best clustering (CL.BCLust)

CL.BCLust is:

Supplementary input:

- The set of lists of nodes nodesS (called *matchingNodes*) such as nodesS[i] is the list of nodes matched in at least one of the reads in sample i.
- A set of metadata M.

Output:

- Find the best subset N of the set of nodes in T for a compatible classification of samples according to values of metadata in M.

Methods

Three different approaches have been designed, each of them solving one or more of these problems. In chronological order:

- First, a method, TaxoTree, that uses a statistical approach with customized measures of microbial populations, determines a distance between the samples to solve problem MDPairs. However, in order to be relevant, this method requires robust measures, and the biological phenomena are not so easy to describe with only a few characteristics.
- Second, an approach, called TaxoClassifier, that uses the *supervised learning* paradigm, and tries to identify nodes (of the taxonomic tree)/bacteria that discriminate the samples according to the values of metadata, that is, finds the nodes which seem to have a relationship with the considered metadata. Strong *a priori* hypotheses on microbial populations have to be made, which could bias the results, but this approach can solve problems CLust.
- Third a method, entitled TaxoCluster, that uses *non-supervised learning.* It clusters the samples using two distances only depending on node populations, and compares the resulting clusters with the groups of metadata values. It can solve problem CL.Comp.

For TaxoTree and TaxoCluster, we have put in pace a novel implementation of taxonomic trees (see Appendix C).

Chapter 3

Statistics-inspired approach (TaxoTree software)

3.1 Definitions

There are a few useful definitions to know:

1. A TREE INDUCED BY A GROUP OF SAMPLES G is the tree (or the forest) containing only nodes that have been assigned at least once in one of the samples in G (see figure 3.1).

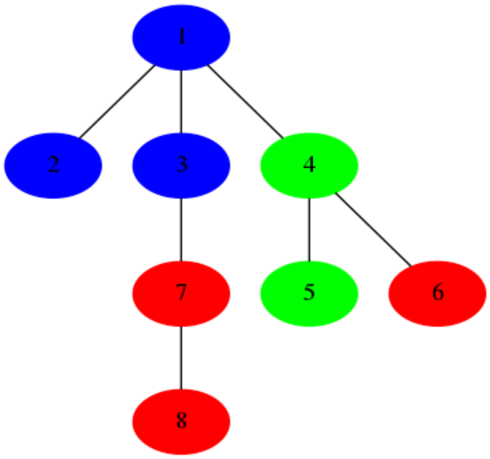


Figure 3.1: Subtrees induced by three groups of samples (each corresponding to a single colour). Incuced subtrees can of course overlap.

2. A PATTERN IN THE TREE A is a connected subgraph of A. For instance, the blue tree from the example above.

3.2 Description of the method

This approach tries to solve MDPairs, and computes a certain number of scores for a group of samples, independantly of the values of metadata. Each score corresponds to a partial definition of identity between two groups of samples. The union of those scores aims at quantifying the most accurately the difference between two sets of samples.

3.2.1 Total Ratio

- Total Ratio: provided two groups G_1 and G_2 of samples, if n is the number of assignments to common nodes in both groups (i.e. the sum of assignments in common nodes in each group), and n_1 et n_2, the number of assignments in these two groups in non-common nodes respectively, then Total Ratio is $TR(G_1, G_2) = \frac{n}{n_1+n_2+n}$ (see figure 3.2). If $n_1 = n_2 = n = 0$, it means that no read from the samples of G_1 and G_2 has matched a node in T. We choose to have in this case $TR = +\infty$.

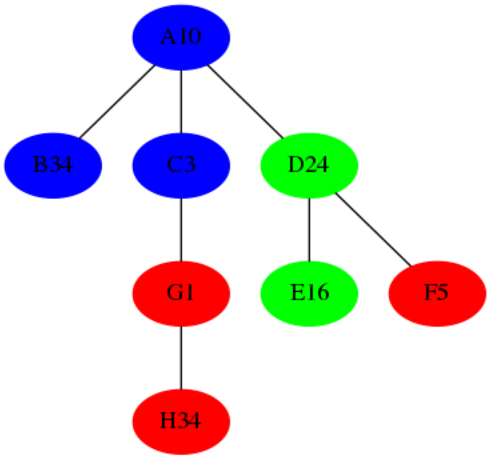


Figure 3.2: Example for Total Ratio: blue nodes are common to green and red trees. Green and red nodes correspond to two distinct groups of samples. Here, $TR = \frac{10+3+34}{(1+34+5)+(10+3+34)+(24+16)} \simeq 0.37$

Biological interpretation: Total Ratio Distance focuses on the node population in samples. If $n_1 + n_2 = 0$, it means that G_1 and G_2 have the same set of nodes and thus $TR = 1$. If $TR = 0$, it means G_1 and G_2 have no node in common.

3.2.2 Pattern Ratio

- Pattern Ratio: if n_{common} is the number of assignments in common patterns of length > 1 in the trees induced by the groups G_1 and G_2 of samples selected, and $n_{specific}$ the number of assignments in specific patterns, then Pattern Ratio is the quantity: $PR(G_1, G_2) = \frac{n_{common}}{n_{specific}}$ (see figure 3.3). If $n_{specific} = 0$, it means that the two trees induced by G_1 and G_2 are the same, so having $PR = +\infty$ is consistent.

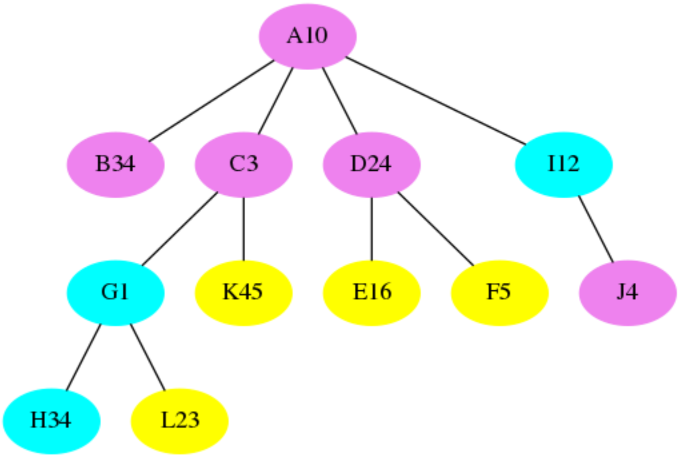


Figure 3.3: Example for Pattern Ratio: violet nodes are common to cyan and yellow trees. Cyan and yellow nodes correspond to two distinct groups of samples. Here, $PR = \frac{10+34+3+24}{(1+34+12)+(23+45+16+5)} \simeq 0.52$

Biological interpretation: Pattern Ratio focuses on the phylogenetic proximity between the nodes of the two groups. When computed for the whole set of samples, it corresponds to the 'functional kernel' of the gut, that is, a group of bacteria that is usually common to every human and that allow the different chemical reactions in the gut.

3.2.3 Microbial Diversity

- Microbial Diversity: provided a group of samples G, if n_{nodes} is the number of nodes in the tree induced by G, and n_{tree} the number of nodes in the whole taxonomic tree, then Microbial Diversity is: $MD(G) = \frac{n_{nodes}}{n_{tree}}$ (see figure 3.4). It is the definition of diversity used in [7], that must be distinguished from Phylogenetic Diversity [26] and Mean Diversity [27] coefficients.

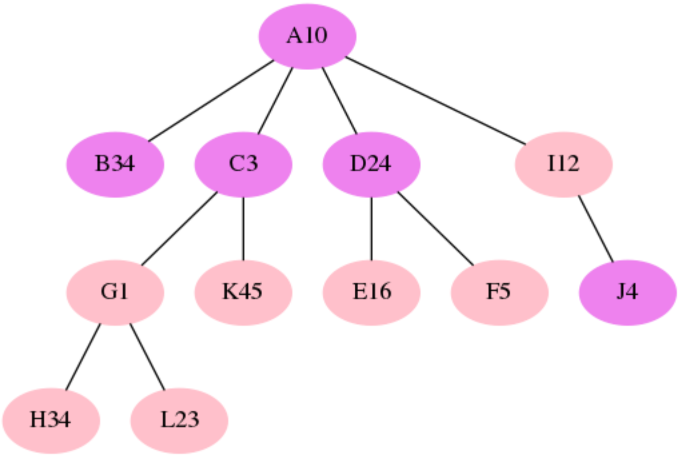


Figure 3.4: Example for MICROBIAL DIVERSITY: Here, MICROBIAL DIVERSITY coefficient for violet tree is: $MD = \frac{5}{12} \simeq 0.42$

> BIOLOGICAL INTERPRETATION: The diversity of bacteria for instance in guts is of paramount importance, because the equilibrium of the ecosystem is compulsory to ensure the good development of the gut. Therefore two samples having the same proeminent sorts of bacteria, but one having less diversity compared to the other, must be considered different. This coefficient may be quite superfluous in the light of the previous scores, but it seems justified by biology.

3.2.4 Computation of the similarity coefficient

The final similarity coefficient s between two groups of samples G_1 and G_2 (distance d can be obtained from s with the formula: $d = \frac{1}{s}$, $d = +\infty$ if $s = 0$) to compare the groups of patients is a mix of these scores. An equal importance here is accorded to each score in the following formula:

$$\text{if } MD(G_1) - MD(G_2) = 0\text{:}$$
$$s(G_1, G_2) = TR(G_1, G_2) + PR(G_1, G_2)$$
$$\text{else:}$$
$$s(G_1, G_2) = TR(G_1, G_2) + PR(G_1, G_2) - |MD(G_1) - MD(G_2)|$$

Then this similarity coefficient is normalized:

$$\overline{s}(G_1, G_2) = \frac{s(G_1,G_2)-E(s)}{\sigma(s)}$$

Then, if m_M is the number of classes induced by the values of a given metadata M, the program asks to cluster the set of samples in groups $(G_i)_{1 \leq i \leq m_M}$, each group corresponding to one single known value of the metadatum. After computing $\overline{s}$ for every pair of $(G_i)_{1 \leq i \leq m_M}$, it returns the list of pairs of groups $(G_i, G_j)_{i \neq j, 1 \leq i,j \leq m_M}$ such as $\overline{s}(G_i, G_j) \leq FQ$, where FQ is the value of the first quartile.

3.3 Implementation

Our implementation of TaxoTree in Python 2.9.7 offers a few other features:

1. After selection of a group of bacteria and of a group of samples/metadata, gives an array with the percentage of assignments to this family of bacteria depending on the samples/metadata.

2. Computes the Pearson correlation coefficient r between a number of assignments to a group of bacteria and a group of metadatas, or between the numbers of assignments to two groups of bacteria.

   Pearson sample product-moment correlation coefficient: [16] for vectors x and y of values of size n, if $\overline{x}$ is the mean of the $(x_i)_i$, and $\overline{y}$ the mean of the $(y_i)_i$, then $r = \frac{\sum_{i=1}^{n}(x_i - \overline{x}) \times (y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2} \times \sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}$. r is comprised between 1 and -1:

   - $r = 1$ meaning there is a perfect uphill linear relationship between x and y.
   - $r = -1$ meaning there is a perfect downhill linear relationship between x and y.
   - $r = 0$ meaning there is no linear relationship at all.

3. Draws graphs and pie charts.

4. Computes a distance coefficient $sim(k, l)$ between two patients P_k and P_l: after selecting a subset $M' = \{m_{i_1}, m_{i_2}, ...m_{i_m}\}$ of the set of metadata, if $n_{j,k}$ known value in P_k for m_{i_j}, then $dist(k, l) = \sum_{j=1}^{m} |n_{j,k} - n_{j,l}|$. The corresponding similarity coefficient is $sim(k, l) = \frac{1}{dist(k,l)}$, $sim(k, l) = +\infty$ iff $dist(k, l) = 0$.

Code is available at: *github.com/cbib/taxotree.*

The overall worst case time complexity of the computation of distance matrix is $O(n_{taxo-nodes}^2 \times n_{samples}^3)$ (see annex for more details). On our computer (see Appendix A for hardware characteristics), it took 10 minutes to compute it.

3.4 Results

3.4.1 Tests

The results have been compared to those obtained by statistical methods on the test database [7]:

- Comparison of microbial populations at Day 0: Output of the statistical analyses (chapter 5.2.2): Microbial populations between groups of samples without antibacterials (samples selected with parameters $Day = 0$, $ATB - IV = 0$) and with antibacterials (samples selected with parameters $Day = 0$, $ATB - IV = 1$) are said to be very alike according to the statistical results of the study. *Clostridium bolteae* and *Pediococcus acidilactici* were overrepresented in $ATB - IV = 1$ groups.

  According to the table and pie charts (see figure 3.5), the results seem to confirm the output of TaxoTree.

Table 3.1: Results from TaxoTree for the comparison at Day 0, (*) with Day = 0, (**) with (Clostridium bolteae,S),(Pediococcus acidilactici,S), where S is one of the taxonomic ranks

Score	Result	Parameters
Normalized Total Ratio	0.92837592759	ATB-IV=0,1 (*)
Pattern Ratio	9.46867603736	ATB-IV=0,1 (*)
Percentage Assignments	59.954342%	ATB-IV=1, option "bacteria" (**)
Percentage Assignments	5.877773%	ATB-IV=0, option "bacteria" (**)
Microbial Diversity	0.0227222589896	ATB-IV=0 (*)
Microbial Diversity	0.0196337966027	ATB-IV=1 (*)

- Evolution of microbial populations: Output of the statistical analyses (chapter 5.2.3): Microbial populations are said to change between Day 0, Day 45 and Day 90 for all patients, with/without $ATB-IV$. Nevertheless microbial diversity seemed not to be sensibly different between Day 0, Day 45 and Day 90 in patients being treated with antibacterials.

  The result of Total Ratio does not match the statistical results. However, Pattern Ratio shows the variations of the microbial populations between Day 0, Day 45 and Day 90. Indeed, when Pattern Ratio is largely superior to 1, it means the samples have similar node population and have got some similar clusters of nodes. It could mean that patients' guts recover their initial $Day - 0$ microbial population at Day 90, which would explain the high coefficient of similarity between Day 0 and Day 90 samples, while the $Day - 45$ samples have got a different node population than initially

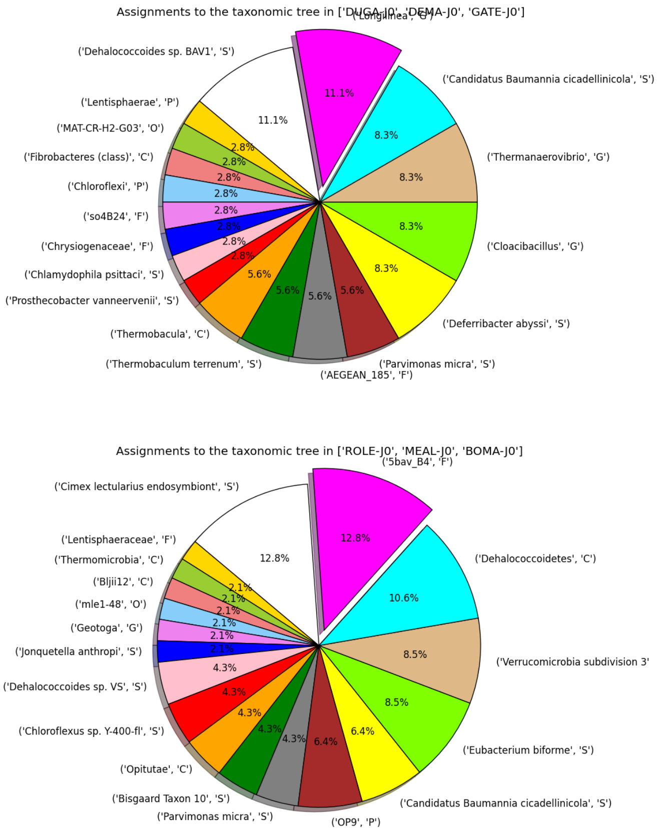


Figure 3.5: Microbial Diversity Day=0, ATB-IV=0 (top) and ATB-IV=1 (bottom)

Table 3.2: Results from TaxoTree for the comparison at Day 0, Day 45, Day 90

Score	Result	Parameters
Normalized Total Ratio	0.969346342836	Day=0,45
Normalized Total Ratio	0.929612804449	Day=45,90
Normalized Total Ratio	0.94998844733	Day=0,90
Pattern Ratio	3.75530416309	Day=0,45
Pattern Ratio	13.0716519314	Day=0,90
Pattern Ratio	1.9536860416	Day=45,90
Microbial Diversity	0.0196337966027	Day=0,ATB-IV=1
Microbial Diversity	0.0195234943746	Day=45,ATB-IV=1
Microbial Diversity	0.0163247297595	Day=90, ATB-IV=1

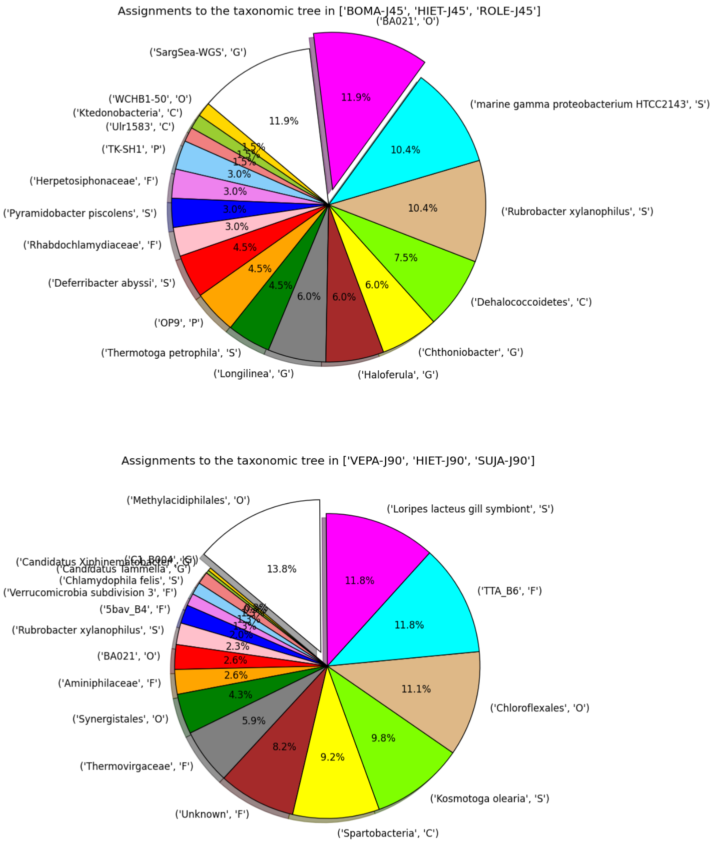
Table 3.3: Results from TaxoTree for the comparison at Day 90, (*) for Day 0, (**) for Day 90

Score	Result	Parameters
Normalized Total Ratio	0.92837592759	ATB-IV=0,1 (*)
Normalized Total Ratio	0.907051044683	ATB-IV=0,1 (**)
Pattern Ratio	9.46867603736	ATB-IV=0,1 (*)
Pattern Ratio	1.80113934679	ATB-IV=0,1 (**)
Microbial Diversity	0.0227222589896	ATB-IV=0 (*)
Microbial Diversity	0.0197440988308	ATB-IV=1 (*)
Microbial Diversity	0.0254798146923	ATB-IV=0 (**)
Microbial Diversity	0.0163247297595	ATB-IV=1 (**)

due to the treatment and a unknown cause for untreated patients (the paper underlines this problem for $ATB - IV = 0$ patients). Microbial diversity seems not to really change between Day 0, Day 45 and Day 90 for $ATB - IV = 1$ patients (see figure 3.6).

- Comparison of the groups of samples at Day 90: Output of the statistical analyses (chapter 5.2.4): $ATB - IV = 1$ and $ATB - IV = 0$ group still bear a significant difference.

  Although the microbial diversity is said to evolve in the same way in both $ATB - IV = 1$ and $ATB - IV = 0$ groups, here the gap between the two diversity coefficients is deeper at Day 90 than at Day 0. We cannot tell if it is a natural variation, or an effect of the antibacterials. The difference between the samples increases according to the variation of Pattern Ratio coefficient, meaning microbial populations have evolved (under antibacterials?) between the two groups. Here, Total Ratio does not give any information.


Assignments to the taxonomic tree in ['BOMA-J45', 'HIET-J45', 'ROLE-J45']

Assignments to the taxonomic tree in ['VEPA-J90', 'HIET-J90', 'SUJA-J90']

Figure 3.6: Microbial Diversity Day=0, ATB-IV=0 (top) and ATB-IV=1 (bottom)

3.4.2 Overview and discussion

About the method

The main difficulty when using this method is thus to find relevant measures to characterize the samples. Since there are still many unknown elements today about the biological mechanisms, we cannot be sure to have the best interpretation of these results. The algorithms below try to get rid of the *a priori* hypotheses we could have over the biological mechanisms, that may have influence on the resulting scores.

This method is particularly naive, and tends to be close to the previous statistical analyses.

About the numerical results

Although being the most intuitive score, Total Ratio measure appears not to be really relevant here. This can be explained by the fact that it does not take into account the proportions of assignments to nodes. As long as the two groups of samples own a same node, it is considered as a common node, even though group 1 might have many more reads assigned to this node than group 2. On the opposite, Pattern Ratio seems justified by biology and gives consistent results.

Some of these results contradict the statistical results. We would need to test it on other databases to know who is right.

Chapter 4

Supervised learning (TaxoClassifier software)

The section just below explains some important concepts, that will help to understand the second method, described in the following sections.

4.1 Machine Learning and supervised learning: the Naive Bayes classifier

This section aims at explaining what *Machine Learning* and *supervised Learning* are, and at describing the *Naive Bayes* classifier, and the YOUDEN'S J COEFFICIENT that will quantify the relevance of the classification.

4.1.1 Machine Learning

The last two approaches, TAXOCLASSIFIER and TAXOCLUSTER, use Machine Learning type algorithms. *Machine Learning* is a paradigm that automates the recognition of meaningful patterns in data [23], that is, the machine is technically able to sort the input according to certain criteria, without having the sorting explictly programmed.

For instance, it can be used to detect spam emails: the program searches throughout the emails keywords such as "fortune", "code bank", or "money". Then it evaluates (by computing the probability of having these keywords if the email is not a spam) whether the email should be discarded or not. If the probability of this email being a spam, with this number of keywords, is greater than the probability of this email being a regular email with these keywords, then the email is deleted. At the opposite, a 'regular' program would maybe need a threshold of number of occurrences above the one it discards the email. However, which value of threshold should we choose to be sure to keep (most of) our non-spam emails?

Along with the ability to deal with tremendously large files, *Machine Learning* has nowadays become a quite common tool in bioinformatics [25]. One of the ways to make the machine "learn" is called *supervised learning*.

4.1.2 Supervised Learning

Supervised learning algorithms try to classify data into fixed labeled categories, basing their decision on prior knownledge provided by a training set of data, e.g. the algorithm from the spam example above. A set of emails comprising safe mails and spams (where the spam or non-spam nature of the emails is indicated and constitutes labels for the categories) is given to the algorithm. It computes then (for instance) the probability of having occurrences of the words "fortune", "money", "love" in spam and non-spam emails. If the difference between the two probabilities is significant, it can help distinguish the unsafe mails out of the mailbox with the previous strategy.

One of the most used classifiers is the *Naive Bayes* one.

4.1.3 Naive Bayes Classifier

Given k (disjoint) classes of data $(C_i)_{1\leq i\leq k}$, a set of criteria $(F_i)_{1\leq i\leq m}$, and a datum d to classify having $(F_i = x_i)_{1\leq i\leq m}$, the *Naive Bayes classifier* [28] computes $(P(C_j|F_1 = x_1, ..., F_m = x_m))_{1\leq j\leq k}$, that is, the probability of d being in C_j having these values of $(F_i)_{1\leq i\leq k}$. Then d belongs to the class C_j such as:

$$P(C_j|F_1 = x_1, ..., F_m = x_m) = \max_{h\in\{1,...,k\}} P(C_h|F_1 = x_1, ...F_m = x_m).$$

Since we do not know these probabilities, using Bayes' theorem [28], conditional probabilities can be rewritten this way (let $(F = x)$ be $(F_1 = x_1, ..., F_m = x_m)$):

$$P(C_j|F = x) = \frac{P(C_j, F=x)}{P(F=x)}$$

Using the chain rule [28], we can get:

$$P(C_j, F_1 = x_1, ..., F_m = x_m)$$
$$= P(C_j)P(F_1 = x_1|C_j)P(F_2 = x_2|C_j, F_1 = x_1)...P(F_m = x_m|C_j, F_1 = x_1, F_2 = x_2, ..., F_{m-1} = x_{m-1})$$

Under the assumption of independance between the $(F_i)_i$ [28], this can be rewritten as follows:

$$P(C_j, F_1 = x_1, ..., F_m = x_m)$$
$$= P(C_j)P(F_1 = x_1|C_j)P(F_2 = x_2|C_j)...P(F_m = x_m|C_j)$$

Eventually, the probability we are looking for is:

$$P(C_j|F = x) = \frac{P(C_j)\prod_{i=1}^{m} P(F_i=x_i|C_j)}{P(F=x)}$$

Later in this report, we use to compute $P(F_i = x_i|C_j)$ the *Bernouilli model* [28]. This model has got a few drawbacks, but can be a good solution when relations between the $(F_i)_i$ are unknown, and when the $(F_i)_i$ can only have two (boolean) values.

If p_i is the probability of F_i being true, then:

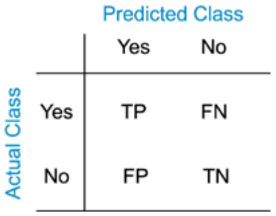
$$P(F_i = x_i|C_j) = p_i^{x_i}(1 - p_i)^{x_i}$$

4.1.4 Youden's J coefficient

The measure used to quantify the relevance of the classification here is Youden's J coefficient [33]. Note that many other measures exist, such as *F-measure* [20], *study of ROC space graph* [8], ... Since *F-measure* raises certain issues [18], and knowing that Youden's J coefficient is often used for medical diagnosis, we have considered the latter to be more helpful.

Some basics in statistics are required here. For a certain class C, we denote (see figure 4.1):

- $TP(C)$ the True Positive data, that is the data that belongs to C and is assigned by the algorithm to C.
- $TN(C)$ the True Negative data, not assigned to C and not belonging to C.
- $FN(C)$ the False Negative data, not assigned to C and belonging to C.
- $FP(C)$ the False Positive data, assigned to C and not belonging to C.



		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

Figure 4.1: Confusion matrix describing the four classes from gepsoft.com

Then the YOUDEN'S J COEFFICIENT for this class C is:

$$J(C) = \frac{TP(C)}{TP(C)+FN(C)} + \frac{TN(C)}{TN(C)+FP(C)} - 1.$$

It uses respectively the *recall/sensitivity*[20] (the proportion of objects assigned to the right class among those assigned to C) and the *precision/specificity*[20] (the proportion of objects that do not belong to C among those not assigned to this class).

- When $J(C) = 1$, it means that the classification is perfect (no FN and no FP).
- When $J(C) = 0$, it means that the test does not do any better than a random classifier.
- When $J(C) = -1$, it means that the test went all wrong (no TP and no TN).

4.2 Description of the method

TAXOCLASSIFIER tries to solve problems CL.COMP and CL.BCLUST. It uses a *Naive Bayes Classifier*, which attempts to classify the samples/patients in one of the groups of values of metadata depending on their microbial populations.

1. First the user is asked to give the metadata M which will partition the set of samples, and the node population N to consider in the classification. This sets the number of classes, according to the different values for each metadatum of M.

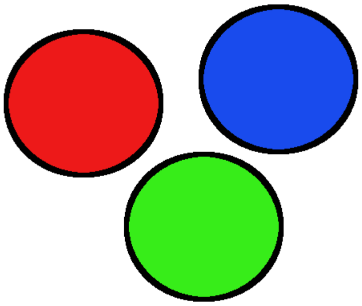


Figure 4.2: Let M_1 be a metadatum having values 0 (red), or 1 (green), or 2 (blue). Then there are three classes associated with M_1.

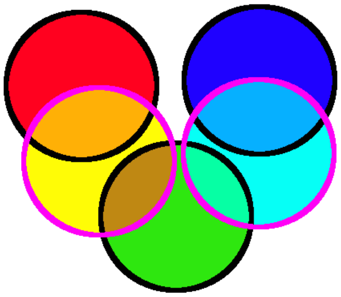


Figure 4.3: Let M_2 be a metadatum having values 0 (yellow), or 1 (cyan). Then there are two classes associated with M_2, that may intersect classes of M_1. Thus there are four classes associated with M_1 and M_2 (classes do not take into account unknown values)

2. Then a random strict subset R of the initial set of samples is chosen to be the *training set*. The size of this subset is chosen by the user. From this training set, the algorithm computes an estimated probability of having a certain node for each node in N (corresponding to the p_i described in the previous chapter), and also for each sample s and for each node n, if n appears in s (this information is stored as a boolean value 0 or 1: with the previous notations, if F_i is associated with node n, whether $F_i = 1 = x_i$ in sample s).

3. Hence for every sample not in R (since samples in R are already assigned), under the hypothesis of equiprobability of being in a certain class, the algorithm computes the probability of being in class C knowing the actual presence of the nodes in N for this sample, for each class C.

4. The algorithm eventually assigns the sample to the class which maximizes this probability.

The relevance of the resulting classification f_{class} is quantified by the YOUDEN'S J COEFFICIENT $j_{f_{class}}$, pretty much like above. The best classification (that hightlights a significant correspondance between the values of the selected metadata and the microbial populations) is the classification such as, for all class C, $J_{f_{class}}(C)$ is the closest to 1.

In other words, if there are k classes $(C_i)_{1 \leq i \leq k}$,

$$k - \sum_{i=1}^{k} J_{f_{class}}(C_i) \text{ is minimum and non-negative.}$$

4.3 Implementation

This method has also been implemented in PYTHON 2.9.7. The program allows two operations, each of them solving either CL.C or CL.BCL:

- The user chooses the sets M and N like above, and the program returns the YOUDEN'S J COEFFICIENT associated with the resulting classification, giving a hint about a possible correspondance between M and N.
- The user chooses two integers s and n, the set M. Then the program randomly picks s times n distinct nodes in the taxonomic tree, and returns the set N of nodes of size n which ensures the best classification (among the s tries) for M. It may give a hint of a possible correspondance between M and the resulting set N.

The two main difficulties with the K-MEANS algorithm is choosing k and the initialization of the clusters. Fortunately, our method solves these problems since k is the number of metadata classes associated with the user-selected metadata, and each cluster is initialized with a sample having the correct values of metadata associated with this cluster.

To deal with multiple metadata classes (see the example above), multi-dimensional lists have been implemented (see annex).

The overall worst case time complexity for a classifiation is roughly in O($n_{samples}^2 \times n_{taxo-nodes}^2 \times n_{values}$) (see annex for more details).

(Still in progress) code can be found at:

github.com/kuredatan/taxoclassifier.

4.4 Results

4.4.1 Tests

Due to time limitations, tests were not yet completed.

4.4.2 Overview and discussion

About the method

Although the algorithm classifies samples not by focusing on specific characteristics of node population, but by directly comparing the microbial populations (which is theoretically better than the previous program), we still have to do *a* *priori* hypotheses on the set of samples.

First, we suppose the equiprobability of being in a random class, that is, equiprobability of having a certain value for the selected metadatum. However, not only does it strongly depend on the values of other metadata, but it would even be sometimes completely wrong according to the metadatum considered. In our database for instance, $ATB - IV = 0$ or $ATB - Os = 0$ imply $ATB - IV - Os = 0$: $ATB - IV = 0$ means that the patient has not been treated with antibacterials by intraveinous injections (IV), $ATB - Os = 0$ means that the patient has not been treated with antibacterials *per os*, and $ATB - IV - Os = 0$ means that the patient has not been treated with antibacterials *per os*, neither IV. Unfortunately, since we do not know the exact relationship between the different metadata (e.g. does the treatment really affect the quality of life of the patient?), we cannot fix this issue.

Same goes for the hypothesis of independance between the probabilities of having a certain node, that is at core of the Bayes conditional model. Biologically speaking, we can wonder if having a certain species of bacteria may prevent another species to develop itself in the gut.

Second, the size and the content of the starting set for the classification given by the *Naive Bayes Classifier* is of paramount importance. Prior probabilities of having a certain node are computed from this starting set by default (because assuming equiprobability would lead to contradictions, since we know antibacterials for instance do affect the microbial population). If the starting set is badly chosen, some nodes might not have matched any read in the starting group of samples, and thus the probability of having such a node is theoretically zero. Then the Bernouilli model would always return zero for a probability of being in a certain class, if this node is selected.

We thus let the user choose themselves the size of the starting set, provided the total size of the set of samples to classify.

A solution to this problem is to use the *Bayesian average* [14], that is, if the probability p_n of having the node n computed from the starting set S_{start} and:

- N_n be the number of samples having node n among the samples in the starting set,
- Boolean variable $x_{i,j} = 1$ iff node j is matched in sample i (else $x_{i,j} = 0$),
- $M_n = \frac{N_n}{|S_{start}|}$,
- $v_n = \sqrt{\sum_{i=1}^{S_{start}} (x_{i,n} - M_n)^2}$,
- $C_n = \frac{|S_{start}|}{v_n + 1}$,

- $m_n = \frac{C_n}{|S_{start}|}$,

$$\text{Then } p_n = \frac{C_n.m_n + N_n}{C_n + |S_{start}|}.$$

The idea of *Bayesian average* is not to fully rely on the samples of the starting set to compute the probabilities of having a certain node. C_n is a coefficient denoting how much confident we are in the probabilities given by the starting set. The more we believe the probabilities are distributed on the whole set of samples the same way they do in the starting set, the larger is C_n. This is quantified by the variation of values in the starting set: if most of the samples have matched this node -that is, variation is low- it is likely that all samples may actually own this node.

Note that the maximum value of C_n is $|S_{start}|$. Having $C_n = |S_{start}|$ (that is $v_n = 0$), would mean that we completely trust the starting set (replacing C_n in the formula above leads to $p_n = \frac{|S_{start}|+N}{2.|S_{start}|}$), whereas $C_n = 0$ gives $p_n = \frac{N_n}{|S_{start}|}$, that is, we consider the non-amplified values of probabilities from the starting set.

For instance, if node n matches in 5 samples out of 17, having $C_n = 0$ would give $p_n = \frac{5}{17} \simeq 0.29$, while having $C_n = |S_{start}|$ gives $p_n = \frac{17+5}{2\times 17} \simeq 0.65$. If node n does not match in any of the 17 samples, then for $C_n = \frac{|S_{start}|}{2}$, $p_n = \frac{|S_{start}|}{4\times|S_{start}|} = \frac{1}{4} = 0.25$.

Chapter 5

Non-supervised learning (TaxoCluster software)

The following sections introduce some useful concepts and notations, that will help to understand the third method described at the end of this chapter.

5.1 Machine Learning and non-supervised learning: the K-Means algorithm

This section explains what *non-supervised learning* is in *Machine Learning*, and present one of the most common tool in this category, which is the *K-Means algorithm*.

5.1.1 Non-Supervised Learning

Another broad category in *Machine Learning* is *non-supervised learning*: unlike *supervised learning*, the different classes to which belong the data are not known at first. Provided the set of elements, the algorithm has to distinguish by itself several different classes according to the similarity between the pieces of data from the initial set. This is why one of the most common approach for *non-supervised learning* is *clustering*.

5.1.2 Clustering

Given a certain distance, *clustering* is the task of gathering objects into disjoint clusters, such that the resulting sets maximize the proximity (in terms of the chosen distance) between elements of a same cluster, and maximize the distance between objects of different clusters.

5.1.3 K-Means Algorithm

Although the problem of partitionning n elements into k clusters [6] is NP-complete[30] (see the note in bibliography for a definition of NP-hardness and NP-completeness), the *K-Means algorithm* [12] is a rather efficient clustering algorithm. After choosing an integer k that is the estimated number of clusters, the user initializes each cluster with one of the elements of the set. Then, for each non-clustered object, the algorithm computes the distance from this object to the mean of every cluster (in our implementation, it is the sample that minimizes the sum of all distances to the other samples in the same cluster), and assigns the object to the closest cluster. Then it updates the mean of this cluster, and iterates the last two steps until convergence of the solution. For more detailed explanation, see [12].

5.2 Definitions

Before starting to describe our TaxoCluster method, here are some helpful notations (adapted from [4]):

5.2.1 Notations

Knowing that T is the whole taxonomic tree, for a certain read i, let M_i be the set of matched leaves for this read, and T_i the subtree of T rooted at the LCA (see above for the definition of LCA) of the nodes in M_i. Let L_i be the set of leaves of T_i, and N_i be such as $L_i = M_i \sqcup N_i$ (M_i and N_i are disjoint, see figure 5.1).

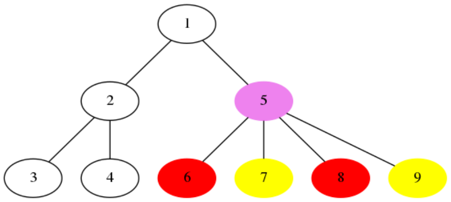


Figure 5.1: Let T be the whole taxonomic tree. Then for the set of red nodes, the subtree associated is rooted at the violet node, and the total set of leaves for this subtree is the set of yellow and red nodes.

5.2.2 Distances

Let us define two distances $d_{matched}$ and $d_{consensus}$ over pairs of reads (R_i, R_j) for the clustering:

- $d_{matched}(R_i, R_j) = |M_i| + |M_j| - 2 * |M_i \cap M_j|$.

It is quite easy to check that $d_{matched}$ is indeed a distance: $d_{matched}$ is symmetric, satisfies the triangle inequality, is non-negative, and $d_{matched}(i, j)$ equals to zero iff $R_i = R_j$ (The unique relevant equality relation between reads here is the equality between the respective sets of matched nodes, because nobody can know to which node the read should truly be assigned).

- Having a fixed parameter $q \in [0, 1]$,

  $$d_{consensus}(R_i, R_j) = |L_i| + |L_j| - q * (|N_i \cap M_j| + |N_j \cap M_i|) - |M_i \cap M_j|.$$

  It is also easy to check here that $d_{consensus}$ is a distance. This distance corresponds to search a consensus taxonomic tree between the trees induced by the set of nodes matched by the reads [1].

  When $q = 0$, we consider a *strict* consensus tree, only keeping leaves matched in both reads.
  When $q = 1$, we consider a tree having leaves that are matched in at least one of the two reads (see figure 5.2).

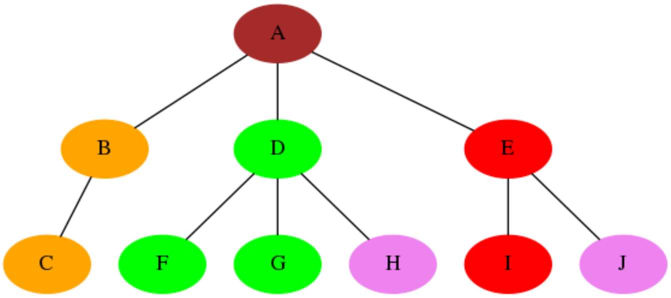


Figure 5.2: Let us consider the trees T_1 and T_2 both rooted at A, such as $M_1 = \{$F, G, H, J$\}$ and $M_2 = \{$H, I, J$\}$. Then $d_{consensus}(T_1, T_2) = 5+5-q\times(1+2)-2$. When $q = 0$, $d_{consensus}(T_1, T_2) = 8$. When $q = 1$, $d_{consensus}(T_1, T_2) = 5$

These distances can easily be extended to samples: if $Reads_k$ is the set of reads for sample S_k,

$$d_{consensus}(S_k, S_l) = \sum r_k \in Reads_k \sum r_l \in Reads_l d_{consensus}(r_k, r_l).$$

Same goes for $d_{matched}$.

5.3 Description of the method

TaxoCluster tries to solve problem CL.Comp. It compares trees induced by samples before the assignment of reads, and avoids providing *a priori* hypotheses on the probabilities of being in a certain class of metadata values, or of having a certain node. The number k of clusters used in the K-Means algorithm is thus the number of vectors of metadata that can be obtained.

We use the very same definition of classes of metadata as in the second method.

- First the set of samples is clustered into k clusters using $d_{matched}$ distance.
- For every cluster, the most remote elements are deleted from the cluster, in order to keep only relevant elements. In our implementation, we discard the points for which sum of all distances to other samples of the same cluster is above the value of the third quartile (of the list of such distances).
- Then the union of the remaining elements in all clusters is clustered again into k clusters using $d_{consensus}$ distance.

The clusters resulting from the second clustering are then compared to the clusters obtained by directly looking at the values of metadata in samples. If the two groups of clusters are alike (this is quantified by a special distance), this could mean that there exists a correspondance between the selected metadata and the microbial populations. The distance to compare two clusters C_1 and C_2 is d(1,2) = $\frac{|C_1 \cap C_2|}{|C_1|+|C_2|-|C_1 \cap C_2|}$ (if both clusters are empty, it returns None). The overall distance is (if one of the clusters is empty, then it returns None) $d_{clusters} = \frac{\sum_{1 \leq i < j \leq k} d(i,j)}{k}$, where k is the total number of clusters. The program also returns (on demand) the bacteria in common for samples in a same cluster.

5.4 Implementation

The whole pipeline has been implemented in Python 2.9.7, and allows to draw and see the clusters in a DOT file.

The worst case time complexity is O($n_{samples}^2 \times (n_{samples} \times n_{paths} \times n_{taxo-nodes} + n_{taxo-nodes}^2) \times n_{samples}^2$) (see annex for details).

(Still in progress) code can be seen at:

github.com/kuredatan/taxocluster.

5.5 Results

5.5.1 Tests

The current implementation of TaxoCluster can only cluster the samples according to the values of a single metadatum. With the use of MDL or Multi-Dimensional Lists (see Appendix D), it would be able to handle different metadata. As for the previous method, tests were not yet completed.

The tests that could be executed on our test database show that the computation of the distance $d_{consensus}$ does not depend on the value of q: indeed, the LCA is too high in the taxonomic tree (has rank R or K), due to the large number of nodes matched by each patient. A solution would be to count the number of matches for every node, and to only keep those which have been matched the most.

5.5.2 Overview and discussion

About the method

However, the main issue with this method is that, unlike the first *Machine Learning* approach, it does not give the nodes that really discriminate the samples. We can only get access to the list of common bacteria for samples in a same cluster. One solution would be to post-process the clusters with a PCA (*Principal Component Analysis*) method [17], or to test this pipeline with another clustering algorithm (different from *K-Means*).

Chapter 6

Comparison of the three approaches

Theoretically, TaxoCluster is the best method, as it aims at fixing the issues in the first two methods TaxoTree and TaxoClassifier. Indeed, it does not require a special measure on microbial populations, that would induce *a priori* hypotheses on the way these populations should be, with respect to the values of metadata. This fact is of paramount importance as most of the biological phenomena still remain partially unknown. Moreover, it does not require strong mathematical hypotheses as it is the case for the *Naive Bayes Classifier*. However, the latter allows more flexibility, whereas it might be irrelevant if the problem is specific enough.

Notice that TaxoCluster can only solve the Problem of Clustering Compatibility (CL.Comp), and not the Problem of Best Clustering (CL.BClust) unlike TaxoClassifier. Furthermore, in terms of worst case time complexity, knowing the actual values for each variable in our test database, it is unfortunately the slowest algorithm:

It is worth noticing that nevertheless the three methods can be completely executed on the test computer, even it is just a regular computer (see Appendix A). Nevertheless, to fully validate the results from the algorithms, we should

Table 6.1: Complexity for each method: for our test database, $n_{taxo-nodes} = 9,065$, $n_{values} \leq 10$, $n_{samples} = 47$ and $n_{paths} = 5,565$

Method	Theoretical complexity
Statistics	$\mathrm{O}(n_{taxo-nodes}^2 \times n_{samples}^3)$
Supervised learning	$\mathrm{O}(n_{samples}^2 \times n_{taxo-nodes}^2 \times n_{values})$
Non-supervised learning	$\mathrm{O}(n_{samples}^2 \times n_{taxo-nodes} \times (n_{samples} \times n_{paths}))$

get access to several other databases, and compare them to current biological knowledge and to the output of statistical methods. Due to time limitations, we unfortunately could not use any other database to perform these necessary tests. Moreover, the consistency of results greatly depends on the previous operations applied to input data (have the numeric results been normalized prior to the algorithm? Have all raw material for samples been collected the same way, following a same standard procedure? Has each assignment of read to nodes been performed under the same parameters? and so on). And, last but not least, the final interpretation of the results cannot still be taken into account without the practitionner or a practised biologist's validation.

Chapter 7

Outlook

7.1 Personal work

During this internship, we suggested and implemented three methods to analyse data from sequencing according to the metadata provided: TaxoTree, TaxoClassifier and TaxoCluster, each of them using a different general paradigm of computer science. We tried to design each method as an improvement of the previous one. These methods solve various clustering-like problems that often occur in biology and medicine. Although their time complexity can probably be improved, the numerical tests done so far appear to be consistent with the medical observations.

We also have implemented a class of multi-dimensional lists, and a rather efficient -in regards to the size of our data...- algorithm to reconstruct taxonomic trees from the list of paths from root to every leaf.

7.2 Comments

Use of algorithms from *Machine Learning* in metagenomics is not so new [25]. However, the last two approaches use a hitherto unseen mix between the common algorithms of *supervised* and *non-supervised learning*, statistics and tree algorithms.

On the one hand, a quite large number of comparison problems over labeled unordered trees -taxonomic trees being a special case of this sort of trees- are NP-complete, for instance the problem of tree inclusion [11]. What makes possible efficient algorithms is that the relevant comparison over trees here mainly focuses on the set of leaves for each tree. However, the time complexity of the last two methods, that is TaxoClassifier and TaxoCluster, should still be improved.

On the other hand, it would be interesting to test these algorithms on other databases than the one of cystic fibrosis-afflicted patients, so as to see if the resulting correspondances given by the softwares are relevant.

The code for all three methods is available on GitHub.

Bibliography

[1] E. Adams. Consensus techniques and the comparison of taxonomic trees. *Syst. Tool.*, 21, 1972.

[2] D. Alonso-Alemany, A. Barré, S. Beretta, P. Bonizzoni, M. Nikolski, and G. Valiente. Further steps in tango: Improved taxonomic assignment in metagenomics. *Bioinformatics Advance Access*, 2013.

[3] D. Barnard, G. Clarke, and N. Duncan. Tree-to-tree correction for document trees. Technical report, 1995.

[4] J. Clemente, J. Jansson, and G. Valiente. Flexible taxonomic assignment of ambiguous sequencing. *BMC Bioinformatics*, 2011.

[5] M. Cooper. Advanced bash-scripting (online), 2014.

[6] S. Dasgupta. The hardness of k-means clustering. Technical report.

[7] R. Enaud. *Impact de l'antibiothérapie sur le microbiote intestinal chez l'enfant atteint de mucovisidose*. PhD thesis, 2016.

[8] T. Fawcett. An introduction to roc analysis. 2005.

[9] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *Society for Industrial and Applied Mathematics*, 1984.

[10] T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. *Theoretical Computer Science*, 1994.

[11] P. Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, 1992.

[12] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[13] H. Mann and D. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1), 1947.

[14] P. Masurel. Bayesian rating. fulmicoton.com.

[15] Q. McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, (12), 1947.

[16] K. Pearson. Notes on regression and inheritance in the case of two parents. 58, 1895.

[17] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2, 1901.

[18] D. Powers. What the f-measure doesn't measure... Technical report, 2014.

[19] J. Rennie, L. Shih, and D. Karger. Tackling the poor assumptions of naive bayes classifiers. 2003.

[20] C. Van Rijsbergen. *Information Retrieval (2nd ed.)*. Butterworth, 1979.

[21] D. Robinson and L. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53, 1981.

[22] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 1987.

[23] S. Shalev-Shwartz and S. Ben-David. Understanding machine learning: From theory to algorithms. 2014.

[24] R. Sokal and C. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38, 1958.

[25] H. Soueidan and M. Nikolski. Machine learning for metagenomics: methods and tools. *Metagenomics*, 2016.

[26] M. Vellend, W. Cornwell, K. Magnuson-Ford, and A. Mooers. Measuring phylogenetic biodiversity. 2015.

[27] M. Vellend, W. Cornwell, K. Magnuson-Ford, and A. Mooers. Measuring phylogenetic biodiversity. 2015.

[28] Wikipedia. Naive bayes classifier.

[29] Wikipedia. Naive bayes classifier.

[30] Wikipedia. Np-hardness.

[31] Wikipedia. Séquençage de l'adn.

[32] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6), 1945.

[33] W. Youden. Index for rating diagnostic tests. *Cancer*, 3, 1950.

[34] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *Society for Industrial and Applied Mathematics*, 18(6), 1989.

Annex

Here are gathered details about the hardware limits of our computer for testing, detailed complexity function by function for each of the three methods described above, different algorithms to reconstruct a taxonomic tree as well as the implementation of multi-dimensional lists used in two out of the three methods.

Appendix A

Hardware characteristics of the testing computer

- Processor: Intel Core i3-3227U 3rd generation.
- Clock speed: 1.90 GHz.
- 3rd level cache: 3 Mo.
- Hard Disk: 1 To, rotation speed: 5,400 r/min.
- RAM: 6 Go, max: 16 Mo, DDR3 1,600 MHz.

Appendix B

Complexity

This chapter give details about the complexity of each method, per function and for the whole pipeline.

B.1 Worst case complexity for the first approach (TaxoTree)

Let $n_{samples}$ be the total number of available samples (in our test database, $n_{samples} = 47$), and $n_{taxo-nodes}$ the number of nodes in the taxonomic tree ($n_{taxo-nodes} = 9,065$).

B.1.1 Per function

- TOTAL RATIO: See functions *misc/inSample*, *misc/takeNodesInTree*, *totalRatio/compute* and *totalRatio/countAssignmentsInCommon*.

  inSample: @sampleNameList (the list of user-selected samples) can be as large as the whole list of samples: O($n_{samples}$).

  takeNodesInTree: This procedure does a top-down search in the taxonomic tree, and executes *inSample* for each list of assignments, that is, for each node of the taxonomic tree: O($n_{taxo-nodes} \times n_{samples}$).

  compute: *compute* executes twice *takeNodesInTree*. The member function *mem* is in O($n_{taxo-nodes}$). In the worst case where the two user-selected lists are the same, the following operations are in O($sizeofthefirstsubforest$) $=$ O($n_{taxo-nodes}$). So *compute* is in O($n_{taxo-nodes} \times n_{samples}$).

  countAssignmentsInCommon: It can easily be seen that *countAssignmentsInCommon* is in O($n_{taxo-nodes}.n_{samples}^2$).

The overall complexity is thus O$(n_{taxo-nodes} \times n^2_{samples})$. The most costly part is *countAssignmentsInCommon*.

- Pattern Ratio: See functions in *patternRatio* and *taxoTree* modules.

  TaxoTree.search: The procedure does a top-down search in the tree for the node in argument: O$(n_{taxo-nodes})$.

  misc/mergeList: The procedure takes two lists of length n and m, sorts both of them and then merge the two lists without duplicates (assuming there is no duplicate in the two lists): O$(nlog(n) + mlog(m))$.

  misc/trimList: This procedure takes two lists of length n and m, and deletes from the first list elements that belongs to the second one. It sorts both of the lists, and then considers in linear time the elements of the two lists. Thus the time complexity is O$(nlog(n) + mlog(m))$.

  enumerateCommonPatterns: This function uses *takeNodesInTree* and considers every node from the tree induced by the user-selected set of samples as a potential starting point for a pattern, and then tries to spread the pattern to the node's children, if it is possible (that is, if the child is matched at least in one of the samples in the user-selected list: the test is in O$(n_{samples})$). The two lists of assignments to this node are merged using *mergeList* procedure: O$(n_{samples}log(n_{samples}))$. Final time complexity is thus O$(n_{taxo-nodes} \times n_{samples} + n^2_{taxo-nodes} \times (n_{samples} + n_{samples}log(n_{samples}))) =$ O$(n^2_{taxo-nodes} \times n_{samples}log(n_{samples}))$.

  enumerateSpecificPatterns: This function takes into argument the two user-selected lists of samples, uses *trimList* procedure to have disjoint lists (such as one pattern of the first list does not belong to the second list, and the other way around), and then executes pretty much the same procedure as *enumerateCommonPatterns*. Thus the time complexity is O$(n^2_{taxo-nodes} \times n_{samples} + n_{samples}log(n_{samples}))$.

  patternRatio: This function counts the number of assignments for all (common or specific) patterns. These patterns are disjoint, by construction, so in the worst case, it counts assignments for every node of the taxonomic tree: O$(n_{taxo-nodes})$.

  Henceforth using the Pattern Ratio procedure is
  in O$(n^2_{taxo-nodes} \times n_{samples}log(n_{samples}))$ time.

- Microbial Diversity: See *computeDiversityCoefficient* function in *diversityCoefficient* modules.

computeDiversityCoefficient: This procedure uses *takeNodesInTree* (see the chapter above). It considers every node in the tree induced by the user-selected list of samples, and counts the total number of assignments to this node in samples belonging to the list (O($n_{samples}$) if the list of samples contains all available samples).

Thus the overall complexity is O($n_{taxo-nodes} \times n_{samples}$).

B.1.2 Overall complexity

Therefore, to compute the distance described above for every pair of samples ($\frac{n_{samples}(n_{samples}-1)}{2}$ pairs, distance being symmetric):

The time complexity is roughly in O($n^2_{taxo-nodes} \times n^3_{samples}$).

B.2 Worst case complexity for second approach (TaxoClassifier)

Without loss of generality, complexity is evaluated without multi-dimensional lists, since one metadatum can have an arbitrary number of different values.

Let $n_{samples}$ be the total number of available samples (in our test database, $n_{samples} = 47$), and $n_{taxo-nodes}$ the number of nodes in the taxonomic tree ($n_{taxo-nodes} = 9,000$). Let n_{values} the maximum number of effective values in the information matrix that a metadatum can have, that is, the maximum number of classes -in our test database, n_{values} is bounded by 10, metadatum *Sample* excepted. Let eventually $n_{metadata}$ the total number of metadata ($n_{metadata} = 21$).

B.2.1 Per function

- Training: See functions in *training* module.

computeClasses: See *misc/partitionSampleByMetadatumValue*: O($n_{samples}log(n_{samples}) + n_{metadata}$).

selectTrainingSample: O($n_{samples}$) (Reservoir Sampling R algorithm).

assignClass: O($n^2_{samples} \times n_{values}$).

getPriorProbability: O($n^2_{samples} \times n^2_{taxo-nodes}$).

The overall complexity is hence O($n^2_{samples} \times n^2_{taxo-nodes} + n^2_{samples} \times n_{values} + n_{metadata}$)
= O($n^2_{samples} \times n^2_{taxo-nodes}$).

- Classification: See functions in *classifier* module.

  probabilityKnowingClass: O($n_{taxo-nodes} \times n_{samples} \times n_{taxo-nodes}$)
  = O($n^2_{taxo-nodes} \times n_{samples}$).

  bayesCalculus: The most costly part is the call to probabilityKnowingClass: O($n^2_{taxo-nodes} \times n_{samples}$).

  classifyIt: with the call to trainingPart: O($n^2_{samples} \times n^2_{taxo-nodes} \times n_{values}$).

- Computation of Youden's J coefficient: See *countYouden* function in *youden* module.

  The whole set of classes contains at maximum the $n_{samples}$ samples. Thus the loop over the classes is in fact a loop over the set of samples. Then the overall time complexity is O($n^2_{samples} + n_{values}$).

B.2.2 Overall complexity

The worst case time complexity is roughly in O($n^2_{samples} \times n^2_{taxo-nodes} \times n_{values}$).

B.3 Worst case complexity for third approach (TaxoCluster)

Same notations as in previous chapter are used here. The most costly part in the clustering is the call to K-means, the computation of the two distance matrix, and the comparison between clusters.

B.3.1 Per function

- Distances: Between two given samples:

  - Distance 1: The maximum length of the list of matching nodes is $n_{taxo-nodes}$ (lists of matching nodes are stored in a dictionary). Thus time complexity is in O($n^2_{taxo-nodes}$).

Thus the computation of the whole distance matrix has got a worst case time complexity of $O(n^2_{taxo-nodes} \times n^2_{samples})$.

  - Distance 2: The computation of the LCA of a list of nodes of length m (m is bounded by $n_{samples}$) is in $O(m \times n_{paths} \times l_{path})$ (see *misc/taxoLCA*). The search of subtrees is in $O(n_{taxo-nodes})$ (see above), as well as the call to *TaxoTree.leaves* that counts leaves of the tree. The final loop is in $O(n_{taxo-nodes})$.

    Thus the computation of the whole distance matrix has got a worst case time complexity of $O(n^2_{samples} \times (n_{samples} \times n_{paths} \times l_{path}$ - $+ n_{taxo-nodes}))$.

- K-means: $O((n_{samples} - n_{values}) \times n_{values} \times n_{iterations})$ where $n_{iterations}$ is the number of needed iterations until convergence (30).

- Cluster comparison: For two given clusters (maximum sum of both lengths being $n_{samples}$: thus maximum value of the product of the lengths is $(\frac{n_{samples}}{2})^2$):

  compareCluster: $O(n^2_{samples})$

  compareCenters: for two given clusters: O(1) (storing distances between samples in a dictionary).

B.3.2 Overall complexity

Thus the worst case time complexity is
$O((n_{samples} - n_{values}) \times n_{values} \times n_{iterations} + n^2_{samples} \times n^2_{values} + n^2_{samples}(n_{samples} \times n_{paths} \times l_{path} + n_{taxo-nodes}) + n^2_{taxo-nodes} \times n^2_{samples})$
$= O(n^2_{samples} \times (n_{samples} \times n_{paths} \times n_{taxo-nodes} + n^2_{taxo-nodes} n^2_{samples}))$.

Appendix C

Construction of taxonomic trees

C.1 State-of-the-art and input

There are of course many algorithms to construct a taxonomic tree, often starting from a distance matrix between the *taxa* (plural of *taxon*, that is, nodes of the taxonomic tree); for instance, NEIGHBOR JOINING [22], or UNWEIGHTED/WEIGHTED PAIR GROUP METHOD WITH ARITHMETIC MEAN [24]. However, here we have already got the whole taxonomic tree and the phylogenetic relation between the *taxa*.

Provided a list of paths *paths* from the root to every leaf of the taxonomic tree, we thus need an efficient algorithm to construct the tree, which must grant easy (i.e. as cheap as possible) access to the list of assignments to node, to the children and also to the phylogeny of the node (the so-called "lineage"). It should also provide labelling of the nodes by an integer identifier. Since we could not find an article having dealt with this subject (trace analysis do use trees, but since we are interested in a very special sort of tree (taxonomic trees), it would be a bit off topic here), we had to design a customized algorithm.

Let $n_{taxo-nodes}$ be the number of the nodes in the taxonomic tree (in our test database, $n_{taxo-nodes} = 9,000$), $n_{samples}$ the number of samples ($n_{samples} = 47$), and n_{paths} the number of paths from the root to every leaf (as it is a tree, this number is as well the number of leaves in the tree, that is $n_{paths} \simeq 8,000$). Let also l_{path} be the maximum length between the root and a leaf (that is the number of ranks: in our test database, $l_{paths} = 8$), and $n_{metadata}$ the number of metadata ($n_{metadata} = 21$).

We present three methods in chronological order of design:

- The first one is a top-down construction, where we go through every path from *root* to a leaf, and construct the missing nodes.

- Unlike the method above, the second one is a bottom-up construction, where for each taxonomic rank (from S to R, that is from the most accurate to the most broad rank), we gather siblings (that is nodes with a same father) and construct the corresponding trees until having the tree rooted at *root*.

- The third one is an improvement of the second method. The first two methods are quite time-consuming, and this one succeeds in being the fastest of the three. Preprocess is applied to the nodes to precompute the groups of siblings and the father node associated, using hash tables.

C.2 A naive top-down construction

The method is to consider one by one every path from *paths* and to create the branch of nodes when the algorithm finds a uncreated node while following the path (Algorithm 1).

INPUT: *paths* the list of paths from the root to every leaf of the tree, *root* the root of the empty taxonomic tree.

OUTPUT: The corresponding taxonomic tree rooted at *root*.

Worst case time complexity: $\mathrm{O}(l_{path} \times n_{paths} \times n_{metadata} \times n_{samples})$ (to find the list of assignment corresponding to a given sample, it takes $\mathrm{O}(n_{samples} \times n_{metadata})$), that is, given the numbers of each variable in our test database and knowing $l_{paths} \leq n_{taxo-nodes}$ and $n_{paths} \simeq n_{taxo-nodes}$, $\mathrm{O}(n^2_{taxo-nodes})$. It takes approximately 2 hours on our computer to construct the tree associated with the parsed *paths* list of length $5,565$ (the total number of nodes was $9,065$).

C.3 A naive bottom-up construction

The previous method can be used for any type of tree. However, taxonomic trees are special indeed. Their height is bounded by the number of ranks in the phylogeny, and there are much more leaves than internal nodes. The following algorithm tries to take advantage of these two characteristics (Algorithm 2): it considers the nodes in decreasing rank (in terms of taxonomic accuracy), then look for each of them their children, create the tree associated with the nodes, and iterates this process until all the nodes had been turned into trees. It eventually returns the tree rooted at *root*.

INPUT: *paths* the list of paths from the root to every leaf of the tree, *root* the root of the empty taxonomic tree, *ranks* the array of ranks in the phylogeny in order of decreasing taxonomic precision (S, then G, then F, O, C, P, ...), and

Algorithm 1 The naive top-down construction

```
for path in $paths$ do
  $currNode \leftarrow root$
  $currLineage \leftarrow [(root.name, root.rank)]$
  $currChildren \leftarrow root.children$
  $currPath \leftarrow paths.tail()$
  while currPath is non-empty do
    $name, rank \leftarrow currPath.head()$
    $nextNode \leftarrow$ currNode's child which name, rank are $name$, $rank$
    if $nextNode \neq NONE$ then
      $currNode \leftarrow nextNode$
      $currLineage \leftarrow append(currLineage, [(currNode.name, currNode.rank)])$
      $currChildren \leftarrow currNode.children$
    else
      $sampleList \leftarrow$ the list of assignments associated with (name,rank)
      Create the node $node$ associated with name,rank and add it to the list
      of children of currNode
      $currNode \leftarrow node$
      $currLineage \leftarrow append(currLineage, [(currNode.name, currNode.rank)])$
      while currPath is non-empty do
        $name, rank \leftarrow currPath.head()$
        $sampleList \leftarrow$ the list of assignments associated with (name,rank)
        Create the node $node$ associated with name,rank and add it to the
        list of children of currNode
        $currNode \leftarrow node$
        $currLineage \leftarrow append(currLineage, [(currNode.name, currNode.rank)])$
      end while
    end if
  end while
end for
return $root$
```

nodesList the list of nodes in the taxonomic tree.

OUTPUT: The corresponding taxonomic tree rooted at *root*.

Algorithm 2 The naive bottom-up construction

```
$currConstructedTrees \leftarrow$ []
$numberRanks \leftarrow$ ranks.length()
for i in  1, ..., numberRanks  do
  $currNodesToProcess \leftarrow$ the list of nodes having rank ranks[i]
  //look at each node in nodesList, push it to currNodesToProcess iff its rank equals ranks[i]
  $potentialChildrenTrees \leftarrow$ currConstructedTrees
  for node in currNodesToProcess do
    $currSampleList \leftarrow$ the list of assignments associated with node node
    $currLineage \leftarrow$ the phylogeny of the node node
    $currChildrenNameRank \leftarrow$ the list of (name,rank) pairs of node's children
    $currChildren \leftarrow$ []
    $currNotChildren \leftarrow$ []
    while potentialChildrenTrees is non-empty do
      $currTree \leftarrow$ potentialChildrenTrees.head()
      if currTree belongs to currChildrenNameRank then
        $currChildren \leftarrow$ append(currChildren,currTree)
      else
        $currNotChildren \leftarrow$ append(currNotChildren,currTree)
      end if
      $currConstructedTrees \leftarrow$ currNotChildren
      $tree \leftarrow$ create the tree associated with node
      $currConstructedTrees \leftarrow$ append(currConstructedTrees,tree)
    end while
  end for
end for
return  currConstructedTrees.head()
```

Worst case time complexity: $O(n_{taxo-nodes} \times (n_{samples} \times n_{metadata} + n_{paths} \times l_{path} + n_{paths}^2 \times l_{paths})) = O(n_{taxo-nodes}^4)$ using the numbers of each variable in our test database and the previous hypotheses. Please note that the number of leaves bounds the width/maximum degree of the taxonomic tree. The number of ranks can be bounded by a constant integer (for instance, 10). For $5,565$ leaves and $9,065$ (internal and leaf) nodes, constructing the taxonomic tree took approximately 10 hours (!) on our computer.

C.4 A less naive bottom-up construction

The previous bottom-up algorithm has got a really bad time complexity, for it must recompute each time the lineage and the lists of assignments. This following method fixes this issue, by precomputing the lineage and the other phylogenetic relations (Algorithm 3) and then using it to get the whole taxonomic tree (Algorithm 4).

A FEW DEFINITIONS:

- A BROTHERHOOD is a list of nodes having the same father in the taxonomic tree. In case of a taxonomic tree, every node of the brotherhood has got the very same rank.
- HASHBROTHERLIST is a list such as *hashBrotherList*[i] = (n,m) means that the (i+1)th node in the list of nodes sorted by ranks belongs to the (m+1)th brotherhood of rank number (n+1) (in order of decreasing rank).
- HASHFATHERLIST is a list such as *hashFatherList*[i] = (n,m) means that the (i+1)th node is the father of the nodes of the (m+1)th brotherhood of rank number (n+1) (in order of decreasing rank). Unlike in *hashBrotherList*, *hashFatherList*[i] may be set to NONE.

C.4.1 Pre-processing

INPUT: *paths* the list of paths from the root to every leaf of the tree, *root* the root of the empty taxonomic tree, *nodesList* the list of all nodes present in the taxonomic tree, *ranks* the array of ranks in the phylogeny in order of decreasing taxonomic precision (S, then G, then F, O, C, P, ...).

OUTPUT: *sortedNodesList* list of nodes sorted by rank, *pathsList* the list of paths such as *pathsList*[i] is the path from root to *sortedNodesList*[i], *samplesList* the list of lists of assignments such as *samplesList*[i] is the list associated with node *sortedNodesList*[i], *brotherhoodsList* the list of brotherhoods in the tree, *hashBrotherList*, *hashFatherList*.

Worst case time complexity:
$O(n_{taxo-nodes} log(n_{taxo-nodes}) + (n_{paths} \times l_{path} + n_{samples} \times n_{metadata}) \times n_{taxo-nodes} + n_{taxo-nodes}^2)$
$= O(n_{taxo-nodes}^3)$ using the previous hypotheses. This pre-processing is faster than the first algorithm, because we have done a really rough analysis of the worst case: it takes less than 30 seconds to get a result.

C.4.2 Final algorithm

Using the preprocess of (Algorithm 3), then for every rank r from S to R, we get all the node identifiers of rank r.

Algorithm 3 The less naive bottom-up construction (pre-processing)

```
$sortedNodesList \leftarrow$ nodes in $nodesList$ sorted by decreasing rank
$pathsList \leftarrow$ paths from root to every node in $nodesList$
$samplesList \leftarrow$ lists of assignments for every node in $nodesList$
$brotherhoodsList \leftarrow$ []
$hashBrotherList \leftarrow$ an array of size $|nodesList|$ initialized with NONE
$hashFatherList \leftarrow$ an array of size $|nodesList|$ initialized with NONE
$currNodeIdent \leftarrow 0$ //index of current node considered in sortedNodesList
$currN \leftarrow -1$
for rank in ranks do
  $currN \leftarrow currN + 1$
  brotherhoodsOfThisRankList$\leftarrow$ []
  $currM \leftarrow -1$
  while currNodeIdent $<$ $|nodesList|$ and pathsList[$currNodeIdent$] and sortedNodesList[$currNodeIdent$].rank = rank do
    $currFather \leftarrow$ last node of path pathsList[currNodeIdent] //looking for the father of current brotherhood
    $currFatherIdent \leftarrow$ NONE
    for i in {1, 2, ... $|nodesList|$ } do
      if sortedNodesList[$i$] = currFather then
        $currFatherIdent \leftarrow$ i
      end if
    end for
    $currM \leftarrow currM + 1$
    currBrotherhoodOfThisRank $\leftarrow$ [currFatherIdent, currNodeIdent]
    hashBrotherList[currNodeIdent] = (currN,currM)
    $currNodeIdent \leftarrow currNodeIdent + 1$
    hashFatherList[currFatherIdent] $\leftarrow$ (currN,currM)
    if currNodeIdent $<$ pathsList.length and pathsList[$currNodeIdent$] exists then
      $father \leftarrow$ last node of pathsList[currNodeIdent]
    end if
    while while father = currFather and currNodeIdent $<$ $|nodesList|$ do
      currBrotherhoodOfThisRank $\leftarrow$ append(currBrotherhoodOfThisRank,-currNodeIdent)
      if currNodeIdent $<$ $|nodesList|$ then
        hashBrotherList[currNodeIdent] $\leftarrow$ (currN,currM)
      end if
      $currNodeIdent \leftarrow currNodeIdent + 1$
      if currNodeIdent $<$ $|pathsList|$ and pathsList[$currNodeIdent$] exists then
        $father \leftarrow$ last node of pathsList[currNodeIdent] //because of root
      end if
      brotherhoodsOfThisRank $\leftarrow$ append(brotherhoodsOfThisRank,-currBrotherhoodOfThisRank)
    end while
    if brotherhoodsOfThisRank is non-empty then
      brotherhoodsList $\leftarrow$ append(brotherhoodsList,brotherhoodsOfThisRank)
    end if
  end while
end for
return sortedNodesList,pathsList,samplesList,brotherhoodsList,
hashBrotherList,hashFatherList
```

Then for every node identifier n of the previously computed list, if n has no child (by checking out the value *hashFatherList*), we construct its tree and store it in *constructedTrees* (*constructedTrees*[i] being the tree rooted at *sortedNodesList*[i]).

Else we get n's children identifiers through *hashFatherList*, we get children's trees (since r is from S to R), then we construct the tree associated with n, and store it in *constructedTrees*. Then we return the tree rooted at *root*.

INPUT: *paths* the list of paths from the root to every leaf, *sortedNodesList* list of nodes sorted by rank, *pathsList* the list of paths such as *pathsList*[i] is the path from root to *sortedNodesList*[i], *samplesList* the list of lists of assignments such as *samplesList*[i] is the list associated with node *sortedNodesList*[i], *brotherhoodsList* the list of brotherhoods in the tree, *hashBrotherList*, *hashFatherList*, *ranks* the list of ranks.

OUTPUT: The corresponding taxonomic tree rooted at *root*.

<u>Worst case time complexity</u>: O(n_{paths}^2). The number of leaves $n_{leaves} = n_{paths}$ bounds the number of same-ranked nodes, that is the width of the tree. It takes less than 30 seconds on our computer to return the result (preprocessing included) for $5,565$ leaves and $9,065$ nodes, and less than 0.2 seconds for 600 nodes and 402 leaves, which seems consistent with the theoretical complexity found.

Algorithm 4 The less naive bottom-up construction

```
for r in ranks do
    $sLs \leftarrow sortedNodesList$.copy()
    $ident \leftarrow 0$
    $sameRankedNodes \leftarrow []$
    $name, rank \leftarrow$ sLs.head()
    while sLs is non-empty and rank = r do
        $sameRankedNodes \leftarrow$ append(sameRankedNodes,(name,rank,ident))
        $ident \leftarrow ident + 1$
        if sLs is non-empty then
            $name, rank \leftarrow$ sLs.head()
        end if
    end while
    $sLs \leftarrow append(sLs, (name, rank))$
    if r = R then
        $sameRankedNodes \leftarrow$ append(sameRankedNodes,(name,rank))
    end if
    while sameRankedNodes is non-empty do
        $name, rank, idt \leftarrow$ sameRankedNodes.head()
        if hashFatherList[idt] is not NONE then
            n,m = hashFatherList[idt]
            $children \leftarrow brotherhoodesList$[n][m].tail()
            $childrenTrees \leftarrow$ the list of trees rooted at each child of children
            //stored in constructedTrees
            $constructedTrees[idt] \leftarrow$ tree rooted at (name,rank)
        else
            $constructedTrees[idt] \leftarrow$ tree rooted at (name,rank)
        end if
    end while
end for
constructedTrees[|nodesList|].children ← list of trees of root's children
return constructedTrees[|nodesList|]
```

Appendix D

Multi-dimensional lists

D.1 Goal

When one considers only one metadatum, classes are sets of samples having the same value of metadatum. When there are more than one metadatum, things go harder (see the classes example in the chapter about the second method).

To implement a easy access and modification of the classes, I have chosen to implement a class of multidimensional lists, because matrices in Python do not allow other objects than integers.

D.2 Implementation and complexity

A Multi-Dimensional List (MDL) contains the attributes *mdList*, which is the real multi-dimensional list, and *shape*, which the list of maximum values for each dimension.

See *MultiDimList* module for code source. Let n_{dim} be the number of dimensions of the MDL, and dim_i the maximum value of the ith dimension of the MDL.

Table D.1: Worst case time complexity of different operations on MDL, (*) The *deepcopy* operation is linear, but the constant is great, (**) Returns first element and the list of other elements, (***) Maps function over the elements of MDL, then returns them as a list

OPERATION	COMPLEXITY	REMARKS
Creation	O($n_{dim} \times \sum_{i=1}^{n_{dim}} dim_i^2$)	
Get length	O(n_{dim})	Product of its dimensions
Copy	O($n_{dim} \times \sum_{i=1}^{n_{dim}} dim_i^2$)	
Access to a case	O(n_{dim})	
Modify a case	O(n_{dim}^2)	Destructive operation (*)
Enumerate (**)	O($n_{dim}^2 \times \sum_{i=1}^{n_{dim}} dim_i^2$)	
Member function	O(n_{dim}^3)	
Map a function (***)	O(n_{dim}^3)	

Appendix E

Summary (in French)

- Le contexte général

  Les arbres taxonomiques (ou *phylogénétiques*) représentent les liens de parenté entre les espèces en fonction de leur histoire évolutive. Ils sont précieux notamment dans un domaine particulier de la bioinformatique qui est la *métagénomique.* La *métagénomique* est l'étude du contenu génétique d'échantillons d'organismes généralement prélevés dans un milieu naturel. En général, ces organismes ne peuvent être cultivés en laboratoire, car les conditions requises à leur survie ne sont pas aujourd'hui réalisables artificiellement, par exemple certaines bactéries dans l'intestin ou vivant dans des écosystèmes marins fragiles. Après obtention des échantillons, le but le plus souvent est de chercher quels organismes s'y trouvent, en traitant par exemple l'ADN ou l'ARN retrouvé dans les échantillons, et en le comparant à des données de référence, et en s'aidant d'un arbre taxonomique. La *métagénomique* permet donc d'étudier des espèces ou des écosystèmes inconnus, et a de nombreuses applications en médecine et en biologie, puisqu'elle peut aider à comprendre le fonctionnement de l'intestin humain par exemple, ou l'équilibre d'un milieu marin fragile.

- Le problème étudié

  Il existe d'ores et déjà des algorithmes qui permettent de préciser l'assignation d'une partie de l'information génétique dans un échantillon à un organisme particulier, des algorithmes de construction d'arbres taxonomiques à partir de données génétiques (algorithme de Neighbor Joining, ...), ou des mesures sur les arbres qui permettent de quantifier leur pertinence phylogénétique (où l'algorithme de Zhang-Shasha peut être utilisé, ...). Cependant, il manque des outils pour analyser les données métagénomiques, et trouver des correspondances entre les espèces retrouvées dans les échantillons, et les informations non-taxonomiques (par exemple la température d'un milieu en biologie, ou l'âge d'un patient en médecine).

Le but de ce stage est d'améliorer l'analyse des données métagénomiques dans le domaine médical, en essayant de clusteriser les patients en fonction à la fois de leurs populations bactériennes, et des données cliniques. Ainsi, le stage s'aide d'une base de données construite par l'étude des ADN bactériens issus d'intestins de patients atteints de mucovisidose. Il peut être intéressant de savoir par exemple quelle est l'influence d'un traitement sur la qualité de vie du patient, et quelles bactéries dominent selon si le traitement est administré oralement ou en intraveineuse. Ces questions sont traitées pour le moment par recoupement d'analyses statistiques, en utilisant le test de WILCOXON ou de MCNEMAR. *In fine*, l'objectif est de pouvoir clusteriser les patients de manière semi-automatique pour pouvoir par exemple améliorer le traitement ou donner un diagnostic.

- LA CONTRIBUTION PROPOSÉE

  Trois approches ont été proposées, toutes répondant à des problèmes légèrement différents.

  1. Une première méthode, qui utilise une approche statistique avec différentes mesures sur des populations bactériennes, détermine une distance entre les échantillons pour renvoyer les paires de groupes d'échantillons (formés avec les valeurs d'une donnée clinique) les moins proches, ce qui permet de savoir si la donnée clinique choisie est discriminante sur les populations bactériennes des patients. Cependant, pour être pertinente, cette méthode nécessite l'utilisation de mesures robustes sur les populations.
  2. La deuxième approche tente de comparer les arbres taxonomiques des patients avant assignation des noeuds. Elle utilise un algorithme d'*apprentissage supervisé*, qui essaie de trouver quels sont les espèces de bactéries qui discriminent le mieux les échantillons en fonction des données cliniques. Mais de fortes hypothèses mathématiques sont nécessaires pour le fonctionnement de cette méthode.
  3. La troisième approche, utilisant de l'*apprentissage non supervisé*, clusterise les échantillons dans des classes de données cliniques, avec deux distances qui ne dépendent que des populations bactériennes des échantillons. Ces clusters sont ensuite évalués en fonction des valeurs réelles des données cliniques de chaque patient. Cette méthode évite d'avoir à fournir *a priori* des hypothèses sur la présence d'une espèce dans une population, ou sur les données cliniques.

- Les arguments en faveur de sa validité

  Les tests sur la base de données précédente avec la première méthode ont donné des résultats similaires à l'analyse statistique. Les premiers tests avec les deux méthodes suivantes semblent être aussi cohérents. Malheureusement, la pauvreté des études sur cette base de données ne permet pas de conclure. Pour pouvoir valider véritablement les résultats biologiques obtenus, il faudrait pouvoir avoir accès à plusieurs dizaines d'ensembles d'échantillons et itérer les calculs précédents pour pouvoir les confirmer.

  Les résultats numériques peuvent dépendre aussi du traitement préalable des informations : normalisation des données numériques, centrées réduites, existence ou non d'une procédure standardisée pour la récupération des données...). De plus, l'interprétation des données obtenus ne peut pas se passer de l'avis du praticien.

- Le bilan et les perspectives

  L'utilisation d'algorithmes de Machine Learning en *métagénomique* n'est pas nouvelle, mais est ici exploitée pour faire le lien entre des données cliniques, dont le traitement relève en général plus de la statistique, et les arbres, qui relèvent de l'algorithmique. D'une part, pour les arbres étiquetés et non ordonnés (dont les arbres taxonomiques sont une sous-catégorie), beaucoup de problèmes de comparaison sont NP-complets. Ce qui permet d'échapper à ce problème de complexité dans notre problème est que la comparaison pertinente ici soit essentiellement celle des ensembles de feuilles. D'autre part, il serait intéressant dans le futur de pouvoir tester ces algorithmes sur d'autres bases de données.