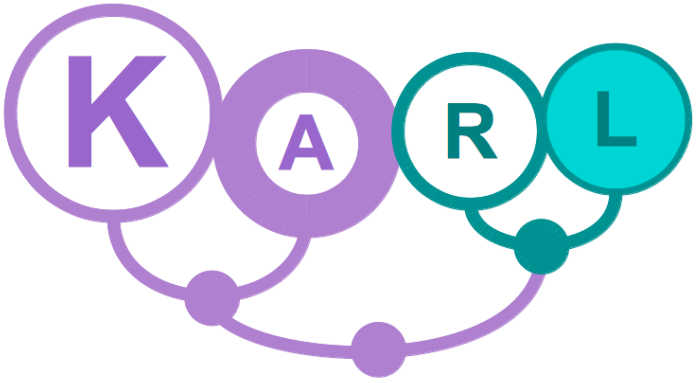
KARL v1.0: an R package for proKaryotes tAxonomy pRediction using metaboLic signatures coded in their genomes

Gregorio IRAOLA

March 16, 2016



FULL USER MANUAL

Contents

1 Overview

The acronym **KARL** comes from pro**K**aryotic t**A**xonomy p**R**ediction using genome-encoded metabo**L**ic features and is on behalf of **Carl Linnaeus** (the parent of taxonomy) and **Carl Woese** (the parent of modern taxonomy of prokaryotes). The KARL package was designed with the intention of exploiting the enormous amount of genomic data that is being constantly produced since some years ago for an increasing diversity of prokaryotes. The main aim of KARL is to provide a stable framework to fill the gap between the exploration of genome-encoded metabolic signatures (that can be thought as omics-fashioned chemotaxonomic traits) and the organism taxonomic position. Throughout the functionalities provided inside KARL, the user can predict the taxonomic position (from domain to genus) of a certain genome by using the metabolic features coded inside it, as well as comparing the distribution of enzymes and pathways along all the available prokaryotic taxa. For the sake of simplicity and usefulness KARL has been divided in three different modules, called **Explorer**, **Predictor** and **Updater**. Accordingly, the functionalities of each module are summarized in this **Full User Manual** where theoretical and practical details are provided.

Developer:

Gregorio IRAOLA[1,2] - giraola@pasteur.edu.uy

[1]Unidad de Bioinformática, Institut Pasteur Montevideo, Uruguay. [2]Sección Genética Evolutiva, Facultad de Ciencias, Universidad de la República, Uruguay.

Citation:

Iraola G, Naya H (2030) Wedding higher taxonomic ranks with metabolic signatures coded in prokaryotic genomes. *Charoná.* **33:**11-12.

2 Installation

The first version of KARL (v1.0) is currently available at **GitHub** repository (`http://github.com/giraola/KARL`). It is planned to be available soon at **CRAN** (`https://cran.r-project.org`) and **Bioconductor** (`https://www.bioconductor.org`).

```
# INSTALL KARL

library(devtools)
install_github('/giraola/KARL')
```

2.1 Platforms

Currently KARL works on **Mac** (tested on Snow Leopard, Lion and Mountain Lion) and **Linux** (tested on Fedora and Ubuntu). A Windows version is planned for the future but is not being developed so far.

2.2 Dependencies

KARL depends on a set of other R packages: seqinr [1], RWeka [2], rJava, gplots, pathview [3], png, caret, VennDiagram [4], doParallel, KEGGREST, rgl, segmented [5] and RCurl.

2.3 External software

KARL uses a handful of state-of-the-art softwares for performing tasks that are not possible (at least as easy as desired) from inside R or by using other R packages. These softwares **do not need to be installed by the user**, as they come with KARL as compiled executables.

List of external softwares used by KARL:

- **BLAST -** Basic Local Alignment Search Tool [6]. KARL uses blastn, blastp, makeblastdb and blastdbcmd.
- **Glimmer -** Gene finding using interpolated Markov Models [7].
- **Prodigal -** Prokaryotic dynamic programming gene-finding algorithm [8].
- **SPAdes -** Genome assembler from short reads [9].
- **Rnammer -** Annotation of ribosomal RNA genes [10].
- **Silva -** Comprehensive ribosomal RNA database [11].

3 Explorer

In this first version of KARL, a total amount of 33,236 bacterial and archaeal genomes representing 55 different phyla, 67 classes, 163 orders, 328 families and 1,480 genera were recovered from the PATRIC database [12]. For each genome, the full taxonomic description from domain to genus was accessed using the NCBI Taxonomy database and the presence/absence (P/A) for 1,328 different enzymes was determined using PATRIC Pathway annotation tables. All this information was stored in a matrix called **alldata** (Fig. 1), which is the main object where most KARL operations are performed.

Functions implemented in the **Explorer** module allows the user to compare the metabolic features between groups of genomes belonging to different taxa at the same taxonomic rank, for example, the distribution of enzymes among two different genera. KARL Explorer provides automatic visualization devices using classic tools like Venn diagrams, heatmaps and more sophisticated ones allowing to map enzymes frequencies over selected metabolic pathways.

List of KARL Explorer functions:

- **taxChild() -** Returns a vector of children taxa for a given parent taxon.
- **prcomTax() -** Plots 2D or 3D PCAs, coloring selected taxa.
- **enzymesFreq() -** Plots enzymes pairwise distribution in selected taxa.
- **getEnzymes() -** Returns a list of shared and exclusive enzymes for two selected taxa at the same taxonomic rank.
- **enzymesVenn() -** Gets getEnzymes output and returns a Venn diagram.
- **comp1heat() -** Plots a heatmap using enzymes P/A of a given taxon.
- **comp2heat() -** Plots a heatmap using enzymes P/A of two given taxa.
- **comp3heat() -** Plots a heatmap using enzymes P/A of three taxa. Ideally, one of them should be an outgroup.
- **findPath() -** Maps a KEGG pathway reference number given a keyword.
- **compPath() -** Uses KEGG Pathway layouts to compare enzymes frequencies among two given taxa.
- **onePath() -** Gives the frequency of all enzymes for a certain pathway.
- **consPath() -** Gives the conservation of metabolic pathways among selected taxa calculating average enzyme conservation.
- **cumsumPath() -** Plots the conservation of metabolic pathways.
- **link2path() -** Identifies significant pathways that difference a pair of taxa at a certain rank.

```
> alldata[1:10,c(1:7,90:95)]
         id                                                genome
1  1000561.3                        Pseudomonas aeruginosa AES-1R
2  1000562.3                             Streptococcus phocae C-4
3  1000565.3                   Methyloversatilis universalis FAM5
4  1000568.3                             Megasphaera sp. UPII 199-6
5  1000569.4                             Megasphaera sp. UPII 135-E
6  1000570.3               Streptococcus anginosus SK52 = DSM 20563
7  1000588.3                    Streptococcus mitis bv. 2 str. SK95
8  1000589.3 Streptococcus dysgalactiae subsp. equisimilis SK1249
9  1000590.6                  Staphylococcus epidermidis 14.1.R1.SE
10 1000935.3               Anaplasma marginale str. Florida Relapse
          phylum               class           order            family
1  Proteobacteria Gammaproteobacteria Pseudomonadales  Pseudomonadaceae
2      Firmicutes             Bacilli Lactobacillales  Streptococcaceae
3  Proteobacteria  Betaproteobacteria   Rhodocyclales    Rhodocyclaceae
4      Firmicutes       Negativicutes Selenomonadales   Veillonellaceae
5      Firmicutes       Negativicutes Selenomonadales   Veillonellaceae
6      Firmicutes             Bacilli Lactobacillales  Streptococcaceae
7      Firmicutes             Bacilli Lactobacillales  Streptococcaceae
8      Firmicutes             Bacilli Lactobacillales  Streptococcaceae
9      Firmicutes             Bacilli      Bacillales Staphylococcaceae
10 Proteobacteria Alphaproteobacteria   Rickettsiales   Anaplasmataceae
              genus 1.1.1.81 1.1.1.83 1.1.1.85 1.1.1.86 1.1.1.87 1.1.1.9
1        Pseudomonas        1        0        1        1        0       0
2      Streptococcus        0        0        0        0        0       0
3  Methyloversatilis        1        0        1        1        0       0
4        Megasphaera        0        0        1        1        0       0
5        Megasphaera        0        0        0        1        0       0
6      Streptococcus        0        0        0        0        0       0
7      Streptococcus        0        0        1        1        0       0
8      Streptococcus        0        0        0        0        0       0
9     Staphylococcus        1        0        1        1        0       0
10         Anaplasma        0        0        0        1        0       0
>
```

Figure 1: Subset of alldata matrix showing taxonomic information and enzymes presence/absence (1/0) for each genome.

3.1 Getting children taxa

The **alldata** matrix is composed of more than 33,000 genomes (rows), the **taxChild()** function allows to list (Fig. 3) and optionally count (Fig. 2) the children taxa of a certain taxon at any taxonomic rank.

```
# Use taxChild()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX takes a string indicating a parent taxon.
# COUNTING takes a logical indicating whether to count each taxa.

taxChild(rank='class',
     tax='Actinobacteria',
     counting=FALSE)
```

```
> taxChild(level='family',tax='Neisseriaceae',counting=F)
 [1] "Neisseria"       "Morococcus"      "Eikenella"       "Alysiella"
 [5] "Conchiformibius" "Stenoxybacter"   "Vitreoscilla"    "Kingella"
 [9] "Snodgrassella"   "Simonsiella"
>
```

Figure 2: List of genera immediately below the family Neisseriaceae. Counts disabled.

```
taxChild(rank='family',
      tax='Neisseriaceae',
      counting=TRUE)
```

```
> taxChild(level='class',tax='Actinobacteria',counting=T)

     Acidothermales        Jiangellales      Kineosporiales      Nakamurellales
                  1                   1                   2                   2
   Catenulisporales  Actinopolysporales      Glycomycetales  Geodermatophilales
                  3                   4                   6                   9
         Frankiales Streptosporangiales     Actinomycetales   Pseudonocardiales
                 18                  45                  67                  81
  Micromonosporales Propionibacteriales   Bifidobacteriales       Micrococcales
                125                 161                 211                 247
   Streptomycetales   Corynebacteriales
                472                2817
>
```

Figure 3: List of orders contained immediately below the class Actinobacteria and their absolute counts.

3.2 Plot Principal Components Analysis (PCA)

Principal Components Analysis is a commonly used method to reduce high-dimentional data. The **prcompTax()** function plots and maps colors to selected taxa in 2 or 3 selected PCA components (Figure 4-5).

```
# Use prcompTax()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAXA is a vector indicating taxa names comprised in RANK.
# COLORS is a vector of same length as TAXA indicating colors.
# AXES takes a vector of length 2 or 3 indicating the PCs to plot.
# LEGPOS is the position for the legend.

prcompTax(rank='genus',
      taxa=c('Streptomyces','Lactobacillus'),
      axes=c(1,2),colors=c('red','darkgreen'),legpos='topright')
```

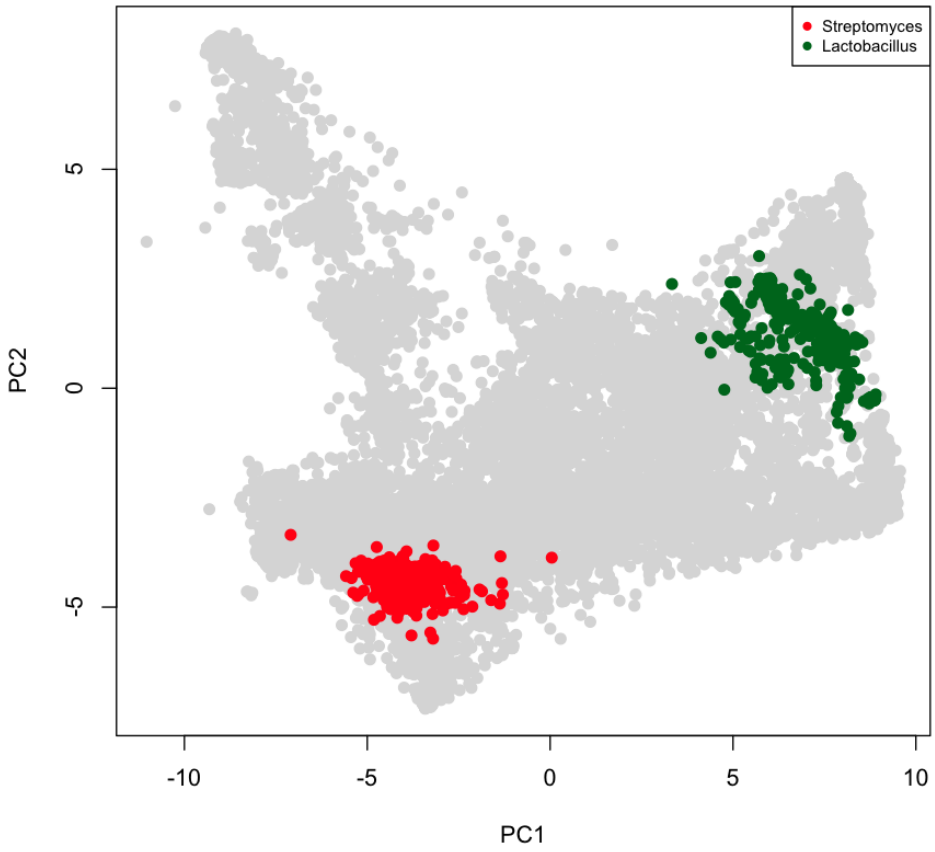


Figure 4: 2D plot showing the first 2 axis in the PCA. Members of gram-positive genera *Lactobacillus* and *Streptomyces* are colored, respectively.

```
# Use prcompTax()

prcompTax(rank='phylum',
      taxa=c('Proteobacteria','Firmicutes'),
      axes=c(1,2,3),colors=c('darkblue','red'))
```

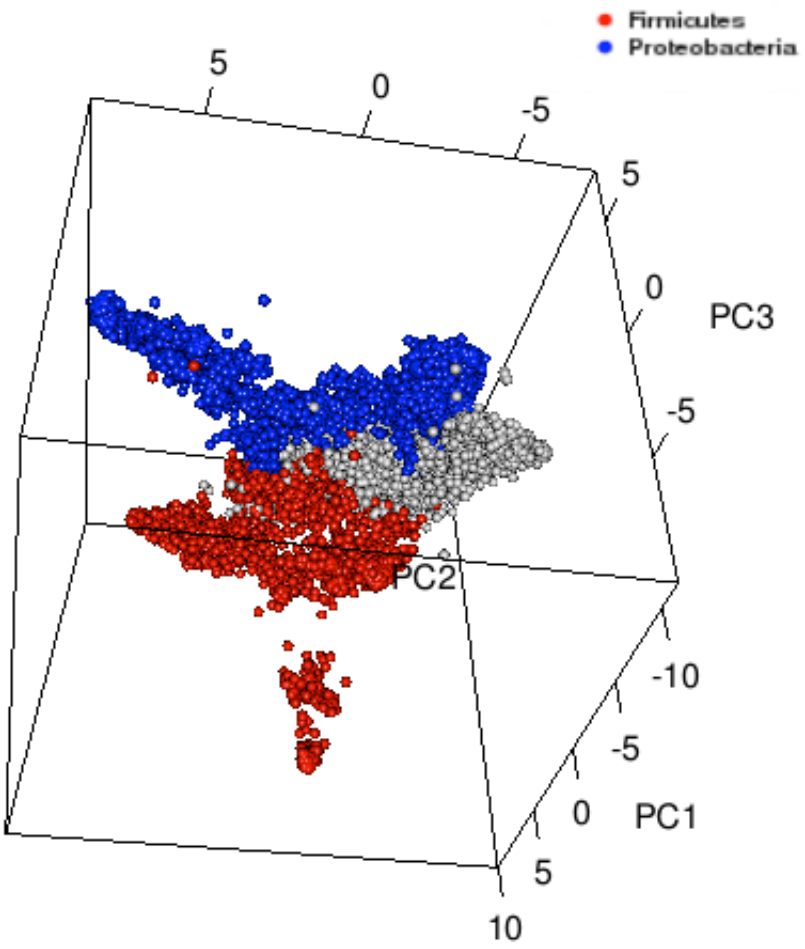


Figure 5: 3D plot showing the first 3 axis in the PCA. The most abundant phyla (Firmicutes and Proteobacteria) are colored, respectively.

3.3 Enzymes pairwise distribution plots

When interested in detecting particular molecular functions that characterize a certain group of organisms, a valid approach consist in identifying which features are present in that group and absent in another groups or viceversa, at different cut-offs. Using the **enzymesFreq()** function, the user can generate highly informative plots showing the frequency distribution of all tested enzymes in a pair of taxa that belong to the same taxonomic rank (Figure 6):

```
# Use enzymesFreq()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX1 takes a string indicating a taxon belonging to a desired rank.
# TAX2 takes a string indicating another taxon of the same rank.

enzymesFreq(rank='class',
      tax1='Actinobacteria',
      tax2='Gammaproteobacteria')
```

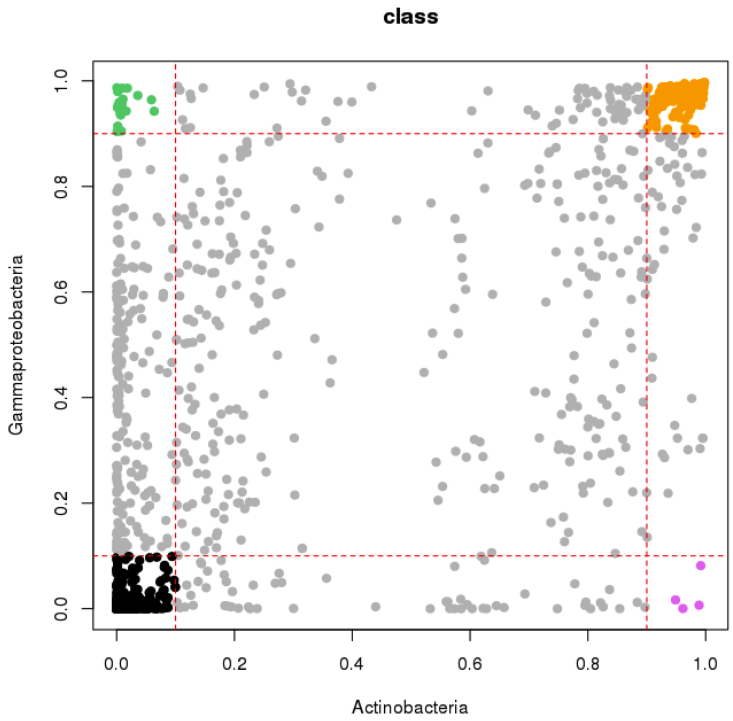


Figure 6: enzymesFreq() function output showing the frequency distribution of enzymes between classes Actinobacteria and Gammaproteobacteria.

In Figure 6, black dots represent enzymes with frequency lower than 0.1 in both tested taxa (hardly always absent in both), while orange dots represent enzymes with a frequency higher than 0.9 in both taxa (hardly always present in both). Green dots correspond to enzymes whose frequency is lower than 0.1 for TAX1 (Actinobacteria) and higher than 0.9 for TAX2 (Gammaproteobacteria), hence almost exclusive for Gammaproteobacteria in this example. Purple dots correspond to those enzymes whose frequency is lower than 0.1 for TAX2 and higher than 0.9 for TAX1, hence exclusive for Actinobacteria in this example.

3.4 Getting informative enzymes lists

The graphical overview provided by **enzymesFreq()** is useful to identify general trends among taxa, however more in-depth analysis require to know which enzymes are taxon-specific. For this purpose, **getEnzymes()** outputs a list with taxon-specific enzymes (if exist), shared enzymes and absent enzymes at desired lower and upper frequency cut-offs (Figure 7).

```
# Use getEnzymes()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX1 takes a string indicating a taxon belonging to a desired rank.
# TAX2 takes a string indicating another taxon of the same rank.
# DWTAX1 is the lower frequency cut-off for TAX1 (default 0.1).
# DWTAX2 is the lower frequency cut-off for TAX2 (default 0.1).
# UPTAX1 is the upper frequency cut-off for TAX1 (default 0.9).
# UPTAX2 is the upper frequency cut-off for TAX2 (default 0.9).

enzymes<-getEnzymes(
   rank='class',
   tax1='Actinobacteria',
   tax2='Gammaproteobacteria')
```

```
$inActinobacteria
2.7.1.63 3.1.3.41 6.1.1.24  6.4.1.3
     643      782     1260     1325

$inGammaproteobacteria
1.1.1.262 1.1.1.284 1.1.1.290  1.3.1.12   1.3.3.3   1.8.1.2 2.3.1.129 2.4.1.182
       40        47        49       248       265       339       411       476
 2.5.1.55  2.6.99.2 2.7.1.107 2.7.1.130  2.7.7.38   3.1.1.2  3.1.2.12  3.1.3.27
      540       587       593       598       695       742       760       779
 3.1.3.45   3.1.3.7  3.6.1.41   5.3.3.8  6.1.1.18   6.3.2.3
      783       786       965      1218      1253      1296

>
```

Figure 7: getEnzymes() function outputs a list showing enzymes present on each tested taxa.

3.5 Drawing Venn diagrams with enzymes lists

For global visualization, the output list from **getEnzymes()** can be automatically turned into a Venn diagram. The **enzymesVenn()** function takes the output list and plots the corresponding Venn diagram (Figure 8).

```
# Use enzymesVenn()
# ELIST is a list object output from getEnzymes() function.

enzymesVenn(elist=enzymes)
```

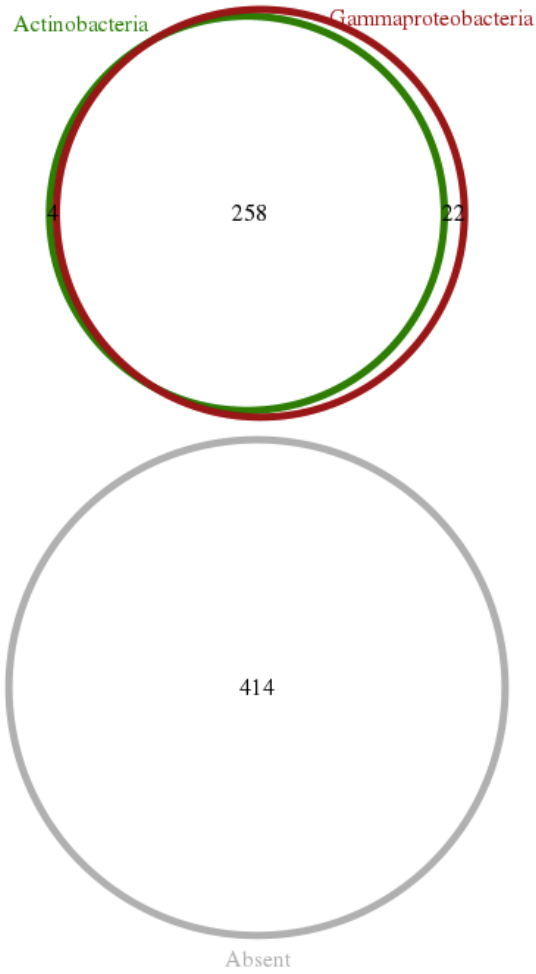


Figure 8: enzymesVenn() function plots a Venn diagram using the list of enzymes provided by **getEnzymes()** function.

3.6 Heatmap: one taxon

For initial whole comparison of enzymes presence/absence patterns it can be useful to visualize the relationships of multiple organisms by plotting a heatmap with **comp1heat()** (Figure 9-10).

```
# Use comp1heat()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX takes a string indicating the selected taxon.
# LROWS is a logical indicating if plotting genome names or not.
# REDUCE is a logical indicating if collapsing identical patterns.

comp1heat(
   rank='genus',
   tax='Salmonella',
   lrows=FALSE,
   reduce=TRUE)
```

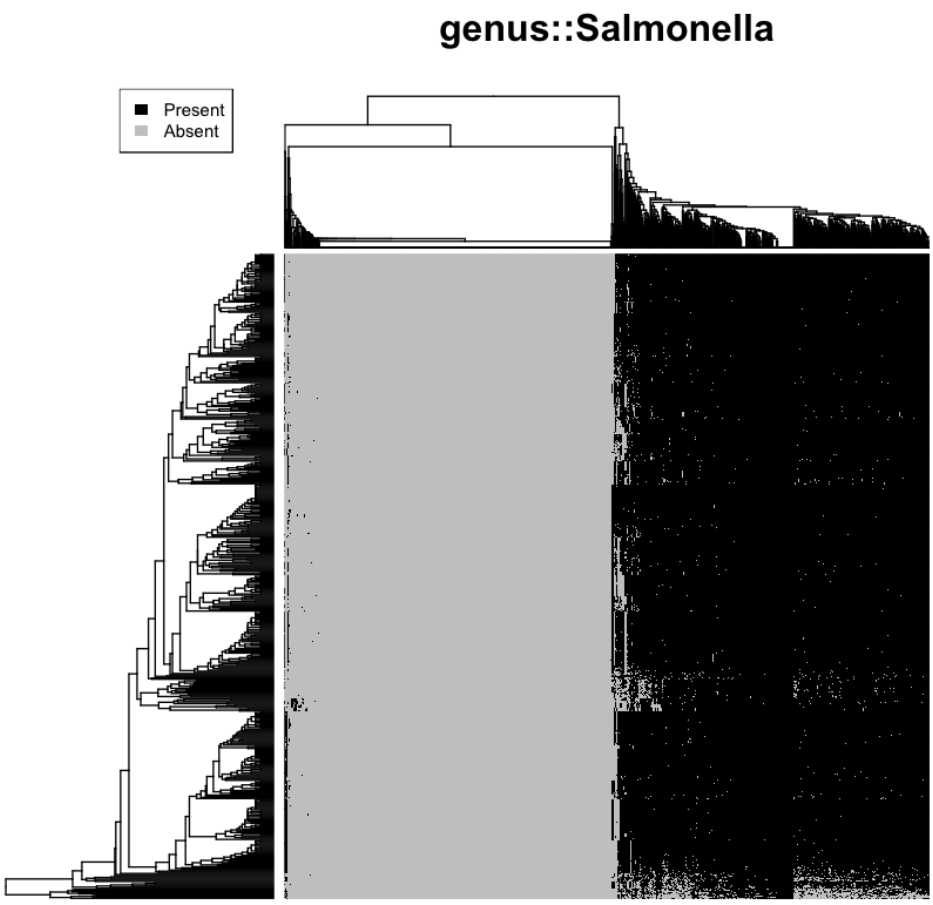


Figure 9: Heatmap showing the hierarchical clustering of *Salmonella* members (rows) based on their enzymes presence/absence patterns (columns).

```
comp1heat(
   rank='genus',
   tax='Actinobacillus',
   lrows=TRUE,
   reduce=FALSE)
```

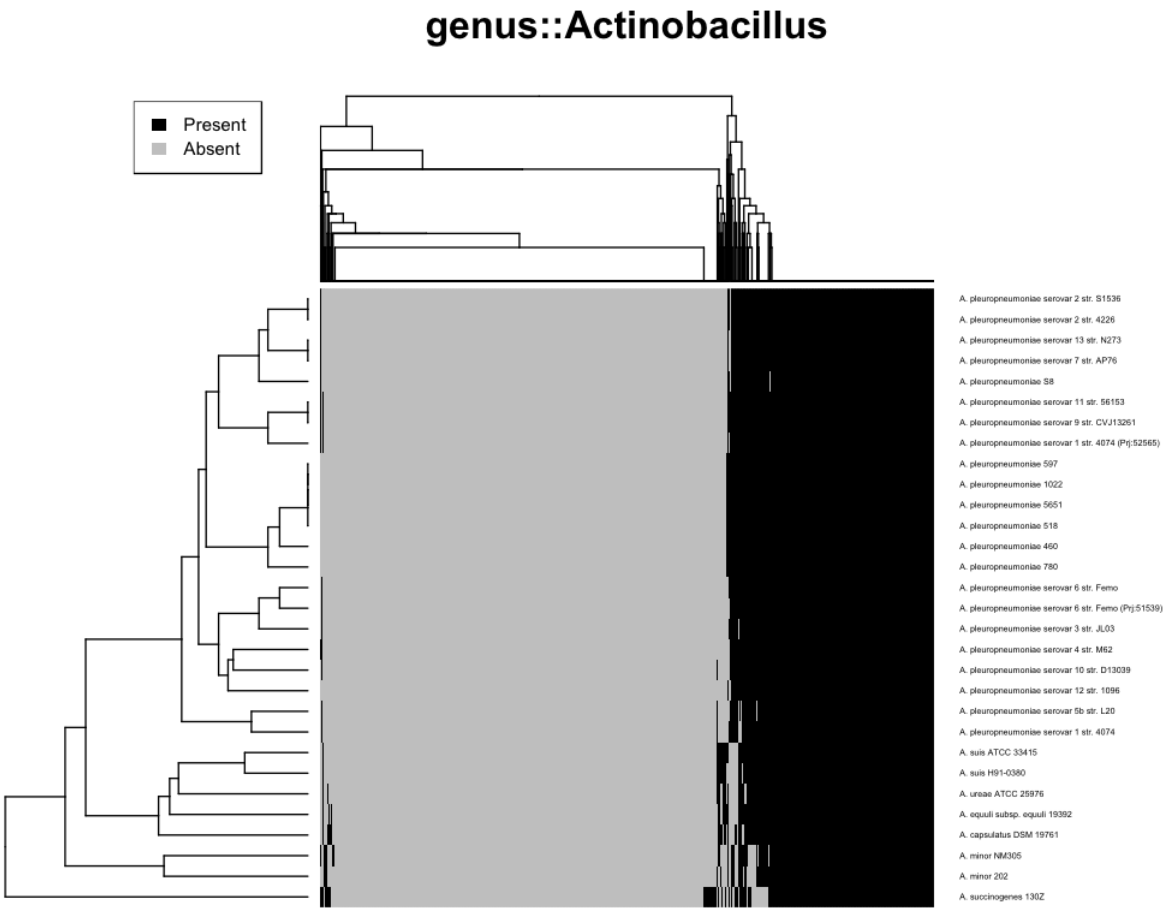


Figure 10: Heatmap of genus *Actinobacillus* where some differences can be seen between *A. pleuropneumoniae* and the rest of the species.

3.7 Heatmap: comparing two taxa

For getting a general overview of metabolic differences among taxa one can select two desired groups among a certain taxonomic rank and plot them with **comp2heat()** (Figure 11,12).

```
# Use comp2heat()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX1 takes a string indicating one selected taxon.
# TAX2 takes a string indicating the other selected taxon.
# REDUCE is a logical indicating if collapsing identical organisms.

comp2heat(
  rank='genus', tax1='Actinobacillus', tax2='Bartonella', reduce=FALSE)
```

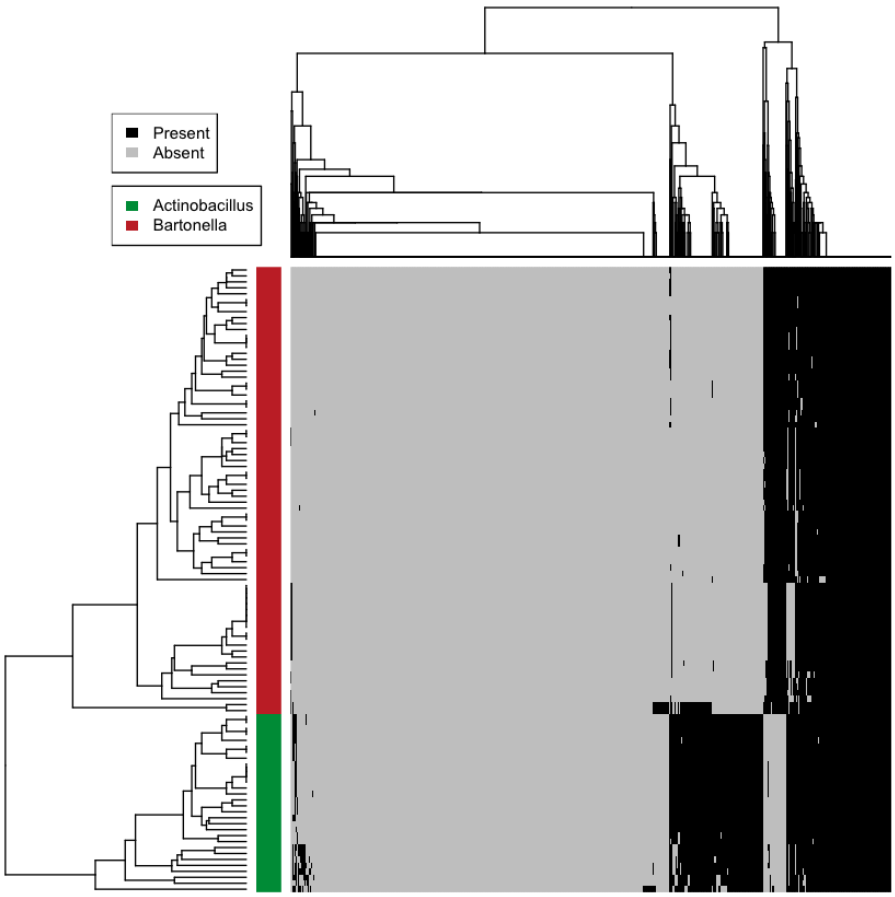


Figure 11: Heatmap of genera *Actinobacillus* and *Bartonella* showing a perfect clustering. At bottom there is a block of distinctive enzymes for *Actinobacillus*.

```
comp2heat(
   rank='genus',
   tax1='Borrelia',
   tax2='Spirochaeta',
   reduce=FALSE)
```

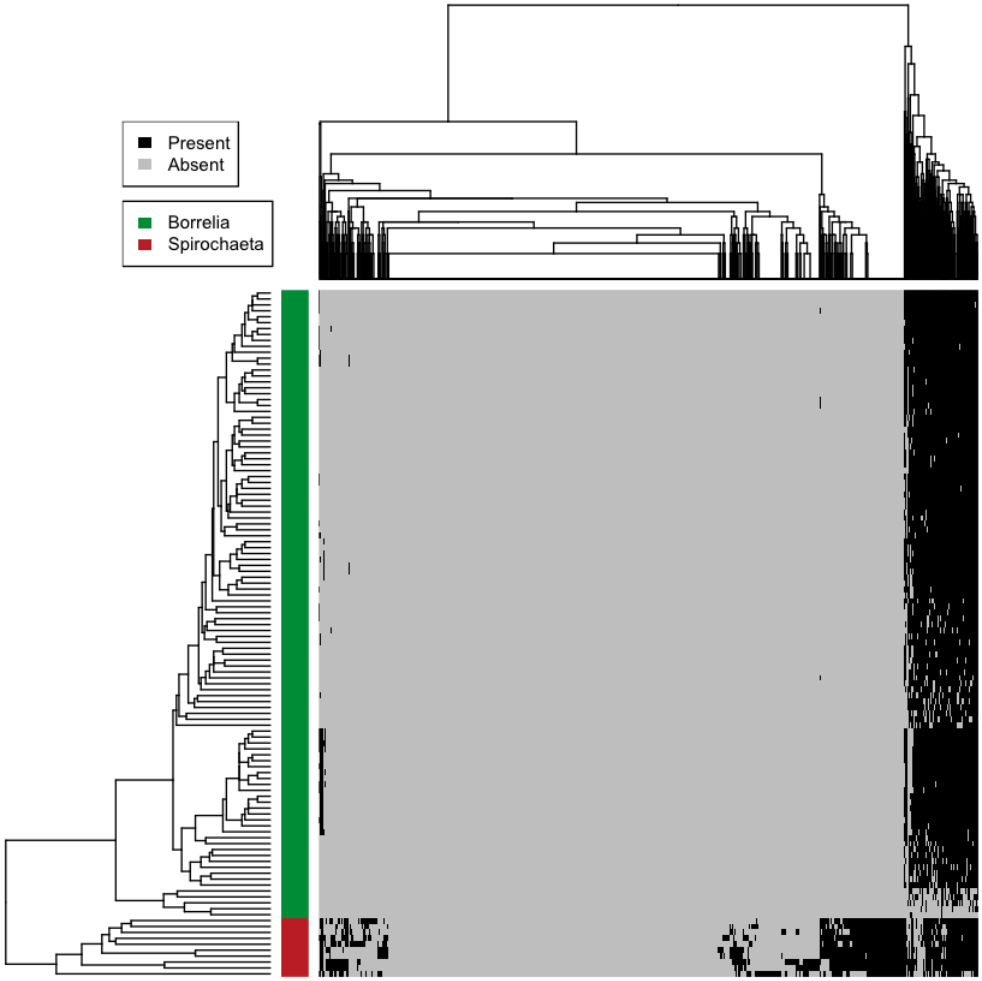


Figure 12: Heatmap of two phylogenetically related genera: *Borrelia* and *Spirochaeta*. Despite belonging to the same family (Spirochaetaceae), the discriminatory power of enzymes presence/absence is enough to totally distinguish each other.

3.8 Heatmap: compare two taxa + outgroup

The use of outgroups is usual to enrich comparisons. Using **comp3heat()** the user can set a third taxa that should works as an outgroup of two other taxa, at the same taxonomic rank (Figure 13). Note that **comp3heat()** uses hierarchical clustering that should not be interpreted as a phylogenetic tree, hence the decision of including an outgroup must be based on previous phylogenetic information.

```
# Use comp3heat()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX1 takes a string indicating one selected taxon.
# TAX2 takes a string indicating the other selected taxon.
# OUTGROUP takes a string indicating the outgroup taxon.
# REDUCE is a logical indicating if collapsing identical organisms.

comp3heat(
   rank='genus', tax1='Xylella', tax2='Dyella',
   outgroup='Nevskia', reduce=TRUE)
```

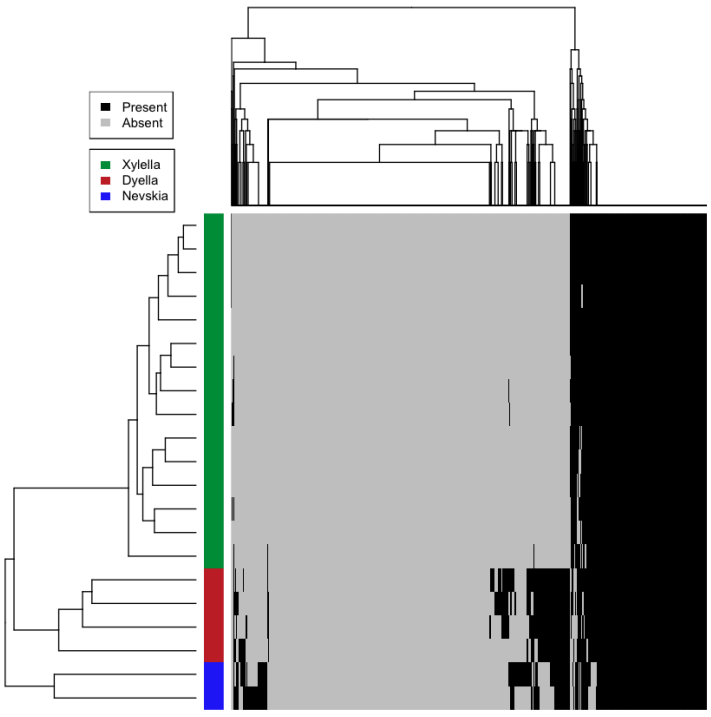


Figure 13: Heatmap of *Dyella* and *Xyllela* genera (family Xanthomonadaceae). The *Nevskia* genus, that belongs to the distinct Sinobacteraceae family was set as outgroup.

3.9 Rendering selected KEGG pathways

A full biological interpretation of enzymes presence/absence patterns can be achieved by comparing them in the context of complete metabolic pathways. Taking advantage from KEGG Pathways database [13], the **compPath()** function allows the user to choose a desired metabolic pathway representation from KEGG, and automatically visualize the frequency (as a color gradient) of each enzyme belonging to that pathway in two desired taxa from a certain taxonomic rank. This is particularly useful for exploring functional metabolic differences across taxa.

3.9.1 Identifying pathways using keywords

KEGG pathways are referenced using a pathway-specific reference number. The common user, however, may not know *a priori* what is the reference number for a pathway of interest. So, by using **findPath()** one can input a certain keyword that identifies a particular pathway and return the specific reference number for it (Figure 14).

```
# Use findPath()
# KEYWORD is a string indicating a keyword for a pathway

findPath(keyword='peptidoglycan')
```

```
> findPath('peptidoglycan')
                path:map00550
"Peptidoglycan biosynthesis"
>
```

Figure 14: Using the keyword “peptidoglycan" the **findPath()** function outputs the reference number (path:map00550) for the Peptidoglycan biosynthesis pathway.

3.9.2 Plotting selected pathways

The **compPath()** function creates a PNG file containing the representation of a selected pathway and adds enzymes presence/absence information for two selected taxa, represented by a color gradient (Figure 15).

```
# Use compPath()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX1 takes a string indicating one selected taxon.
# TAX2 takes a string indicating the other selected taxon.
# PID is a KEGG pathway reference number.

compPath(rank='class',tax1='Betaproteobacteria',
     tax2='Mollicutes',pid='00550')
```

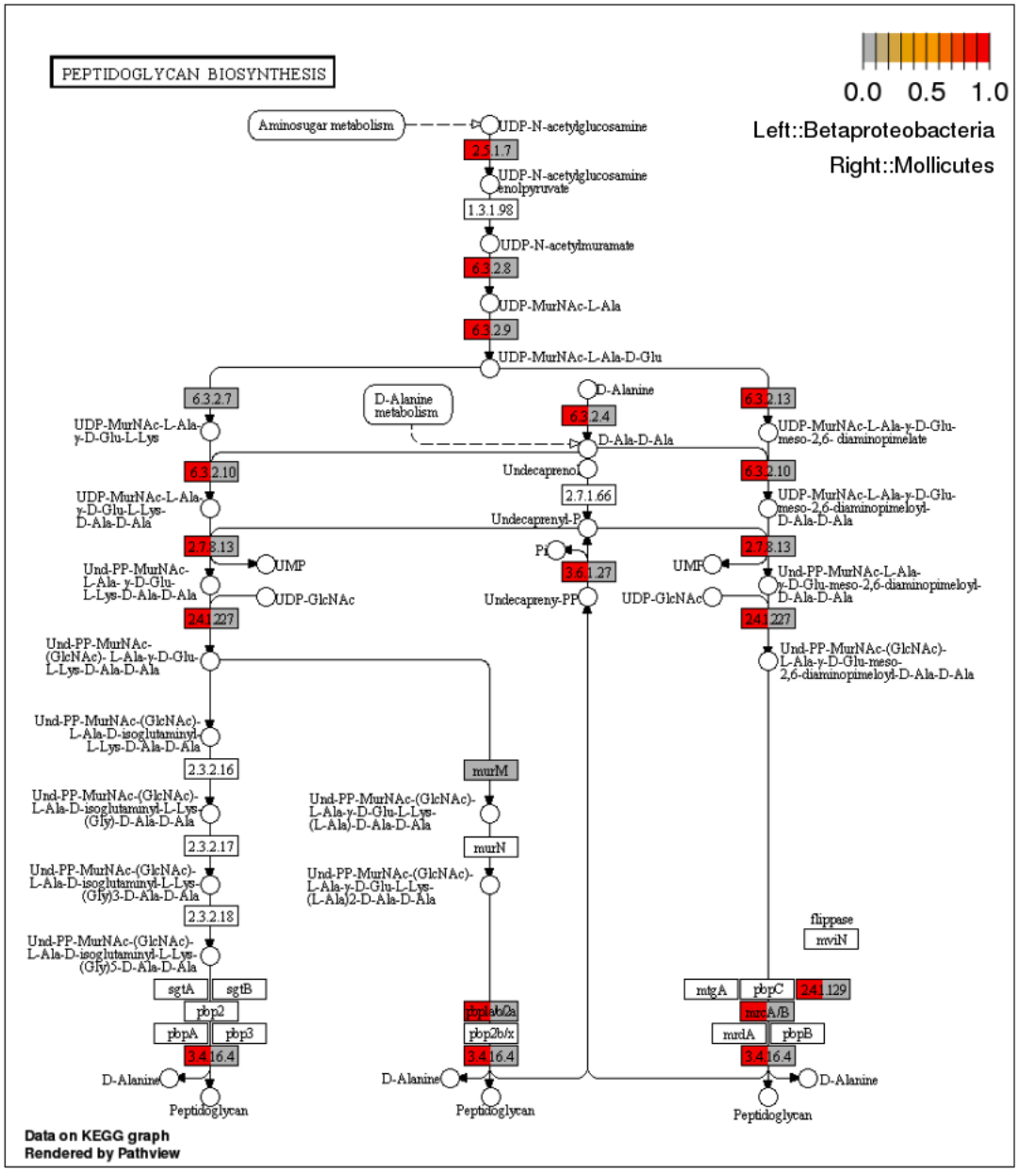


Figure 15: KEGG pathway for peptidoglycan biosynthesis showing the great conservation of these enzymes in class Betaproteobacteria (left) and the absence of all enzymes in class Mollicutes (right), which lack peptidoglycan cell walls.

3.10 Identifying significant KEGG pathways

Beyond selecting particularly interesting pathways by hand, the user can automatically infer which pathways significantly discriminate a pair of taxa by mapping enzyme distributions into reference metabolic pathways. For each pathway with at least one enzyme present in any taxon, a test of proportions is applied to determine deviations from the null hypothesis (same proportion in both taxa) by calculating the Pearson's χ-squared statistic (Fig. 16,17) [14].

```
# Use link2path()
# ELIST is a list object output from getEnzymes() function.
# CONF is a p-value for confidence cut-off.
# COL1 color for the first taxon.
# COL2 color for the second taxon.

enzymes<-getEnzymes(tax1='Helicobacteraceae',
   tax2='Enterococcaceae', rank='family')

sig.paths<-link2path(elist=enzymes, conf=0.01,
   col1='blue', col2='firebrick')
```

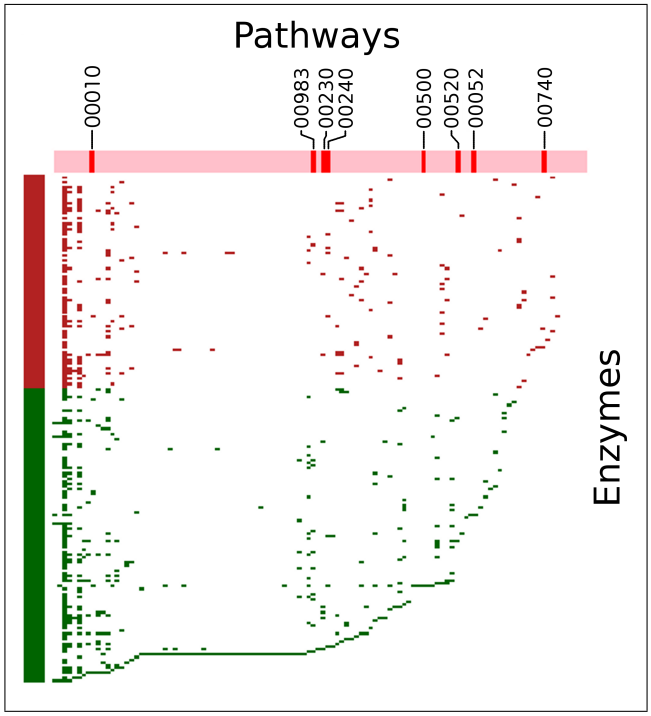


Figure 16: Output from **link2path()** mapping enzymes to pathways in both families. Significant pathways (p <0.01) are highlighted in red.

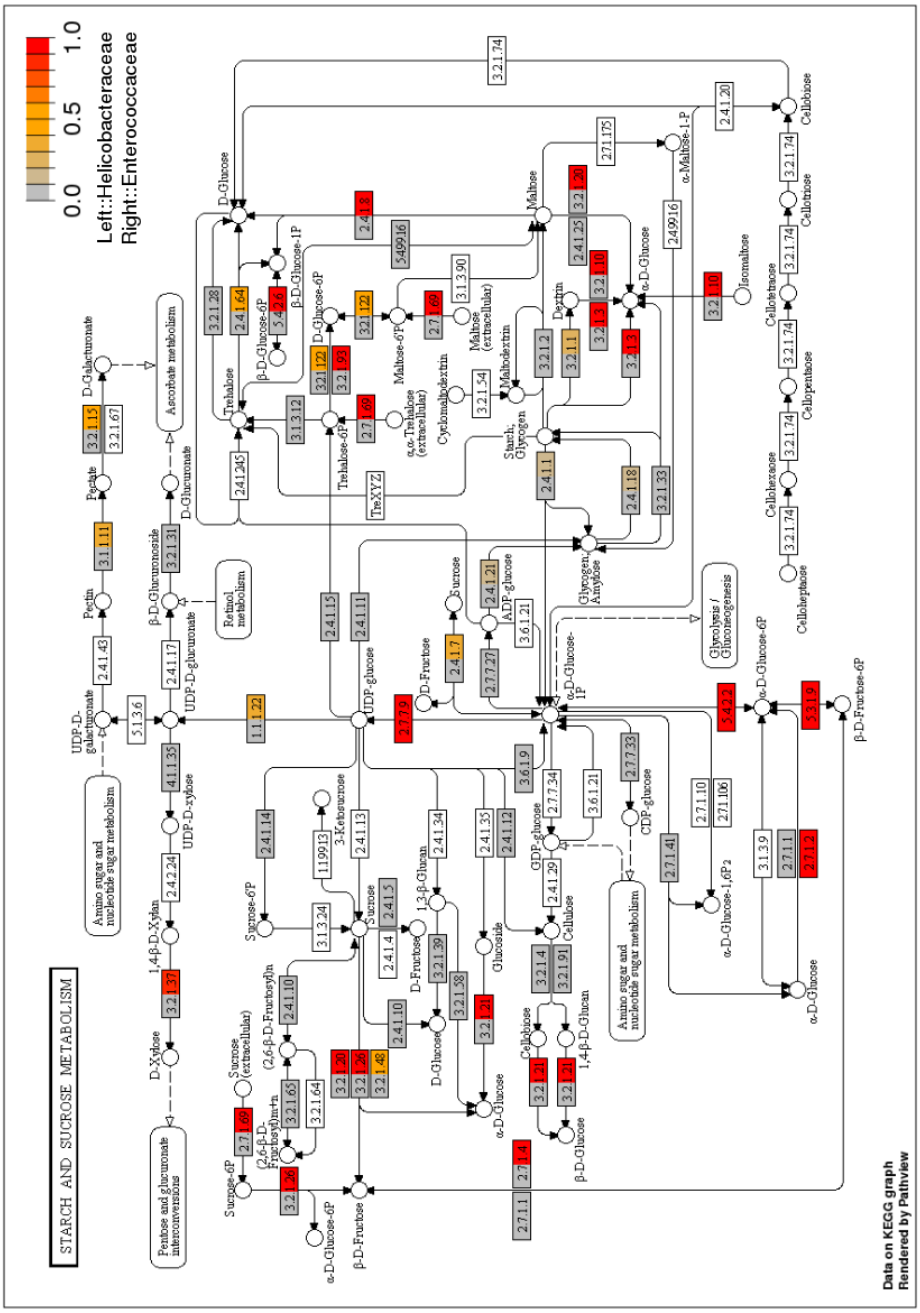


Figure 17: Output from **compPath()** for starch and sucrose metabolism (00500), previously identified as significant by **link2path()** for families Helicobacteraceae and Enterococcaceae.

3.11 Metabolic pathways conservation

The current version of **alldata** accounts for 1,328 enzymes that can be allocated in 144 different metabolic pathways. In order to explore the conservation of these pathways along the members of different taxonomic groups, KARL provides three functions: **onePath()**, **consPath()** and **cumsumPath()**.

1. In first place, **onePath()** allows the user to retrieve the frequency of each enzyme belonging to a certain pathway in a set of selected taxa (Figures 18-19).

```
# Use onePath()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAXA is a vector of names indicating taxa belonging to the RANK.
# PID is a pathway identifier, like '00901'.

tone<-c('Gammaproteobacteria', 'Mollicutes', 'Spirochaetia',
    'Actinobacteria')

pone<-onePath(rank='class', taxa=tone, pid='00550')
```

```
> pone
                  2.3.2.- 2.3.2.10   2.4.1.- 2.4.1.129 2.4.1.227   2.5.1.7
Gammaproteobacteria       0        0 0.9721546 0.9941768 0.9861302 0.9908947
Mollicutes                0        0 0.1753555 0.0000000 0.0000000 0.0000000
Spirochaetia              0        0 0.6948177 0.9923225 0.9654511 0.9731286
Actinobacteria            0        0 0.9820847 0.9974407 0.9767334 0.9925547
                   2.7.8.13     3.4.-.-    3.4.16.4   3.6.1.27  6.3.2.10
Gammaproteobacteria 0.9907888 0.590259397 0.989835892 0.94494442 0.9885654
Mollicutes          0.0000000 0.009478673 0.009478673 0.03791469 0.0000000
Spirochaetia        0.9750480 0.078694818 0.364683301 0.93090211 0.9712092
Actinobacteria      0.9937180 0.234062355 0.982550023 0.97417403 0.9920893
                     6.3.2.13   6.3.2.4     6.3.2.7   6.3.2.8   6.3.2.9
Gammaproteobacteria 0.988353626 0.9758602 0.000000000 0.9881419 0.9885654
Mollicutes          0.004739336 0.0000000 0.000000000 0.0000000 0.0000000
Spirochaetia        0.644913628 0.9443378 0.000000000 0.9692898 0.9673704
Actinobacteria      0.884132154 0.9723127 0.005583993 0.9851094 0.9874360
```

Figure 18: Output from **onePath()** indicating the frequency of all enzymes belonging to the Peptidoglycan biosynthesis pathway (00550).

```
# Barplot using onePath() output

barplot(sort(pone[,9]), ylim=c(0,1), border=F,
   cex.names=.8, ylab='Frequency', main='E.C. 3.4.16.4')
```

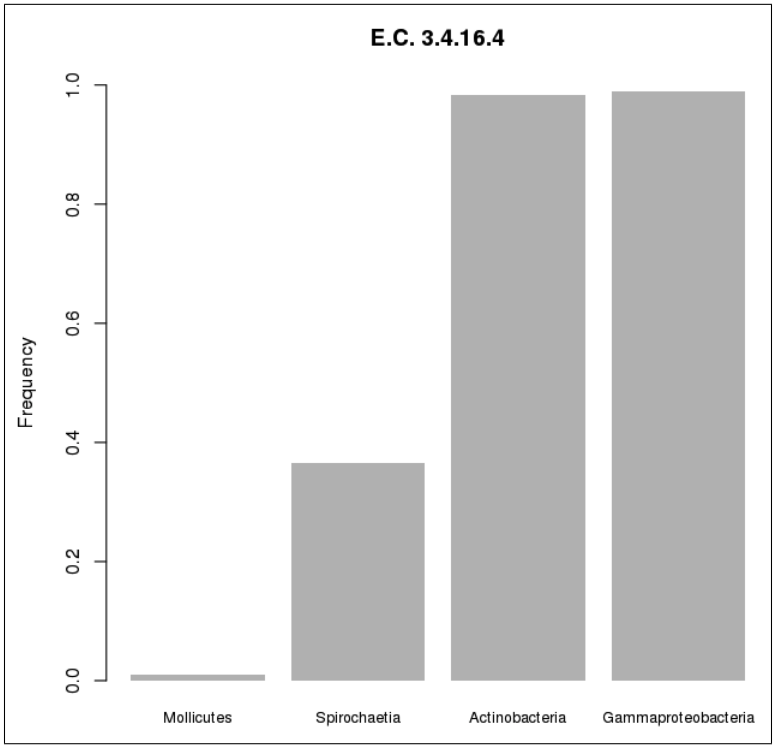


Figure 19: Barplot showing the frequency for DD-peptidase (E.C. 3.4.16.4) in 4 selected classes: Mollicutes, Spirochaetia, Actinobacteria and Gammaproteobacteria.

2. Second, by using **consPath()** the user can obtain and plot the completeness of each metabolic pathway (C_p) based on the frequency of each enzyme (Figure 20). Equation 1 defines C_p as the frequency mean for the enzymes of a certain pathway for a certain taxon:

$$C_p = \frac{\sum_{i=1}^{n} f_i}{n} \tag{1}$$

```
# Use consPath()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAXA is a vector of names indicating taxa belonging to the RANK.

taxco<-c('Streptomyces','Acinetobacter','Mycoplasma',
     'Escherichia','Borrelia','Rickettsia',
     'Buchnera','Pseudomonas','Haemophilus')

coco<-consPath(rank='genus',taxa=taxco)
```

```
> coco
                ec00564   ec00670    ec00720   ec00230   ec00983    ec00130
Streptomyces  0.41379310 0.7555556 0.63203704 0.7038815 0.5207778 0.44527778
Acinetobacter 0.54434921 0.8023844 0.49912836 0.6323259 0.2526424 0.45365400
Mycoplasma    0.19416802 0.2730705 0.09134974 0.2039374 0.2453020 0.06319911
Escherichia   0.62397494 0.8042374 0.64093017 0.8041113 0.6328471 0.72322747
Borrelia      0.22248876 0.2315217 0.14975845 0.2470725 0.2334783 0.08913043
Rickettsia    0.33421751 0.3528846 0.37051282 0.1764103 0.0500000 0.31730769
Buchnera      0.03256705 0.5000000 0.13888889 0.2800000 0.1333333 0.11805556
Pseudomonas   0.58879687 0.8068231 0.52493935 0.7221543 0.4706114 0.41229985
Haemophilus   0.52475685 0.6883013 0.36680912 0.5540171 0.4262821 0.46367521
```

Figure 20: Output from **consPath()** function showing the C_p of each pathway (n = 144) in 9 selected genera.

```
# Plot consPath() results

heatmap(coco)
```

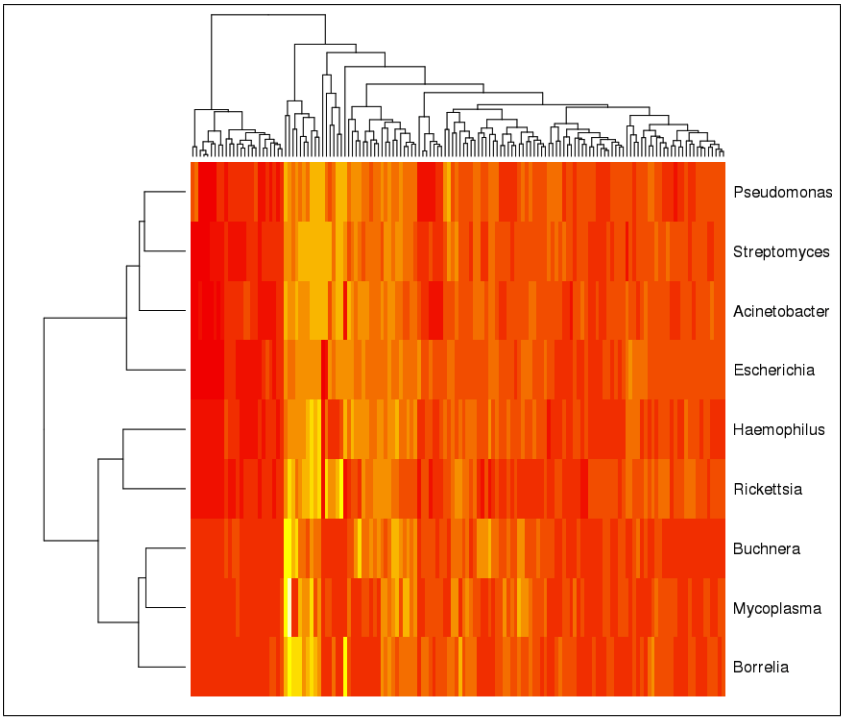


Figure 21: A subset of genomes are intracellular and have suffered genome reduction; these cluster together in agreement with the hypothesis of ecological coherence of higher taxa.

Figure 21 shows a heatmap created with **consPath()** output. In this example, some genera that have suffered reductive genome processes (*Buchnera*, *Haemophilus*, *Rickettsia*, *Mycoplasma*, *Borrelia*) were compared with a set of genera that have not suffered this evolutionary process (*Pseudomonas*, *Streptomyces*, *Acinetobacter*, *Escherichia*). Despite of being phylogenetically distant, those genera with reduced genomes cluster together evidencing their ecological coherence.

3. Third, **cumsumPath()** allows to plots the percentage of organisms belonging to a certain taxon carrying different fractions of enzymes that belong a certain pathway, in a cumulative way. In other words, given all the organisms belonging a certain taxon, the percentage of organisms carrying different fractions of enzymes (from 0 to 1) are plotted in a cumulative curve.

```
# Use cumsumPath().
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAXA is a vector of names indicating taxa belonging to the RANK.
# PID is a pathway identifier like '00550'.
# ECOL is a vector of color names of same length than TAXA.

taxcum<-c('Actinobacteria', 'Tenericutes', 'Firmicutes',
    'Proteobacteria', 'Bacteroidetes')

taxcol<-c('red', 'green', 'blue', 'violet', 'orange')

cumsumPath(rank='phylum',
   taxa=taxcum,
   ecol=taxcol,
   pid='00550')
```

Figure 22 shows how **cumsumPath()** allows to discriminate bacterial phyla that present almost all enzymes needed for Peptidoglycan biosynthesis in almost all their genomes, from Tenericutes that lacks almost all these enzymes due to their inability to synthesize a peptideglycan cell wall. For Actinobacteria, Firmicutes, Proteobacteria and Bacteroidetes, the cumulative curve is quickly saturated since most of their genomes are rich in peptidoglycan biosynthetic enzymes.

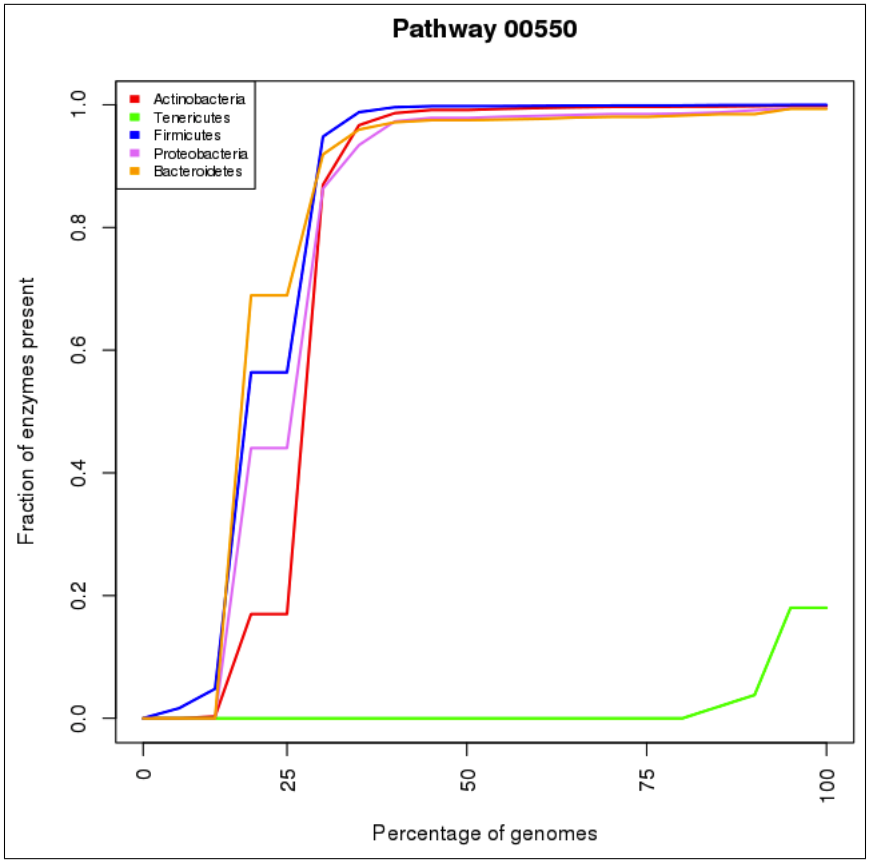


Figure 22: Heatmap showing pathways conservation index for genera that have suffered genome reduction.

4 Predictor

KARL Predictor module implements all functions needed to perform taxonomic classifications of organisms based on the presence/absence patterns of enzymes coded in their genomes. Beyond a set of specific functions that implement machine-learning algorithms for assigning genomes to taxonomic units, KARL Predictor offers a set of pre-processing functions that allow the user to classify a certain genome just inputing HTS (High-Throughput Sequencing) reads, unannotated contigs/scaffolds or completed genomes. Furthermore, a 16S-based classification is also available.

List of KARL Predictor functions:

- **assemblR() -** is a wrapper function for SPAdes *de novo* assembler.
- **findR() -** implements Prodigal and Glimmer for protein encoding genes (PEGs) discovery.
- **assignR() -** builds a presence/absence vector based on BLAST searches against local enzyme-specific databases.
- **classifiR() -** predicts the membership of a genome to taxonomic units.
- **buildSVM() -** builds a SVM classifier for a certain taxon.
- **evaluateSVM() -** evaluates a SVM classifier using repeated cross-validation.
- **dropSVM() -** eliminates uninformative enzymes for classifying a certain taxon based on frequencies and correlations.
- **rankSlope() -** eliminates uninformative enzymes for classifying a certain taxon based on Information Gain and Davies' test.
- **votingTaxa() -** calculates the euclidean distance for a certain organism against the rest in the PCA space.
- **iMAGO() -** calculates MAGO distance for re-classifying wrongly classified genomes.
- **predictOne() -** predicts the membership of genomes to a particular taxa.
- **predictFull() -** predicts the full lineage from domain to genus for a set of genomes.
- **identifiR() -** extracts the 16S gene from a genome and compares it with Silva database.

4.1 Building classification models

KARL models are based on Support Vector Machine (SVM) classification algorithms dependent on RWeka package [2]. Given the binary nature of data (enzymes presence/absence), these classifiers are trained using linear kernels and evaluated using cross-validation schemes. All models are built for classifying one particular taxon against the rest, for example, given the 55 distinct phyla comprised in **alldata**, there will be 55 models to classify each one against the rest. Classification models can be built by default with the whole set of 1,328 enzymes in the **alldata** object, however for desired taxa one can improve classification performance by applying some feature selection procedures.

4.1.1 Default models

The **buildSVM()** function implements **SMO()** from RWeka using a linear kernel to train and automatically evaluate (using an internal cross-validation scheme) a SVM classification model for a certain taxon (Figure 23).

```
# Use buildSVM()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX takes the name of desired taxon to build the classifier

cyanobacteriaSVM<-buildSVM(rank='phylum',tax='Cyanobacteria')
```

```
=== 10 Fold Cross Validation ===

=== Summary ===

Correctly Classified Instances       33153               99.9849 %
Incorrectly Classified Instances         5                0.0151 %
Kappa statistic                          0.9923
K&B Relative Info Score            3266412.28   %
K&B Information Score                 2628.5107 bits      0.0793 bits/instance
Class complexity | order 0            2661.8276 bits      0.0803 bits/instance
Class complexity | scheme             5370      bits      0.162  bits/instance
Complexity improvement     (Sf)      -2708.1724 bits     -0.0817 bits/instance
Mean absolute error                      0.0002
Root mean squared error                  0.0123
Relative absolute error                  0.7662 %
Root relative squared error             12.3895 %
Coverage of cases (0.95 level)          99.9849 %
Mean rel. region size (0.95 level)      50      %
Total Number of Instances            33158

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.012    1.000      1.000    1.000      0.992    0.994     1.000     FALSE
                 0.988    0.000    0.997      0.988    0.992      0.992    0.994     0.985     TRUE
Weighted Avg.    1.000    0.012    1.000      1.000    1.000      0.992    0.994     1.000

=== Confusion Matrix ===

     a     b   <-- classified as
 32828     1 |     a = FALSE
     4   325 |     b = TRUE
>
```

Figure 23: The content of **cyanobacteriaSVM**, which stores statistics for the internal evaluation of the model. Example for phylum Cyanobacteria.

Additionally, when running **buildSVM()**, two objects will be stored at the current working directory as binary R files recoverable with **load()**:

- **tax.model.Rdata -** Stores the model itself, which will be used to predict the membership of a certain genome to that particular taxon.
- **tax.eval.Rdata -** Stores the internal evaluation of the model, as described previously and showed in Figure 23.

Note that the **tax** suffix will be replaced by the value of "tax" parameter in **buildSVM()** function. In the example case, the files will be named **Cyanobacteria.eval.Rdata** and **Cyanobacteria.model.Rdata**.

4.1.2 Detecting misclassified instances

The default internal evaluation scheme of **buildSVM()** is based on the implementation of **evaluate_Weka_classifier()** from RWeka package, which performs a standard 10-fold cross-validation approach. However, one broad drawback for this implementation is the impossibility to easily inform which instances were wrongly classified as false positives or false negatives, respectively.

Considering that knowing exactly which genomes are systematically misclassified can provide important biological information about the reasons for that misclassification, KARL implements a novel function called **evaluateSVM()** that performs m-repeated n-fold cross-validation and stores the detailed results for each genome in a purposely created R class called **evalModel**.

```
# Use evaluateSVM()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX takes the name of desired taxon to perform evaluation.
# FOLDS is an integer indicating the number of folds in cross-validation.
# NPROC is an integer for the number of parallel processes.
# ITERS is an integer for the iterations of repeated cross-validation.

evalCyanobacteria.df.i3<-evaluateSVM( rank='phylum',
             tax='Cyanobacteria',
             folds=10, nproc=10, iters=3)
```

The resulting object **evalCyanobacteria.df.i3** belongs to class **evalModel** (Figure 24), which is composed by 5 slots consisting in 4 lists of length 2 and a confusion matrix:

- **perfectMatches -** Stores a list of genome identifiers that were consistently classified as true positives (TprT) or true negatives (FprF) in all m repetitions of n-fold cross-validations. These are highly supported matches.
- **variousMatches -** Stores a list of genome identifiers that were classified as true positives (TprT) or true negatives (FprF) in the at least *(m/2)+1* repetitions of n-fold cross-validations. These are supported matches with a certain degree of uncertainty.

- **perfectErrors -** Stores a list of genome identifiers that were consistently classified as false positives (FprT) or false negatives (TprF) in all m repetitions of n-fold cross-validations. These are highly supported errors.
- **variousErrors -** Stores a list of genome identifiers that were classified as false positives (FprT) or false negatives (TprF) in the at least *(m/2)+1* repetitions of n-fold cross-validations. These are supported errors with a certain degree of uncertainty.
- **confusionMatrix -** Stores a confusion matrix indicating the total number of genomes classified as TprT, FprF, TprF and FprT.

```
> evalCyanobacteria.df.i3@confusionMatrix
    prF prT
F 32828   1
T     4 325
>
> evalCyanobacteria.df.i3@perfectErrors
$FprT
[1] "1259808.3"

$TprF
[1] "1165095.3" "322866.3"  "569152.3"  "569154.3"

>
```

Figure 24: Two slots of **evalModel** class storing the evaluation performed with **evaluateSVM()** for phylum Cyanobacteria.

Note that when for a certain taxon there are less than 10 genomes, it is not possible to perform a 10-fold cross-validation since when splitting data in 10 chunks some of them will be just conformed by members of the negative class (the rest). When this happens, **evaluateSVM()** automatically detects the number of genomes for the analyzed taxon and performs a n-fold cross-validation if n is lower than 10 (Figure 25).

```
[1] "HALOVIVAX"
[1] "Not enough Halovivax members (2) to perform 10-fold cross-validation!"
[1] "3-repeated, 2-fold cross-validation started!"
```

Figure 25: Evaluation of genus *Halovivax*, which only has 2 instances.

4.1.3 Deciding on misclassified instances

The presence of misclassified instances both as false negatives or false positives may have different reasons. In first place, and more desirable, can be due

to organisms mislabelings or true taxonomic misplacements. In second place, can evidence classification errors inherent to the model construction or the lack of resolution power in presence/absence patterns for some taxa.

Those misclassified instances due to model inaccuracy could be rescued to certain extent using alternatives approaches. In this sense, KARL implements a function called **iMAGO()** that uses an Euclidean distance based on n-dimentional PCA coordinates (calculated by **votingTaxa()** function) to assign probabilities of membership of a certain misclassified genome to spatially neighboring taxa. A complete description of this methodology is herein presented step by step:

1. Given a certain misclassified genome, pairwise Euclidean distances against all the rest are calculated using a fixed number of PCA axes as:

$$D_{Euc} = \sqrt{\sum_{1}^{n}(q_i - p_i)^2} \tag{2}$$

Where q and p are the coordinates in a certain PCA axis for the target genome and each one of the rest, and n is the number of PCA axs.

```
# Use votingTaxa()
# IDS is a vector of identifiers for misclassified genomes.
# NPROC is an integer indicating the number of processors.
# PLIM is the number of PCA axes used in distance calculation.

load('evalActinobacteria.df.i3.Rdata')
actinoTprF<-evalActinobacteria.df.i3@perfectErrors$TprF

actino.vt7<-votingTaxa(ids=actinoTprF, nproc=10, plim=7)
```

2. Considering the calculated distance for the target misclassified genome against all the rest, it is expectable that the target genome falls closer (lower distance) to genomes belonging to its taxonomic group than to other taxonomic groups. To evaluate this we constructed the MAGO index, that integrates the fraction of members of different taxa (voters) at concentric distances (bands or spheres) around the target genome, normalizing by group abundances and the radial distance to the genome of interest.

$$M_t = \frac{f_1}{r/2} + \frac{f_2}{r(3/2)} + \frac{f_3}{r(5/2)} + \ldots \tag{3}$$

Equation 3 describes MAGO index calculation, where M_t is the index calculated for a certain taxon, f is the fraction of organisms belonging to that taxon present in the current band and r is the radial distance that defines the sphere where voters are being evaluated (Figure 26). M_t terms

will be defined as the number of bands that allocates at least the same number of taxonomic categories than the first band with some voter. For example, in Figure 26 calculations are performed from band 1 (without voters, hence zero) to band 3. Band 4 is not considered because there are not voters for black category, which in fact are present in band 2. In summary, integration stops when a band that did not contain the same categories than the previous is detected.

$$M_A = \frac{0}{1/2} + \frac{5/8}{3/2} + \frac{3/8}{5/2} = \boxed{0.57} \tag{4}$$

$$M_B = \frac{0}{1/2} + \frac{1/10}{3/2} + \frac{6/10}{5/2} = \boxed{0.30} \tag{5}$$

$$M_C = \frac{0}{1/2} + \frac{1/11}{3/2} + \frac{6/11}{5/2} = \boxed{0.27} \tag{6}$$

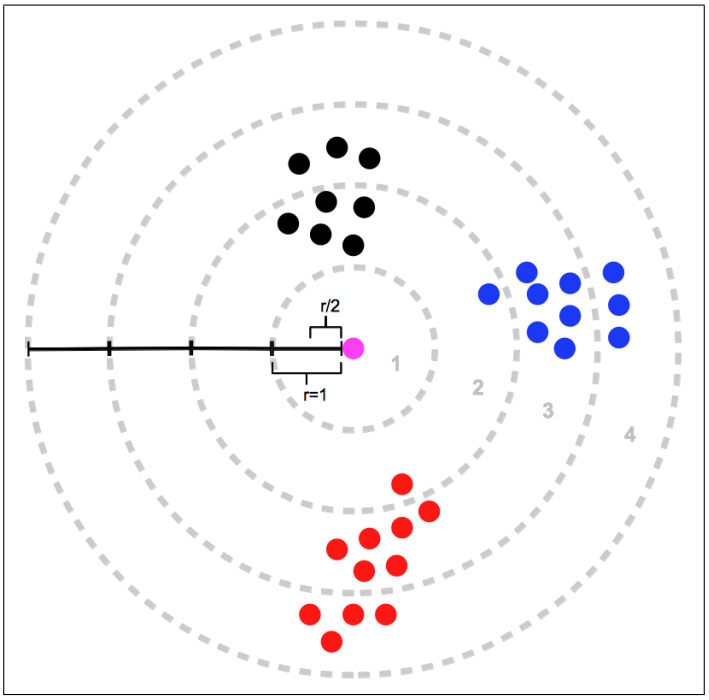


Figure 26: Example case of MAGO index application to solve the membership of the violet genome at the center of the circle.

3. Once MAGO index is calculated for each neighbor taxa, a taxon-specific membership probability is calculated as indicated in Equations 7-9. In this example case, the target misclassified genome (the violet one in Figure 3) would be assigned to taxon A.

$$P_A = \frac{M_A}{M_A + M_B + M_C} = \boxed{0.5} \tag{7}$$

$$P_B = \frac{M_B}{M_A + M_B + M_C} = \boxed{0.26} \tag{8}$$

$$P_C = \frac{M_C}{M_A + M_B + M_C} = \boxed{0.24} \tag{9}$$

As living example of MAGO usefulness, the false negative (TprF) rate for the classification model of phylum Actinobacteria is 0.009, achieving 42 out of 4,357 true Actinobacteria classified as non-Actinobacteria. Using MAGO, 8 out of 42 TprF Actinobacteria were re-assigned to that phylum, reducing the false negative rate in around %20 (Figure 27).

```
# Use iMAGO()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# VCLIST is the output for votingTaxa().

actino.imago<-iMAGO(vclist=actino.vt7,
               rank='phylum')
```

```
     Chlamydiae                Tenericutes
     0.00000000                 0.00000000
  Euryarchaeota              Cyanobacteria
     0.00000000                 0.00000000
   Spirochaetes             Bacteroidetes
     0.00000000                 0.00000000
     Firmicutes            Proteobacteria
     0.00000000                 0.03163734
 Actinobacteria
     0.96836266
```

Figure 27: iMAGO() re-classification for genome 465515.4 belonging to *Micrococcus lutes* NCTC 2665, that was initially misclassified by the Actinobacteria model.

4.1.4 Feature selection

Beyond most classification models can perform very well using the whole dataset that includes 1,328 enzymes, for some particular taxa this big amount of data can cause suboptimal performance. In this sense, feature selection helps to reduce model's complexity allowing shorter processing times and reduction of variance. The central premise of feature selection is that data contains features that are redundant or irrelevant and can be removed without losing discriminatory power.

KARL implements two functions that can be used individually or together to reduce data complexity by selecting most informative enzymes for distinguishing a certain taxon. Note that in this context feature selection must be performed for each model, hence for each taxon:

1. **dropUninformative() -** When comparing a certain taxon against the rest, some enzymes will be present in almost all genomes of that taxon but also in almost all the genomes that do not belong to that taxon, likewise some enzymes will be absent in almost all of both groups. As first step, the function detects enzymes that are present or absent in almost all genomes of the analyzed taxon and the rest and discard them. Then, over the remaining set of enzymes the function calculates pairwise correlations between them and discards those with correlation coefficient higher than a selected threshold. For correlated enzymes, the function keeps the representative with lowest overall correlation values. This process gently reduce the number of enzymes just keeping those with some potential discriminatory power.

```
# Use dropUninformative()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX takes the name of desired taxon to perform evaluation.
# DWTAX is the lower frequency cut-off (default 0.1).
# UPTAX is the upper frequency cut-off (default 0.9).
# CF is the correlation cut-off (default 0.9).

streptococcus.droped<-dropUninformative(rank='genus',
                tax='Streptococcus',
                dwtax=0.1, uptax=0.9, cf=0.9)
```

2. **rankSlope() -** The second step of feature selection involves determining most informative enzymes using an entropy-based measure known as Information Gain (IG) (Figure 28, Equation 11).

$$E = \sum -p_i \log_2 p_i \tag{10}$$

$$IG = E_{parent} - \sum_{i=1}^{n} wE_i \tag{11}$$

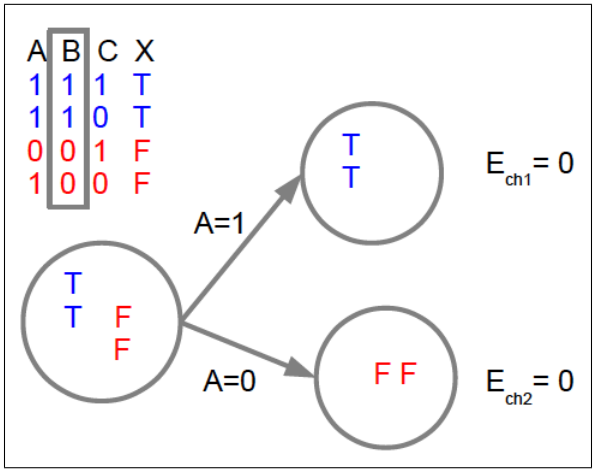


Figure 28: Information Gain calculation for an example enzyme B. IG is maximum (1) because it completely discriminates T from F.

$$E_{parent} = (-0.5)\log_2(0.5) + (-0.5)\log_2(0.5) = \boxed{1} \tag{12}$$

$$E_{child1} = E_{child2} = \boxed{0} \tag{13}$$

$$IG = 1 - 0 = \boxed{1} \tag{14}$$

The function will calculate IG for each enzyme and rank these values. Typically, the ranked values follow a continuous curve, with some enzymes that contribute very little while others have a bigger contribution to category discrimination. However, as this curve is continuous is difficult to empirically set a threshold to decide which enzymes are more informative and keep them. For this reason, the function implements the Davies slope change test for automatically setting an IG threshold that separates informative enzymes from the rest.

```
# Use rankSlope()
# IDATA is the output for dropUninformative().
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX takes the name of desired taxon to perform feature selection.
# RANK and TAX must be the same as in dropUninformative() run.
# PLOTTING is a logical indicating if producing plots or not.

streptococcus.ranked<-rankSlope(idata=streptococcus.droped,
                rank='genus', tax='Streptococcus',
                plotting=T)
```

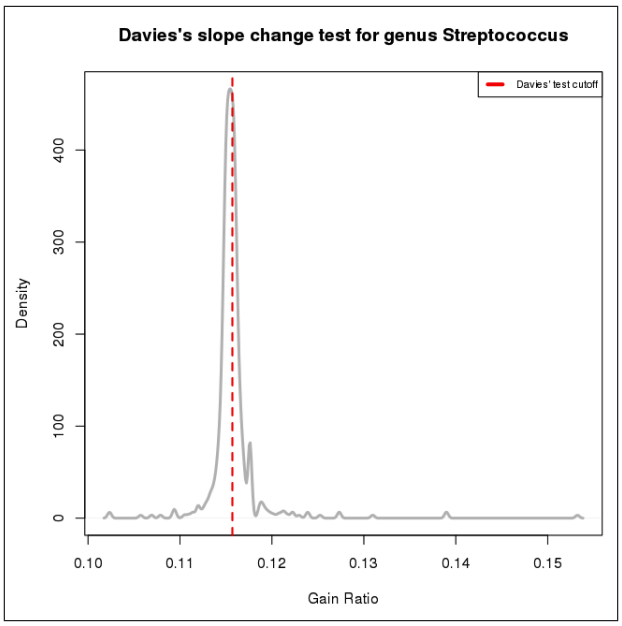


Figure 29: Information Gain threshold calculation by Davies test.

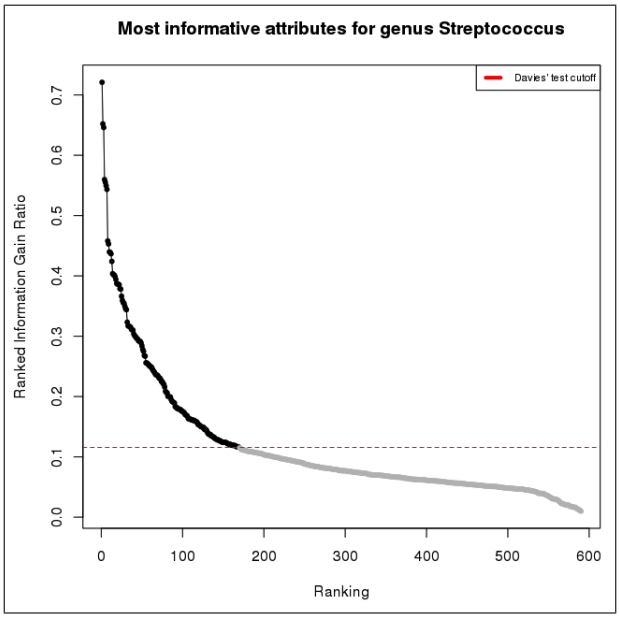


Figure 30: Ranked enzymes based on Information Gain.

In the first step of feature selection, the **dropUninformative()** function keeped 590 enzymes out of $1,328$ for genus *Streptococcus* after filtering by frequency and correlation. Taking this as input, the **rankSlope()** function (graphical output in Figures 29-30) retained 169 out of 590 that passed the threshold of IG >0.1 detected by Davies test.

IMPORTANT NOTES: **Interaction with the Explorer module.**

Note that **rankSlope()** can also be used as a previous step to identify very informative enzymes that are potential markers for a certain taxon. Then one can use the Explorer functions to analyze this subset of enzymes in the context of complete metabolic pathways, as explained in the first section of this manual.

4.2 Preparing input for classification

4.2.1 Starting from sequencing reads

To classify a new genome into taxonomic units using KARL requires to previously detect the presence or absence of a set of enzymes, which are PEGs. The **assemblR()** function implements the SPAdes software [9] to perform an automatic *de novo* assembly over a set of single or paired-end reads provided for the desired genome, informing some basic statistics useful to evaluate the assembly goodness (Figure 31).

```
# Use assemblR()
# R1 is the name of a fastq file containing reads 1.
# R2 is the name of a fastq file containing reads 2 (left empty for
    single read sequencing).
# MODE takes 'fast' for running SPAdes by default (faster), or
    'accurate' for running SPAdes with a wider K-mer range and the
    '--careful' flag (slower but more accurate).
# OUT is the name for the output contigs file.
# REPORT is the name for the assembly report.

assemblR(r1='reads1.fastq',
     r2='reads2.fastq',
     mode='accurate',
     out='contigs.fasta',
     report='assembly_report.txt')
```

```
#### REPORT ####


Genome length: 4339182
Number of contigs: 359
Percentage of contigs >= 1000 bp: 74.65
Percentage of contigs < 1000 bp: 25.35
Min contig length: 100
Max contig length: 92786
Average contig length: 12087
N50: 27939
```

Figure 31: Assembly quality report from **assemblR()** function indicating basic statistics of the assembled genome.

4.2.2 Finding protein encoding genes (PEGs)

Once having assembled contigs from reads (see previous section 3.1), or when starting with unannotated contigs or genomes as input data, the user needs to identify PEGs for posterior enzymatic function assignments. This task is performed by **findR()** function, which implements two well-known gene prediction tools: Glimmer [7] and Prodigal [8].

```
# Use findR()
# GENOME is the name of a fasta file containing the genome (completed or
    contigs).
# COMPLETE takes a logical indicating if GENOME is complete (TRUE) or
    draft (FALSE, default).
# PROGRAM takes 'prodigal', 'glimmer', or 'both' (default). When 'both'
    is chosen identified genes are the intersection of predictions with
    both programs.
# OUTPRE is the prefix for the outfile name.

findR(genome='contigs.fasta',
   complete=FALSE,
   program='both',
   outpre='pegs.fasta')
```

The output will be a multi-fasta file containing all predicted PEGs for the inputed genome in amino acids. This will be the input for enzymatic function prediction, as described in the next section.

4.2.3 Assigning enzymatic functions to PEGs

Once having PEGs defined from the input genome or even when starting from an annotated genome, the **assignR()** function will take a multi-fasta file containing PEGs protein sequences and, using BLAST [6] against enzyme-specific local databases, will create the presence/absence vector which will be used as input for taxonomy classifiers.

```
# Use assignR()
# GENES is the name of a multi-fasta file containing PEGs either in
    nucleotides or amino acids.
# TYPE takes 'prot' or 'nucl' depending on GENES type. If 'nucl',
    sequences will be automatically translated to proteins.
# QC is the query coverage threshold selected to consider a good enough
    hit (default = 90).
# ID is the identity threshold selected to consider a good enough hit
    (default = 70).
# THRDS is the number of threads to be used in BLAST searches.

pavector<-assingnR(genome='pegs.fasta', type='prot',
          qc=90, id=70, thrds=5)
```

4.3 Performing taxonomic predictions

KARL offers two ways of performing taxonomic assignments given the presence/absence vector for a single or a set of genomes. The user can decide on predicting the membership of a certain genome to a particular single taxon or to perform the complete taxonomic prediction from domain to genus, by considering all taxa comprised in the current version of the **alldata** object.

4.3.1 Predicting a single taxon

The function **predictOne()** allows the user to select a particular taxon of interest and predict the membership of a set of genomes to that taxon. The input is a single presence/absence vector or a set of them as a data frame (organisms in rows) (Figure 32). The output is a data frame indicating 'Yes' for the membership of the genome to the selected taxon, or 'No' if it does not belong to that taxon (Figure 33).

	6.3.5.5	6.3.5.6	6.3.5.7	6.4.1.1	6.4.1.2	6.4.1.3	6.4.1.4	6.6.1.1
1000561.3	1	1	1	1	1	0	1	0
1000562.3	1	1	1	0	1	0	0	0
1000565.3	1	1	1	0	1	1	0	1
1000568.3	1	1	1	1	1	1	0	0
1000569.4	1	1	1	1	1	1	0	0

Figure 32: Example input data frame for **predictOne()** showing five genomes (rows) and a subset of enzymes (columns).

```
# Use predictOne()
# RANK: 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX takes the name of desired taxon to perform prediction.
# INPUT is a vector or data frame

input.pred<-predictOne(rank='phylum',
     tax='Proteobacteria',
     input=inputdata)
```

	Proteobacteria
1000561.3	No
1000562.3	Yes
1000565.3	No
1000568.3	Yes
1000569.4	Yes

Figure 33: Example output from **predictOne()** showing the predictions for five genomes in phylum Proteobacteria.

4.3.2 Full prediction

The function **predictFull()** allows to predict the whole lineage of a genome from domain to genus. Making use of the hierarchical nature of taxonomy, this function applies taxon-specific classification models at each taxonomic ranks and decides which models to apply in the next rank according to the previous assignment. Note that for time saving classification is performed from genus to domain.

```
# Use predictFull()
# INPUT is a vector or data frame

full.pred<-predictFull(rank='phylum', input=inputdata)
```

4.4 Predict with 16S

As the 16S gene analysis is a central paradigm in bacterial taxonomy, we have additionally provided inside KARL the **identifR()** function that automatically extracts the 16S gene sequence from a set of contigs or a genome using RNAmmer [10] and then compares this sequence against the Silva database [11] which comes with KARL, retrieving (when possible) the full taxonomic classification for that genome based on its 16S gene.

```
# Use identifiR()
# SEQUENCE takes 'genome' or '16S' indicating if the input file are
    genomic sequences or the 16S gene.
# INSEQ is the name of the input sequence file.
# TAXOUT name for the output file storing the taxonomic classification.
# GENEOUT name for the output file storing identified 16S sequences.

identifiR(inseq='contigs.fasta', sequence='contigs',
     taxout='taxreport.txt', geneout='contigs16S.fasta')
```

```
Your strain was identified at species level with identity > 99%
###### Taxonomy for the closest 16S sequence ######
Bacteria
Actinobacteria
Actinobacteria
Corynebacteriales
Mycobacteriaceae
Mycobacterium
Mycobacterium tuberculosis F11
```

Figure 34: Sample output from **identifiR()**, showing the classification of *Mycobacterium tuberculosis*.

The **textreport.txt** file stores the result for classification at main taxonomic ranks (Figure 34), while the **contigs16S.fasta** file will contain the predicted 16S sequence for that genome. This function can be useful to compare classification results obtained with KARL models and the classical 16S assignment.

5 Updater

KARL Updater module consists in three functions that allow to update the package with newly sequenced and released genomes. Using Updater the user can retrieve and incorporate the information of new genomes without the need of re-installing the package or downloading a new version. One can decide to update all the data and classification models, just the data or just the models. Also the user can decide to keep the updating in the working directory or replace the original data located where KARL is installed.

List of KARL Updater functions:

- **updateData() -** udaptes **alldata** object with new genomes at PATRIC.
- **updateModel() -** updates a single selected model based on new data.
- **updateAll() -** udaptes **alldata** object with new genomes at PATRIC and all models.

5.1 Updating genomic data

Sequencing of new microorganisms, genome re-sequencing and database deposition is a routine pipeline nowadays. Since the power of exploratory analyses using **Explorer** and the accuracy of models in **Predictor** take benefit from this increasing amount of data, KARL datasets can be updated regularly (user's choice) by using the function **updateData()**. This function will automatically connect with the FTP site where genomes are stored in PATRIC database [12], compare the list of genomes in the locally installed version of KARL with those available at PATRIC at this moment, download the new genomes absent in the current version of KARL and parse their annotations to upgrade the enzymes presence/absence matrix (**alldata** object).

By default, the updated data will be stored in a new directory automatically created inside the current working directory, alternatively the user can indicate to replace the old data installed in the package's /**data** folder by the new data.

```
# Use updateData()
# ROUTE is the full URL where genomes are stored at PATRIC database.
# REPLACE is a logical indicating if old data must be replaced or not.

patricurl<-'ftp://ftp.patricbrc.org/patric2/patric3/genomes'
updateData(route=patricurl,replace=T)
```

IMPORTANT NOTES:

- When parameter **replace** is set to **TRUE**, the user needs to restart the R session and re-load the package for making the updates available.
- Running **updateData()** does not affect classification models. To update classification models based on updated genomic data follow the instructions in the following pages.
- To date, the unique way that KARL uses for retrieving new genomic information is through PATRIC. The development of tools for interacting with other databases like the NCBI is scheduled for upcoming versions.

5.2 Updating a single model

Suppose a set of genomes for a novel taxon become available and are retrieved by using **updateData()**. So the user should be interested in having available one classification model for this novel taxon, this can be done with **updateModel()** function. Also the same function can be used with the purpose of updating a model of pre-existing data by applying an automatic feature selection procedure as described in 4.1.4.

```
# Use updateModel()
# RANK takes 'domain', 'phylum', 'class', 'order', 'family' or 'genus'.
# TAX is the name of the taxon.
# REPLACE is a logical indicating if old model must be replaced or not.
# MODE takes 'de' for building the new model with the whole date, or
    'fs' for building the model after automatic feature selection
    procedure.

updateModel(rank='genus',
   tax='Escherichia',
   replace=T,mode='fs')
```

IMPORTANT NOTES:

- When using **mode='fs'**, the automatic feature selection process applies by default a minimum frequency cut-off = 0.1, a maximum frequency cut-off = 0.9 and a correlation cut-off = 0.9 in **dropUninformative()** (dwtax, uptax and cf parameters, respectively). See 4.1.4 for further details.

5.3 Updating all models

Following the same criterion of the previous section, the user can be interested in updating the whole set of models that include all taxa at the six taxonomic ranks evaluated in KARL (domain to genus). For this purpose, the function **updateAll()** functions in the same way as **updateModel()** but it automatically identifies all taxa present in the **alldata** object and re-builds the models. As well, it can replace the old models or store the news separately, also using the whole dataset or taxon-specific feature-selected ones.

```
# Use updateAll()
# REPLACE is a logical indicating if old model must be replaced or not.
# MODE takes 'de' for building the new model with the whole date, or
    'fs' for building the model after automatic feature selection
    procedure.

updateAll(replace=T,mode='fs')
```

IMPORTANT NOTES:

- When using **mode='fs'**, the automatic feature selection process applies by default a minimum frequency cut-off = 0.1, a maximum frequency cut-off = 0.9 and a correlation cut-off = 0.9 in **dropUninformative()** (dwtax, uptax and cf parameters, respectively). See 4.1.4 for further details.
- When using **replace=F** the whole set of new models will be stored in a folder called **new.models** created at the current working directory.

6 References

[1] D. Charif, J. Thioulouse, J. R. Lobry, and G. Perriere. Online synonymous codon usage analyses with the ade4 and seqinR packages. *Bioinformatics*, 21(4):545–547, Feb 2005.

[2] Kurt Hornik, Christian Buchta, and Achim Zeileis. Open-source machine learning: R meets Weka. *Computational Statistics*, 24(2):225–232, 2009.

[3] W. Luo and C. Brouwer. Pathview: an R/Bioconductor package for pathway-based data integration and visualization. *Bioinformatics*, 29(14):1830–1831, Jul 2013.

[4] H. Chen and P. C. Boutros. VennDiagram: a package for the generation of highly-customizable Venn and Euler diagrams in R. *BMC Bioinformatics*, 12:35, 2011.

[5] V. M. Muggeo. Estimating regression models with unknown break-points. *Stat Med*, 22(19):3055–3071, Oct 2003.

[6] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, Oct 1990.

[7] A. L. Delcher, D. Harmon, S. Kasif, O. White, and S. L. Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Res.*, 27(23):4636–4641, Dec 1999.

[8] D. Hyatt, G. L. Chen, P. F. Locascio, M. L. Land, F. W. Larimer, and L. J. Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, 11:119, 2010.

[9] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, A. V. Pyshkin, A. V. Sirotkin, N. Vyahhi, G. Tesler, M. A. Alekseyev, and P. A. Pevzner. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.*, 19(5):455–477, May 2012.

[10] K. Lagesen, P. Hallin, E. A. R?dland, H. H. Staerfeldt, T. Rognes, and D. W. Ussery. RNAmmer: consistent and rapid annotation of ribosomal RNA genes. *Nucleic Acids Res.*, 35(9):3100–3108, 2007.

[11] C. Quast, E. Pruesse, P. Yilmaz, J. Gerken, T. Schweer, P. Yarza, J. Peplies, and F. O. Glockner. The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucleic Acids Res.*, 41(Database issue):D590–596, Jan 2013.

[12] A. R. Wattam, D. Abraham, O. Dalay, T. L. Disz, T. Driscoll, J. L. Gabbard, J. J. Gillespie, R. Gough, D. Hix, R. Kenyon, D. Machi, C. Mao, E. K. Nordberg, R. Olson, R. Overbeek, G. D. Pusch, M. Shukla, J. Schulman, R. L. Stevens, D. E. Sullivan, V. Vonstein, A. Warren, R. Will, M. J.

Wilson, H. S. Yoo, C. Zhang, Y. Zhang, and B. W. Sobral. PATRIC, the bacterial bioinformatics database and analysis resource. *Nucleic Acids Res.*, 42(Database issue):D581–591, Jan 2014.

[13] M. Tanabe and M. Kanehisa. Using the KEGG database resource. *Curr Protoc Bioinformatics*, Chapter 1:Unit1.12, Jun 2012.

[14] Robert G Newcombe. Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in medicine*, 17(8):873–890, 1998.