Classify Sequences

Erik S. Wright

October 24, 2023

Contents

1 Introduction

This document describes how to perform taxonomic classification of amino acid or nucleotide sequences with the DECIPHER package using the *IDTAXA* algorithm. By definition, the taxonomy can be any scheme of classification: organismal, functional, or operational. The *IDTAXA* algorithm is split into two phases: a "training" phase where the classifier learns attributes of the training set, and a "testing" phase where sequences with unknown taxonomic assignments are classified. The objective of sequence classification is to accurately assign a taxonomic label to as many sequences as possible, while refraining from labeling sequences belonging to taxonomic groups that are not represented in the training data. As a case study, the tutorial focuses on classifying a set of 16S ribosomal RNA (rRNA) gene sequences using a training set of 16S rRNA sequences from organisms belonging to known taxonomic groups. Despite the focus on the 16S rRNA gene, the *IDTAXA* process is the same for any set of sequences where there exist a training set with known taxonomic assignments and a testing set with unknown taxonomic assignments.

2 Getting Started

2.1 Startup

To get started we need to load the DECIPHER package, which automatically loads a few other required packages.

```
> library(DECIPHER)
```

The classification process is split into two parts: training carried out by `LearnTaxa` and testing with `IdTaxa`. Help for either function can be accessed through:

```
> ? IdTaxa
```

Once DECIPHER is installed, the code in this tutorial can be obtained via:

```
> browseVignettes("DECIPHER")
```

3 Training the Classifier

The training process only needs to occur once per training set, and results in an object that can be reused for testing as many sequences as desired. If you already have the output of training the classifier (an object of *class* `Taxa` and subclass `Train`), then you can skip to subsection 3.4 (Viewing the training data) below. Otherwise follow along with this section to learn how to train the classifier.

The training process begins with a set of sequence representatives assigned to a taxonomic hierarchy, called a "training set". Typically taxonomic assignments are obtained from an authoritative source, but they can also be automatically created (e.g., with `TreeLine`). Here we describe the general training process, where the classifier iteratively learns about the reference taxonomy.

Note that the training sequences should ideally span the full-length of the gene or target region. The test (query) sequences can be partial length, but the training sequences are expected to overlap the same region as the test sequences. Having excess training sequence beyond the region of the test sequence should not negatively effect performance unless it is very large amount of excess sequence. Also, the training sequences should all have complete taxonomy. That is, every training sequence should be classified to its terminal rank and not have an incomplete classification.

3.1 Importing the training set

The first step is to set filepaths to the sequences (in FASTA format) and the "taxid" file containing information about taxonomic ranks. The "taxid" file is optional, but is often provided (along with training sequences) in a standard 5-column, asterisks ("*") delimited, text format used by many classifiers. To create your own "taxid" file, see 5 (Creating a "taxid" file) below. Be sure to change the path names to those on your system by replacing all of the text inside quotes labeled "<<path to ...>>" with the actual path on your system.

```
> # specify the path to your file of training sequences:
> seqs_path <- "<<path to training FASTA file>>"
> # read the sequences into memory
> seqs <- readDNAStringSet(seqs_path)
> # Alternatively use readAAStringSet or readRNAStringSet
>
> # (optionally) specify a path to the taxid file:
> rank_path <- "<<path to taxid text file>>"
> taxid <- read.table(rank_path,
```

```
          header=FALSE,
          col.names=c('Index', 'Name', 'Parent', 'Level', 'Rank'),
          sep="*", # asterisks delimited
          quote="", # preserve quotes
          stringsAsFactors=FALSE)
> # OR, if no taxid text file exists, use:
> #taxid <- NULL
```

The training sequences cannot contain gap ("-" or ".") characters, which can easily be removed with the `RemoveGaps` function:

```
> # if they exist, remove any gaps in the sequences:
> seqs <- RemoveGaps(seqs)
```

Note that the training sequences must all be in the same orientation. If this is not the case, it is possible to reorient the sequences with `OrientNucleotides`:

```
> # ensure that all sequences are in the same orientation:
> seqs <- OrientNucleotides(seqs)
```

Here, we make the assumption that each sequence is labeled in the original (FASTA) file by its taxonomy starting with "Root;". For example, a sequence might be labeled "AY193173 Root; Bacteria; SR1; SR1_genera_incertae_sedis", in which case we can extract all of the text starting from "Root;" to obtain the sequence's "group". In this context, groups are defined as the set of all possible taxonomic labels that are present in the training set.

```
> # obtain the taxonomic assignments
> groups <- names(seqs) # sequence names
> # assume the taxonomy begins with 'Root;'
> groups <- gsub("(.*)(Root;)", "\\2", groups) # extract the group label
> groupCounts <- table(groups)
> u_groups <- names(groupCounts) # unique groups
> length(u_groups) # number of groups
```

3.2 Pruning the training set

The next step is to count the number of representatives per group and, *optionally*, select only a subset of sequences if the group is deemed too large. Typically there is a diminishing return in accuracy for having more-and-more representative sequences in a group. Limiting groups size may be advantageous if some groups contain an inordinately large number of sequences because it will speed up the classification process. Also, larger groups oftentimes accumulate errors (that is, sequences which do not belong), and constraining the group size can help to make the classification process more robust to rare errors that may exist in the training data. In the code below, *maxGroupSize* controls the maximum size of any group, and can be set to `Inf` (infinity) to allow for an unlimited number of sequences per group.

```
> maxGroupSize <- 10 # max sequences per label (>= 1)
> remove <- logical(length(seqs))
> for (i in which(groupCounts > maxGroupSize)) {
          index <- which(groups==u_groups[i])
          keep <- sample(length(index),
                  maxGroupSize)
          remove[index[-keep]] <- TRUE
 }
> sum(remove) # number of sequences eliminated
```

3.3 Iteratively training the classifier

Now we must train the classifier on the training set. One unique feature of the *IDTAXA* algorithm is that during the learning process it will identify any training sequences whose assigned classifications completely (with very high confidence) disagree with their predicted classification. These are almost always sequences that are mislabeled in the training data, and they can make the classification process slower and less accurate because they introduce error in the training data. We have the option of automatically removing these putative "problem sequences" by iteratively repeating the training process. However, we may also want to be careful not to remove sequences that are the last remaining representatives of an entire group in the training data, which can happen if the entire group appears to be misplaced in the taxonomic tree. These two training options are controlled by the *maxIterations* and *allowGroupRemoval* variables (below). Setting *maxIterations* to `1` will simply train the classifier without removing any problem sequences, whereas values greater than `1` will iteratively remove problem sequences.

```
> maxIterations <- 3 # must be >= 1
> allowGroupRemoval <- FALSE
> probSeqsPrev <- integer() # suspected problem sequences from prior iteration
> for (i in seq_len(maxIterations)) {
        cat("Training iteration: ", i, "\n", sep="")

        # train the classifier
        trainingSet <- LearnTaxa(seqs[!remove],
                names(seqs)[!remove],
                taxid)

        # look for problem sequences
        probSeqs <- trainingSet$problemSequences$Index
        if (length(probSeqs)==0) {
                cat("No problem sequences remaining.\n")
                break
        } else if (length(probSeqs)==length(probSeqsPrev) &&
                all(probSeqsPrev==probSeqs)) {
                cat("Iterations converged.\n")
                break
        }
        if (i==maxIterations)
                break
        probSeqsPrev <- probSeqs

        # remove any problem sequences
        index <- which(!remove)[probSeqs]
        remove[index] <- TRUE # remove all problem sequences
        if (!allowGroupRemoval) {
                # replace any removed groups
                missing <- !(u_groups %in% groups[!remove])
                missing <- u_groups[missing]
                if (length(missing) > 0) {
                        index <- index[groups[index] %in% missing]
                        remove[index] <- FALSE # don't remove
                }
        }
 }
> sum(remove) # total number of sequences eliminated
```

```
> length(probSeqs) # number of remaining problem sequences
```

3.4 Viewing the training data

The training process results in a training object (`trainingSet`) of *class* `Taxa` and subclass `Train` that contains all of the information required for classification. If you want to use the pre-trained classifier for 16S rRNA sequences, then it can be loaded with the `data` function. However, **if you just trained the classifier using your own training data then you should skip these next two lines of code**.

```
> data("TrainingSet_16S")
> trainingSet <- TrainingSet_16S
```

We can view summary properties of the training set (`trainingSet`) by printing it:

```
> trainingSet
  A training set of class 'Taxa'
   * K-mer size: 8
   * Number of rank levels: 10
   * Total number of sequences: 2472
   * Number of groups: 2472
   * Number of problem groups: 5
   * Number of problem sequences: 8
```

And, as shown in Figure 1, we can plot the training set (`trainingSet`) to view a variety of information:

1. The first panel contains the taxonomic tree with the "Root" at the very top. This training set contains different numbers of ranks for each group, which is why the leaves of the tree end at different heights. Edges of the tree that are colored show putative "problem groups" that persist after the iterative removal of "problem sequences" (see above). These colored edges are problematic in that the classifier cannot descend below this edge on the tree during the initial "tree descent" phase of the algorithm. This slows down the classification process for sequences belonging to a group below this edge, but does not affect the classifier's accuracy.

2. The second panel of Fig. 1 shows the number of unique groups at each taxonomic rank, ordered from highest to lowest taxonomic rank in the dataset. We can see that there are about 2.5 thousand genera, where *genus* is the lowest rank in this training set.

3. The bottom left panel contains a histogram of the number of sequences per group. The maximum group size is in accordance with the *maxGroupSize* set above. Here, the pre-trained classified has only a single sequence per group so that it will take up minimal space. Typically classifiers will have a wide distribution of the number of sequences per group.

4. The bottom right panel displays the *inverse document frequency* (IDF) weights associated with each k-mer. We can see that there are many rare k-mers that have high weights (i.e., high information content), and a few common k-mers that have very low weights. This highly-skewed distribution of information content among k-mers is typical among sequence data.

```
> plot(trainingSet)
```

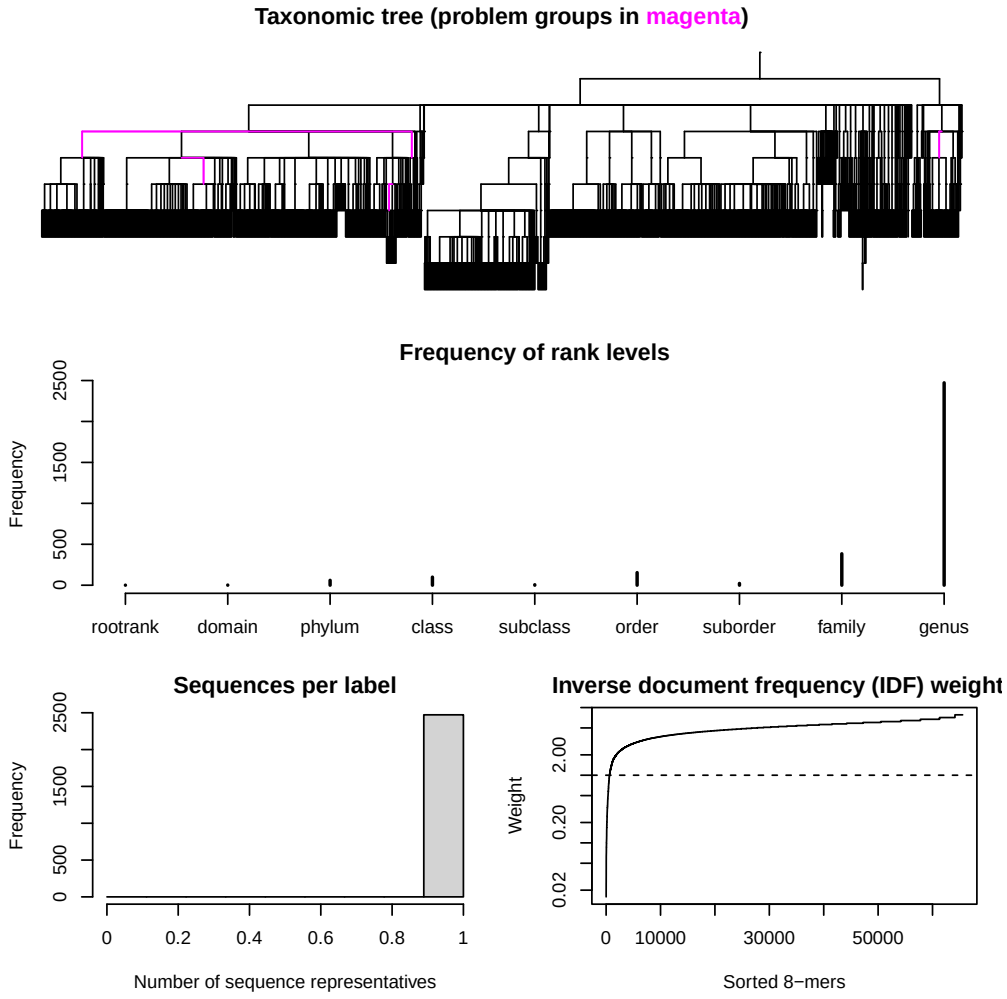


Figure 1: Result of plotting the training set (`trainingSet`) produced by `LearnTaxa`.

4 Classifying Sequences

Now that we have trained the classifier, the next step is to use it to assign taxonomic classifications to new sequences. This is accomplished with the `IdTaxa` function, which takes in "test" (new) sequences along with the training set (`trainingSet`) object that was returned by `LearnTaxa`. For the purposes of this tutorial, we are going to use some 16S rRNA gene sequences collected from organisms present in tap water. Feel free to follow along with your own sequences, or load the FASTA file included with the tutorial.

```
> fas <- "<<path to FASTA file>>"
> # OR use the example 16S sequences:
> fas <- system.file("extdata",
          "Bacteria_175seqs.fas",
          package="DECIPHER")
> # read the sequences into memory
> test <- readDNAStringSet(fas)
> # Alternatively use readAAStringSet or readRNAStringSet
```

As in training (above), the test sequences cannot contain gap ("-" or ".") characters, which can easily be removed with the `RemoveGaps` function:

```
> # if they exist, remove any gaps in the sequences:
> test <- RemoveGaps(test)
> test
DNAStringSet object of length 175:
      width seq                                               names
  [1]  1235 TCTGATATAGCGGCGGACGGGT...TTCTCAGTTCGGATTGTAGGCT uncultured bacter...
  [2]  1351 TTAGCGGCGGACGGGTGAGTAA...GAGTTTGTAACACCCGAAGCCG uncultured bacter...
  [3]  1326 CGGCGGACGGGTGAGTAACACG...CACCGCCCGTCACACCACGAGA uncultured bacter...
  [4]  1345 GCGAACGGGTGAGTAACACGTG...TTGGAACACCCGAAGTCGGCCG uncultured bacter...
  [5]  1343 AACGCGTGGGTAACCTACCCAT...GTCTGCACACCCGAAGCCGGTG uncultured bacter...
  ...   ... ...
[171]  1314 CGGACGGGTGAGTAAAGCATAG...GCCCGTCACACCATGGGAGTGG uncultured bacter...
[172]  1316 ACGGGTGAGTAATGCTTAGGAA...CCCGTCACACCATGGGAGTTGG uncultured bacter...
[173]  1308 GGCAACCCCAGAGAATGGCGAA...TGAACACGTTCCCGGGCCTTGT uncultured bacter...
[174]  1313 GACGGGTGGTTAACACGTAGGT...AGAGGGTCACGCCCGAAGTCGG uncultured bacter...
[175]  1333 CTTTCGGGGGTGCTTCAGTGGC...CGAAAGAAGGTCACGCCCGAAG uncultured bacter...
```

4.1 Assigning classifications

Now, for the moment we have been waiting for: it's time to classify some test sequences! It's important to have read the help file for `IdTaxa` to acquaint yourself with the available options before performing this step. The most important (optional) arguments are the *type* of output, the *strand* used in testing, the confidence *threshold* of assignments, and the number of *processors* to use. Here, we are going to request the `"extended"` (default) output *type* that allows for plotting the results, but there is also a `"collapsed"` *type* that might be easier to export (see section 4.4 below). Also, we know that all of the test sequences are in the same ("+" strand) orientation as the training sequences, so we can specify to only look at the `"top"` strand rather than the default of `"both"` strands (i.e., both "+" and "-" strands). This makes the classification process over twice as fast. We could also set *processors* to `NULL` to use all available processors.

```
> ids <- IdTaxa(test,
          trainingSet,
```

```
        type="extended",
        strand="top",
        threshold=60,
        processors=1)

  |======================================================================| 100%

Time difference of 12.23 secs
```

The `threshold` of 60% is recommended at the default confidence threshold. Confidence levels are informally defined as 70% (stringent), 60% (cautious), 50% (sensible), and 40% (lenient). Using a `threshold` of 0% will report classifications down to all rank levels. Note that the test sequences should generally be fully-overlapped by the information in the training sequences. In this way, the training sequences can be longer than the test sequences, but the reverse situation would result in lower confidences.
Let's look at the results by printing the object (*ids*) that was returned:

```
> ids
  A test set of class 'Taxa' with length 175
     confidence name                 taxon
  [1]        73% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Ba...
  [2]        68% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Ba...
  [3]        63% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Ba...
  [4]        92% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; La...
  [5]        61% uncultured bacter... Root; Bacteria; Firmicutes; Clostridia;...
  ...        ... ...                  ...
[171]        39% uncultured bacter... Root; unclassified_Root
[172]        48% uncultured bacter... Root; unclassified_Root
[173]        31% uncultured bacter... Root; unclassified_Root
[174]        49% uncultured bacter... Root; unclassified_Root
[175]        54% uncultured bacter... Root; unclassified_Root
```

Note that the data has *class* `Taxa` and subclass `Test`, which is stored as an object of *type list*. Therefore we can access a subset of the returned object (*ids*) with single square brackets (`[`) or access the contents of individual list elements with double square brackets (`[[`):

```
> ids[1:5] # summary results for the first 5 sequences
  A test set of class 'Taxa' with length 5
    confidence name                 taxon
[1]        73% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Baci...
[2]        68% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Baci...
[3]        63% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Baci...
[4]        92% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Lact...
[5]        61% uncultured bacter... Root; Bacteria; Firmicutes; Clostridia; C...
> ids[[1]] # results for the first sequence
$taxon
[1] "Root"                          "Bacteria"
[3] "Firmicutes"                    "Bacilli"
[5] "Bacillales"                    "Planococcaceae"
[7] "unclassified_Planococcaceae"

$confidence
```

```
[1] 74.21434 74.21434 74.21434 74.21434 74.21434 73.49649 73.49649

$rank
[1] "rootrank" "domain"   "phylum"   "class"    "order"    "family"   "genus"
> ids[c(10, 25)] # combining different sequences
  A test set of class 'Taxa' with length 2
    confidence name                 taxon
[1]        49% uncultured bacter... Root; unclassified_Root
[2]        34% uncultured bacter... Root; unclassified_Root
> c(ids[10], ids[25]) # merge different sets
  A test set of class 'Taxa' with length 2
    confidence name                 taxon
[1]        49% uncultured bacter... Root; unclassified_Root
[2]        34% uncultured bacter... Root; unclassified_Root
> ids[, c("rootrank", "domain", "class")] # only look at specific rank levels
  A test set of class 'Taxa' with length 175
      confidence name                 taxon
  [1]        74% uncultured bacter... Root; Bacteria; Bacilli
  [2]        71% uncultured bacter... Root; Bacteria; Bacilli
  [3]        68% uncultured bacter... Root; Bacteria; Bacilli
  [4]        92% uncultured bacter... Root; Bacteria; Bacilli
  [5]        66% uncultured bacter... Root; Bacteria; Clostridia
  ...        ... ...                  ...
[171]        39% uncultured bacter... Root; unclassified_Root
[172]        48% uncultured bacter... Root; unclassified_Root
[173]        31% uncultured bacter... Root; unclassified_Root
[174]        49% uncultured bacter... Root; unclassified_Root
[175]        54% uncultured bacter... Root; unclassified_Root
> ids[threshold=70] # threshold the results at a higher confidence
  A test set of class 'Taxa' with length 175
      confidence name                 taxon
  [1]        73% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Ba...
  [2]        71% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Ba...
  [3]        68% uncultured bacter... Root; unclassified_Root...
  [4]        92% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; La...
  [5]        66% uncultured bacter... Root; unclassified_Root...
  ...        ... ...                  ...
[171]        39% uncultured bacter... Root; unclassified_Root
[172]        48% uncultured bacter... Root; unclassified_Root
[173]        31% uncultured bacter... Root; unclassified_Root
[174]        49% uncultured bacter... Root; unclassified_Root
[175]        54% uncultured bacter... Root; unclassified_Root
```

The output can easily be converted to a character vector with taxonomic information assigned to each sequence:

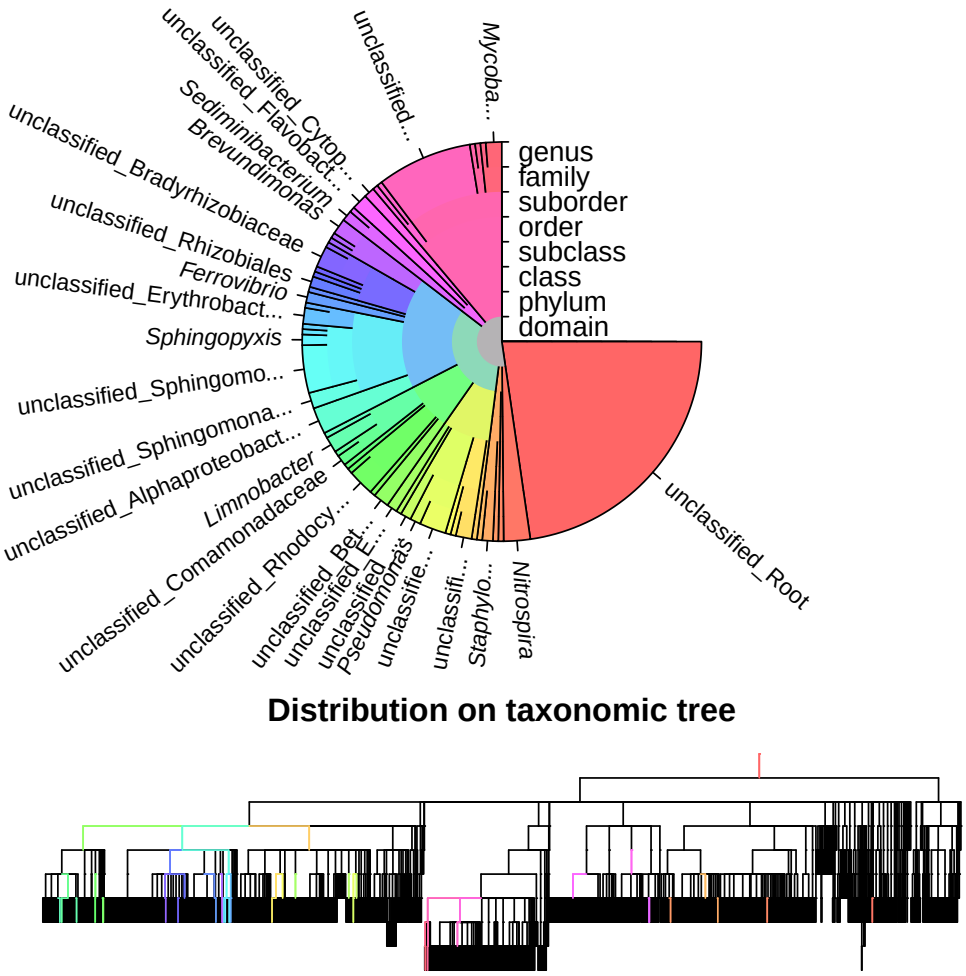
```
> assignment <- sapply(ids,
        function(x)
                paste(x$taxon,
                        collapse=";"))
> head(assignment)
```

```
                                                    uncultured bacterium; Pro_CL-05069_OTU-15.
"Root;Bacteria;Firmicutes;Bacilli;Bacillales;Planococcaceae;unclassified_Planococcaceae"
                                                    uncultured bacterium; Fin_CL-100646_OTU-6.
          "Root;Bacteria;Firmicutes;Bacilli;Bacillales;Staphylococcaceae;Staphylococcus"
                                                   uncultured bacterium; Mar_CL-050642_OTU-13.
          "Root;Bacteria;Firmicutes;Bacilli;Bacillales;Staphylococcaceae;Staphylococcus"
                                                    uncultured bacterium; Mar_CL-100626_OTU-8.
     "Root;Bacteria;Firmicutes;Bacilli;Lactobacillales;Carnobacteriaceae;Dolosigranulum"
                                                   uncultured bacterium; Fin_CL-100633_OTU-22.
  "Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Peptococcaceae 1;Desulfosporosinus"
                                                    uncultured bacterium; Fin_CL-050645_OTU-2.
                                                                 "Root;unclassified_Root"
```

4.2 Plotting the results

We can also plot the results, as shown in Figure 2. This produces a pie chart showing the relative abundance of the taxonomic groups assigned to test sequences. It also displays the training taxonomic tree, with edges colored where they match the taxonomic groups shown in the pie chart. Note that we could also have only plotted the pie chart by omitting the `trainingSet`. Also, it is possible to specify the parameter *n* if each classification represents a varying number of sequences, e.g., when only unique sequences were originally classified.

```
> plot(ids, trainingSet)
```



Distribution on taxonomic tree

Figure 2: Result of plotting the classifications (`ids`) made by `IdTaxa`.

4.3 Create and plot a classification table

When analyzing multiple samples, it is often useful to create a classification table with the number of times each taxon is observed. Here we can choose a specific taxonomic rank to consider, or simply select the lowest (i.e., basal) taxonomic level:

```
> phylum <- sapply(ids,
         function(x) {
                 w <- which(x$rank=="phylum")
                 if (length(w) != 1) {
                         "unknown"
                 } else {
                         x$taxon[w]
                 }
         })
> table(phylum)
phylum
"Actinobacteria"  "Bacteroidetes" "Proteobacteria"       Firmicutes
              26                8               78                5
     Nitrospirae          unknown
               5               53
> taxon <- sapply(ids,
         function(x)
                 x$taxon[length(x$taxon)])
> head(taxon)
 uncultured bacterium; Pro_CL-05069_OTU-15.
              "unclassified_Planococcaceae"
 uncultured bacterium; Fin_CL-100646_OTU-6.
                           "Staphylococcus"
uncultured bacterium; Mar_CL-050642_OTU-13.
                           "Staphylococcus"
 uncultured bacterium; Mar_CL-100626_OTU-8.
                           "Dolosigranulum"
uncultured bacterium; Fin_CL-100633_OTU-22.
                         "Desulfosporosinus"
 uncultured bacterium; Fin_CL-050645_OTU-2.
                         "unclassified_Root"
```

Next, we need to know which test sequences belonged to each sample. This must be in the form of a vector of sample names that is the same length as the number of samples. For example, in this case the sample names are part of the sequence names. Using this vector we can easily generate a classification table:

```
> # get a vector with the sample name for each sequence
> samples <- gsub(".*; (.+?)_.*", "\\1", names(test))
> taxaTbl <- table(taxon, samples)
> taxaTbl <- t(t(taxaTbl)/colSums(taxaTbl)) # normalize by sample
> head(taxaTbl)
                samples
taxon              Chlminus    Chlplus        Fin        LO1        LO3
  Achromobacter  0.00000000 0.03846154 0.00000000 0.00000000 0.00000000
  Acidovorax     0.07692308 0.00000000 0.00000000 0.00000000 0.00000000
  Afipia         0.07692308 0.00000000 0.00000000 0.00000000 0.00000000
```

```
    Asinibacterium 0.00000000 0.03846154 0.00000000 0.00000000 0.00000000
    Blastomonas    0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
    Brevundimonas  0.00000000 0.03846154 0.00000000 0.00000000 0.12500000
                  samples
taxon                    Mar        Pro        UWH        UWL
    Achromobacter  0.00000000 0.00000000 0.00000000 0.00000000
    Acidovorax     0.00000000 0.00000000 0.00000000 0.00000000
    Afipia         0.00000000 0.00000000 0.00000000 0.00000000
    Asinibacterium 0.00000000 0.00000000 0.00000000 0.00000000
    Blastomonas    0.00000000 0.00000000 0.04166667 0.00000000
    Brevundimonas  0.00000000 0.00000000 0.04166667 0.00000000
```

We can summarize the results in a stacked `barplot`:

```
> include <- which(rowMeans(taxaTbl) >= 0.04)
> barplot(taxaTbl[include,],
          legend=TRUE,
          col=rainbow(length(include), s=0.4),
          ylab="Relative abundance",
          ylim=c(0, 1),
          las=2, # vertical x-axis labels
          args.legend=list(x="topleft", bty="n", ncol=2))
```

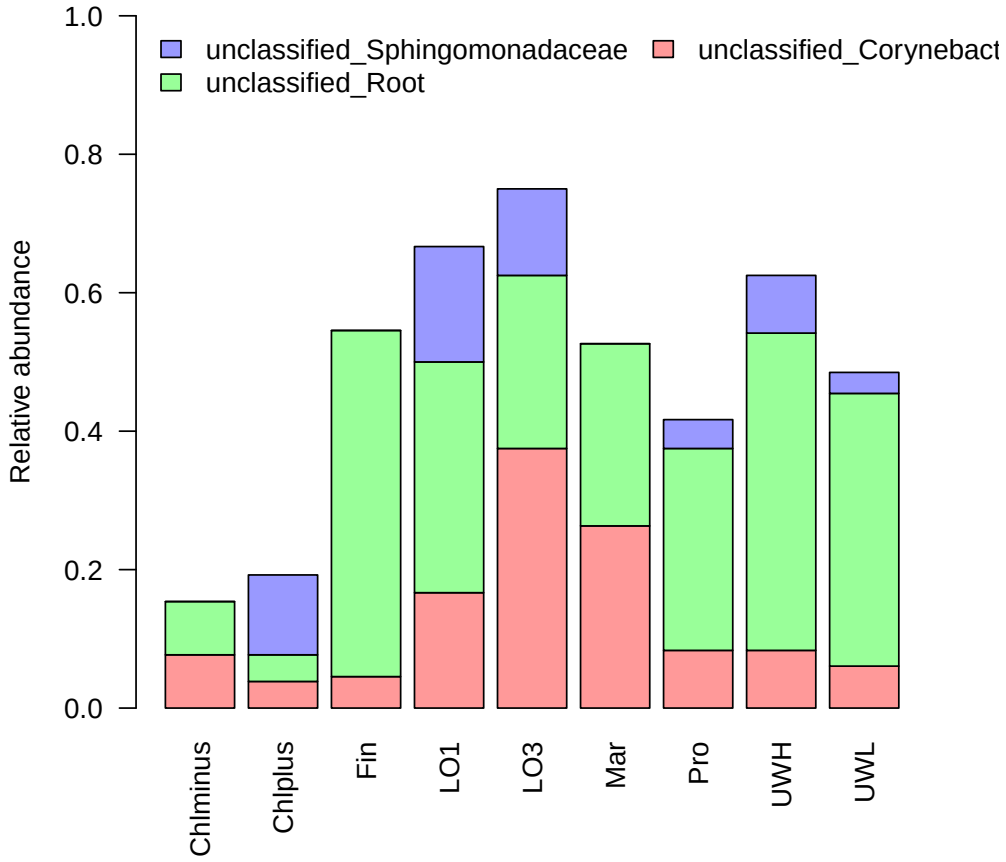


Figure 3: Barplot of taxonomic assignments by sample.

4.4 Exporting the classifications

We can switch between outputting in `extended` or `collapsed` format by setting the *type* argument in `IdTaxa`. The `collapsed` *type* of output is simply a character vector, which cannot be plotted but is easy to write to a text file with the `writeLines` function. In this tutorial we requested the `extended` *type* of output, which is stored in a list structure that must be converted into a character vector before we can write it to a text file. Here we may choose what we want the text output to look like, by pasting together the result for each sequence using delimiters. For example:

```
> output <- sapply(ids,
          function (id) {
                  paste(id$taxon,
                          " (",
                          round(id$confidence, digits=1),
                          "%)",
                          sep="",
                          collapse="; ")
          })
> tail(output)
 uncultured bacterium; UWH_CL-010746_OTU-2.
      "Root (46%); unclassified_Root (46%)"
 uncultured bacterium; UWH_CL-01079_OTU-21.
      "Root (39%); unclassified_Root (39%)"
  uncultured bacterium; UWH_CL-08061_OTU-5.
  "Root (48.6%); unclassified_Root (48.6%)"
 uncultured bacterium; Fin_CL-03079_OTU-21.
  "Root (31.7%); unclassified_Root (31.7%)"
uncultured bacterium; UWL_CL-110518_OTU-11.
  "Root (49.3%); unclassified_Root (49.3%)"
uncultured bacterium; UWL_CL-110548_OTU-32.
  "Root (54.1%); unclassified_Root (54.1%)"
> #writeLines(output, "<<path to output text file>>")
```

4.5 Guaranteeing repeatability

The *IDTAXA* algorithm uses bootstrapping, which involves random sampling to obtain a confidence score. For this reason, the classifications are expected to change slightly if the classification process is repeated with the same inputs. For some applications this randomness is undesirable, and it can easily be avoided by setting the random seed before classification. The process of setting and then unsetting the seed in R is straightforward:

```
> set.seed(123) # choose a whole number as the random seed
> # then classify sequences with IdTaxa (not shown)
> set.seed(NULL) # return to the original state by unsetting the seed
```

5 Creating a "taxid" file

The "taxid" file format supplies a table that can be used by `LearnTaxa` to specify taxonomic ranks (e.g., phylum, class, order, etc.) associated with each taxon. Previously we imported the rank information from a plain text file containing 5 columns separate by asterisks ("*"). An example of the contents of this file is:

```
0*Root*-1*0*rootrank
1*Bacteria*0*1*domain
2*Actinobacteria*1*2*phylum
3*Acidimicrobiales*2*3*order
4*Acidimicrobiaceae*3*4*family
5*Acidimicrobium*4*5*genus
6*Ferrimicrobium*4*5*genus
...
```

The leftmost column is simply an index starting at zero. Next, there is a column with each unique taxonomic name in the training set. The third column contains a pointer to the index of each line's parent. The fourth column gives the rank level starting from "Root" at level 0. The last column provides the taxonomic rank information that is used by `LearnTaxa`.

The first line is always the same, and specifies that the "Root" rank is index 0, has no parent (-1), and points to itself (index 0). The rest of the lines must point to a positive index for their parent. For example, the line for index 6 states that the genus "Ferrimicrobium" exist within the family "Acidimicrobiaceae" (index 4) at rank level 5.

If you would like to create a custom "taxid" file for your training set, the easiest way is to start with a set of taxonomic labels preceded by prefixes indicating their rank. For example, the above "taxid" file could be generated from these lines of text:

```
d__Bacteria;p__Actinobacteria;o__Acidimicrobiales;f__Acidimicrobiaceae;g__Acidimicrobium
d__Bacteria;p__Actinobacteria;o__Acidimicrobiales;f__Acidimicrobiaceae;g__Ferrimicrobium
...
```

Then the following code will convert this text into the fields required for the "taxid" file:

```
> ranks <- readLines("<<path to lines of text>>")
> taxa <- setNames(c("domain", "phylum", "order", "family", "genus"),
                   c("d__", "p__", "o__", "f__", "g__"))
> ranks <- strsplit(ranks, ";", fix=T)
> count <- 1L
> groups <- "Root"
> index <- -1L
> level <- 0L
> rank <- "rootrank"
> pBar <- txtProgressBar(style=3)
> for (i in seq_along(ranks)) {
          for (j in seq_along(ranks[[i]])) {
                   rank_level <- taxa[substring(ranks[[i]][j], 1, 3)]
                   group <- substring(ranks[[i]][j], 4)
                   w <- which(groups==group & rank==rank_level)
                   if (length(w) > 0) {
                            parent <- match(substring(ranks[[i]][j - 1], 4),
                                     groups)
                            if (j==1 || any((parent - 1L)==index[w]))
                                     next # already included
                   }

                   count <- count + 1L
                   groups <- c(groups, group)
                   if (j==1) {
                            index <- c(index, 0)
```

```
                } else {
                        parent <- match(substring(ranks[[i]][j - 1], 4),
                                groups)
                        index <- c(index,
                                parent - 1L)
                }
                level <- c(level, j)
                rank <- c(rank, taxa[j])
        }

        setTxtProgressBar(pBar, i/length(ranks))
 }
> groups <- gsub("^[ ]+", "", groups)
> groups <- gsub("[ ]+$", "", groups)
> taxid <- paste(0:(length(index) - 1L), groups, index, level, rank, sep="*")
> head(taxid, n=10)


[1] "0*Root*-1*0*rootrank"
[2] "1*Bacteria*0*1*domain"
[3] "2*Actinobacteria*1*2*phylum"
[4] "3*Acidimicrobiales*2*3*order"
[5] "4*Acidimicrobiaceae*3*4*family"
[6] "5*Acidimicrobium*4*5*genus"
[7] "6*Ferrimicrobium*4*5*genus"
```

Now these lines of text can be written to a file to be imported as the "taxid" file above.

```
> writeLines(taxid,
        con="<<path to taxid file>>")
```

6 Annotating Protein Sequences

The *IDTAXA* algorithm can also be used to classify amino acid sequences into a taxonomy of genes, functions, or organisms. As an example, we can train the classifier on the set of named genes from the phylum Planctobacteria. Sets such as this can be constructed from various databases, including https://www.uniprot.org/uniprot/?query=reviewedSw
The example training sequences can be loaded with:

```
> fas <- system.file("extdata",
        "PlanctobacteriaNamedGenes.fas.gz",
        package="DECIPHER")
> aa <- readAAStringSet(fas)
> aa
AAStringSet object of length 2497:
      width seq                                               names
   [1]   227 MAGPKHVLLVSEHWDLFFQTKE...VGYLFSDDGDKKFSQQDTKLS A0A0H3MDW1|Root;N...
   [2]   394 MKRNPHFVSLTKNYLFADLQKR...GKREDILAACERLQMAPALQS O84395|Root;2;6;1...
   [3]   195 MAYGTRYPTLAFHTGGIGESDD...GFCLTALGFLNFENAEPAKVN Q9Z6M7|Root;4;1;1...
   [4]   437 MMLRGVHRIFKCFYDVVLVCAF...TASFDRTWRALKSYIPLYKNS Q46222|Root;2;4;9...
   [5]   539 MSFKSIFLTGGVVSSLGKGLTA...FIEFIRAAKAYSLEKANHEHR Q59321|Root;6;3;4...
   ...   ... ...
```

```
[2493]  1038 MFEEVLQESFDEREKKVLKFWQ...EGTDWDLNGEPTKIIIKKSEY Q6MDY1|Root;6;1;1...
[2494]   102 MVQIVSQDNFADSIASGLVLVD...VERSVGLKDKDSLVKLISKHQ Q9PJK3|Root;NoEC;...
[2495]   224 MKPQDLKLPYFWEDRCPKIENH...NLWRSKGEKIFCTEFVKRVGI Q9PL91|Root;2;1;1...
[2496]   427 MLRRLFVSTFLIFGMVSLYAKD...KIVIGLGEKRFPSWGGFPNNQ Q256H8|Root;NoEC;...
[2497]   344 MLTLGLESSCDETACALVDAKG...GIHPCARYHWESISASLSPLP Q822Y4|Root;2;3;1...
> head(names(aa))
[1] "A0A0H3MDW1|Root;NoEC;chxR"        "O84395|Root;2;6;1;83;dapL"
[3] "Q9Z6M7|Root;4;1;1;19;aaxB"        "Q46222|Root;2;4;99;Multiple;waaA"
[5] "Q59321|Root;6;3;4;2;pyrG"         "P0C0Z7|Root;NoEC;groL"
```

Here, protein sequences are named by their enzyme commission (EC) number and three or four-letter gene name. It is important to only train the classifier with sequences having complete labels. In this case, we will get rid of sequences without any EC number.

```
> aa <- aa[!grepl("Root;NoEC", names(aa), fixed=TRUE)]
> aa
AAStringSet object of length 1542:
       width seq                                               names
   [1]   394 MKRNPHFVSLTKNYLFADLQKR...GKREDILAACERLQMAPALQS O84395|Root;2;6;1...
   [2]   195 MAYGTRYPTLAFHTGGIGESDD...GFCLTALGFLNFENAEPAKVN Q9Z6M7|Root;4;1;1...
   [3]   437 MMLRGVHRIFKCFYDVVLVCAF...TASFDRTWRALKSYIPLYKNS Q46222|Root;2;4;9...
   [4]   539 MSFKSIFLTGGVVSSLGKGLTA...FIEFIRAAKAYSLEKANHEHR Q59321|Root;6;3;4...
   [5]    92 MQVNEYFDGNVTSIAFENGEGR...DANQKFQVRVIEPTAYLCFYS Q7USA1|Root;2;4;2...
   ...   ... ...
[1538]  1036 MDNEDKISISAKEEKILSFWKE...EGEEWDINGHAVSFVLERVER B0BB05|Root;6;1;1...
[1539]   342 MTIQEELEAVKQQFSCDVSLAH...YGISDIRLFSENDLRFLRQFS O84843|Root;6;1;1...
[1540]  1038 MFEEVLQESFDEREKKVLKFWQ...EGTDWDLNGEPTKIIIKKSEY Q6MDY1|Root;6;1;1...
[1541]   224 MKPQDLKLPYFWEDRCPKIENH...NLWRSKGEKIFCTEFVKRVGI Q9PL91|Root;2;1;1...
[1542]   344 MLTLGLESSCDETACALVDAKG...GIHPCARYHWESISASLSPLP Q822Y4|Root;2;3;1...
```

Since this taxonomy contains widely disparate sequences, we would not expect tree descent to be useful. We can disable tree descent by setting *maxChildren* to `1` in `LearnTaxa`.

```
> trainingSet <- LearnTaxa(train=aa,
         taxonomy=names(aa),
         maxChildren=1)
================================================================================

Time difference of 1.76 secs
```

Next we need a set of query sequences to classify. To this end, we will use a representative genome of the *Chlamydia trachomatis* species, a member of the Planctobacteria phylum. We can find genes in the genome using the DECIPHER function `FindGenes`.

```
> fas <- system.file("extdata",
         "Chlamydia_trachomatis_NC_000117.fas.gz",
         package="DECIPHER")
> genome <- readDNAStringSet(fas)
> genes <- FindGenes(genome, verbose=FALSE)
> test <- ExtractGenes(genes, genome, type="AAStringSet")
> test
```

```
AAStringSet object of length 897:
      width seq
  [1]   392 AAAREIAKRWEQRVRDLQDKGAARKLLNDPLGR...QVEGILRDMLTNGSQTFRDLMRRWNREVDRE*
  [2]    91 MLCKVCRGLSSLIVVLGAINTGILGVTGYKVNL...CLNFLKCCFKKRHGDCCSSKGGYHHHHMDRE*
  [3]   101 MTESYVNKEEIISLAKNAALELEDAHVEEFVTS...DMVTSDFTQEEFLSNVPVSLGGLVKVPTVIK*
  [4]   492 MYRKSALELRDAVVNRELSVTAITEYFYHRIES...ICQVGYSFQEHSQIKQLYPKAVNGLFDGGIE*
  [5]   489 MGIAHTEWESVIGLEVHVELNTESKLFSPARNH...GFLVGQIMKRTEGKAPPKRVNELLLAAMRDM*
  ...   ... ...
[893]  1017 MPFSLRSTSFCFLACLCSYSYGFASSPQVLTPN...HHFGRAYMNYSLDARRRQTAHFVSMGLNRIF*
[894]   101 MLATIKKITVLLLSKRKAGIRIDYCALALDAVE...LDASLESAQVRLAGLMLDYWDGDSRLECKKI*
[895]   879 MRPDHMNFCCLCAAILSSTAVLFGQDPLGETAL...LHRLQTLLNVSYVLRGQSHSYSLDLGTTYRF*
[896]    32 MSKKSNNLQTFSSRALFHVFQDEELRKIFGL*
[897]   200 MSIRGVGGNGNSRIPSHNGDGSNRRSQNTKGNN...NLDVNEARLMAAYTSECADHLEANKLAGPDGV
```

Now, we can take advantage of the fact that our training and testing sets are composed of full-length sequences by setting *fullLength* to `0.99` in `IdTaxa`. This will automatically infer the expected length variability among proteins, and filter potential classifications to only those within a reasonable length range. Furthermore, we will lower the *threshold* to 50%, the recommended value for protein sequences.

```
> ids <- IdTaxa(test,
          trainingSet,
          fullLength=0.99,
          threshold=50,
          processors=1)
===============================================================================

Time difference of 3.76 secs
> ids
  A test set of class 'Taxa' with length 897
      confidence taxon
  [1]         8% Root; unclassified_Root
  [2]         0% Root; unclassified_Root
  [3]        99% Root; 6; 3; 5; -; gatC
  [4]       100% Root; 6; 3; 5; 7; gatA
  [5]        99% Root; 6; 3; 5; -; gatB
  ...        ... ...
[893]         5% Root; unclassified_Root
[894]         3% Root; unclassified_Root
[895]         8% Root; unclassified_Root
[896]         0% Root; unclassified_Root
[897]         8% Root; unclassified_Root
```

Since only about a third of the proteins are classifiable in this dataset, we can display the subset of genes that did not belong to "unclassified_Root". To make the plot more interesting, we will subset to the first EC number.
We see that most genes either are not placed in a class with an EC number or belong to EC 2 (Transferases) or EC 3 (Hydrolases).

```
> unclassified <- sapply(ids,
         function(x)
                 "unclassified_Root" %in% x$taxon)
> plot(ids[!unclassified, 1:2])
```

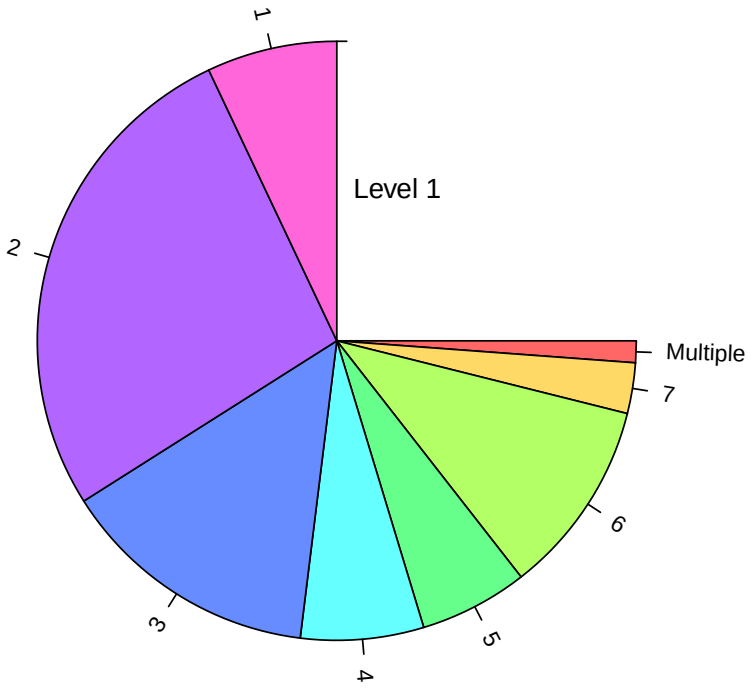


Figure 4: Names of genes in the *E. coli* genome.

7 Session Information

All of the output in this vignette was produced under the following conditions:

- R version 4.3.1 (2023-06-16 ucrt), `x86_64-w64-mingw32`
- Running under: `Windows Server 2022 x64 (build 20348)`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.48.0, Biostrings 2.70.0, DECIPHER 2.30.0, GenomeInfoDb 1.38.0, IRanges 2.36.0, RSQLite 2.3.1, S4Vectors 0.40.0, XVector 0.42.0
- Loaded via a namespace (and not attached): DBI 1.1.3, GenomeInfoDbData 1.2.11, RCurl 1.98-1.12, bit 4.0.5, bit64 4.0.5, bitops 1.0-7, blob 1.2.4, cachem 1.0.8, cli 3.6.1, compiler 4.3.1, crayon 1.5.2, fastmap 1.1.1, memoise 2.0.1, pkgconfig 2.0.3, rlang 1.1.1, tools 4.3.1, vctrs 0.6.4, zlibbioc 1.48.0