# Classify Sequences

Erik S. Wright

January 4, 2019

## Contents



# 1 Introduction

This document describes how to perform taxonomic classification of nucleotide sequences with the DECIPHER package using the *IDTAXA* algorithm. The 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 gene 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 create the training object.

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

## 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 the 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. 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>>"
> # read the sequences into memory
> seqs <- readDNAStringSet(seqs_path)
> # NOTE: use readRNAStringSet for RNA sequences
>
> # (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 taxonomic 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 in red show putative "problem groups" that persist after the iterative removal of "problem sequences" (see above). These red 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, which 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.

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 nucleotide sequence data.

```
> plot(trainingSet)
```

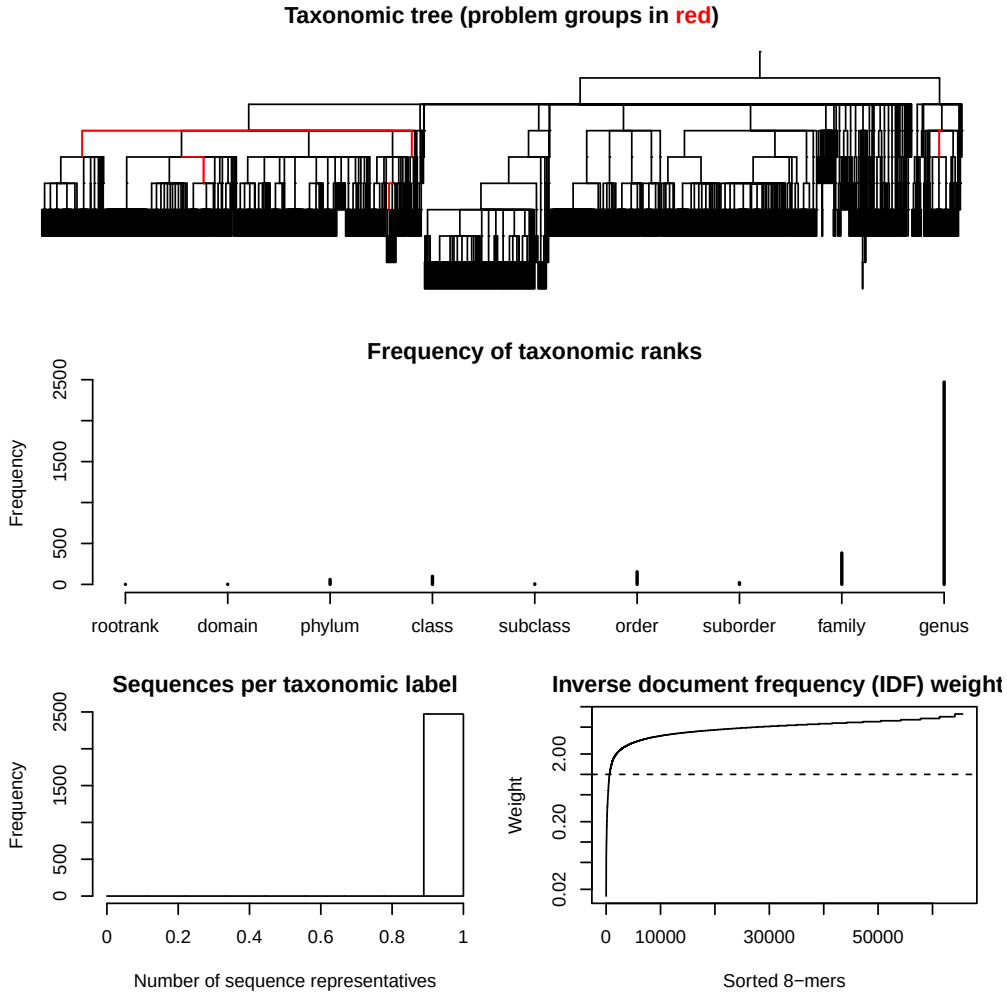


Figure 1: Result of plotting the training set (`trainingSet`) produced by `LearnTaxa`.

# 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)
> # NOTE: use readRNAStringSet for RNA sequences
```

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
  A DNAStringSet instance 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 about 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
```

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]      61.6% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Ba...
  [2]      67.5% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Ba...
  [3]      63.3% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Ba...
  [4]      93.6% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; La...
  [5]      62.4% uncultured bacter... Root; Bacteria; Firmicutes; Clostridia;...
  ...        ... ...                  ...
[171]      38.5% uncultured bacter... Root; unclassified_Root
[172]      48.6% uncultured bacter... Root; unclassified_Root
[173]      31.7% uncultured bacter... Root; unclassified_Root
[174]      49.6% uncultured bacter... Root; unclassified_Root
[175]      53.9% 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]      61.6% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Baci...
[2]      67.5% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Baci...
[3]      63.3% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Baci...
[4]      93.6% uncultured bacter... Root; Bacteria; Firmicutes; Bacilli; Lact...
[5]      62.4% uncultured bacter... Root; Bacteria; Firmicutes; Clostridia; C...
> ids[[1]] # results for the first sequence
$taxon
[1] "Root"           "Bacteria"       "Firmicutes"     "Bacilli"
[5] "Bacillales"     "Planococcaceae" "Lysinibacillus"

$confidence
[1] 75.33956 75.33956 75.33956 75.33956 75.33956 74.82807 61.64954

$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]      48.6% uncultured bacter... Root; unclassified_Root
[2]      34.1% 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]      48.6% uncultured bacter... Root; unclassified_Root
[2]      34.1% 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;Lysinibacillus"
                                                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 to a varying number of sequences, e.g., when only unique sequences were originally classified.

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

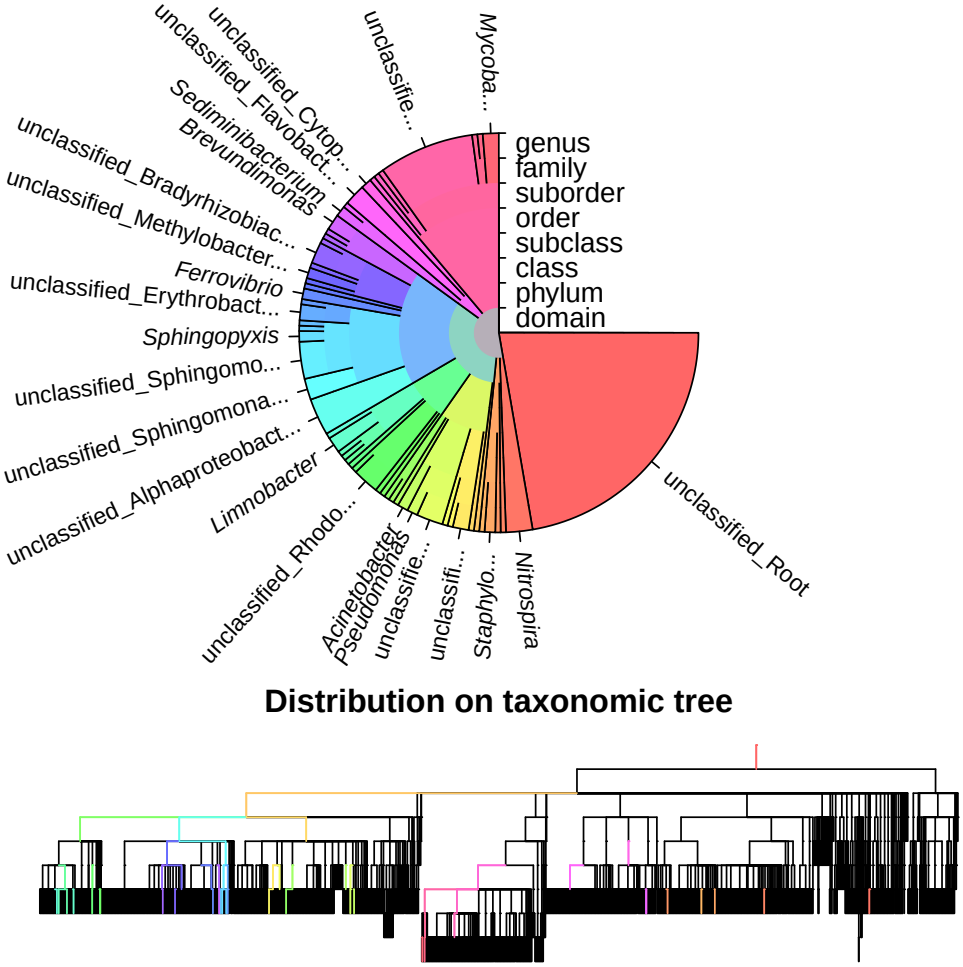


Figure 2: Result of plotting the classifications (`ids`) made by `IdTaxa`.

### 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                9               78                5
     Nitrospirae          unknown
               5               52
> taxon <- sapply(ids,
         function(x)
                 x$taxon[length(x$taxon)])
> head(taxon)
 uncultured bacterium; Pro_CL-05069_OTU-15.
                           "Lysinibacillus"
 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
  Acinetobacter  0.15384615 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
                samples
taxon                    Mar        Pro        UWH        UWL
  Achromobacter  0.00000000 0.00000000 0.00000000 0.00000000
  Acidovorax     0.00000000 0.00000000 0.00000000 0.00000000
  Acinetobacter  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
```

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))
```

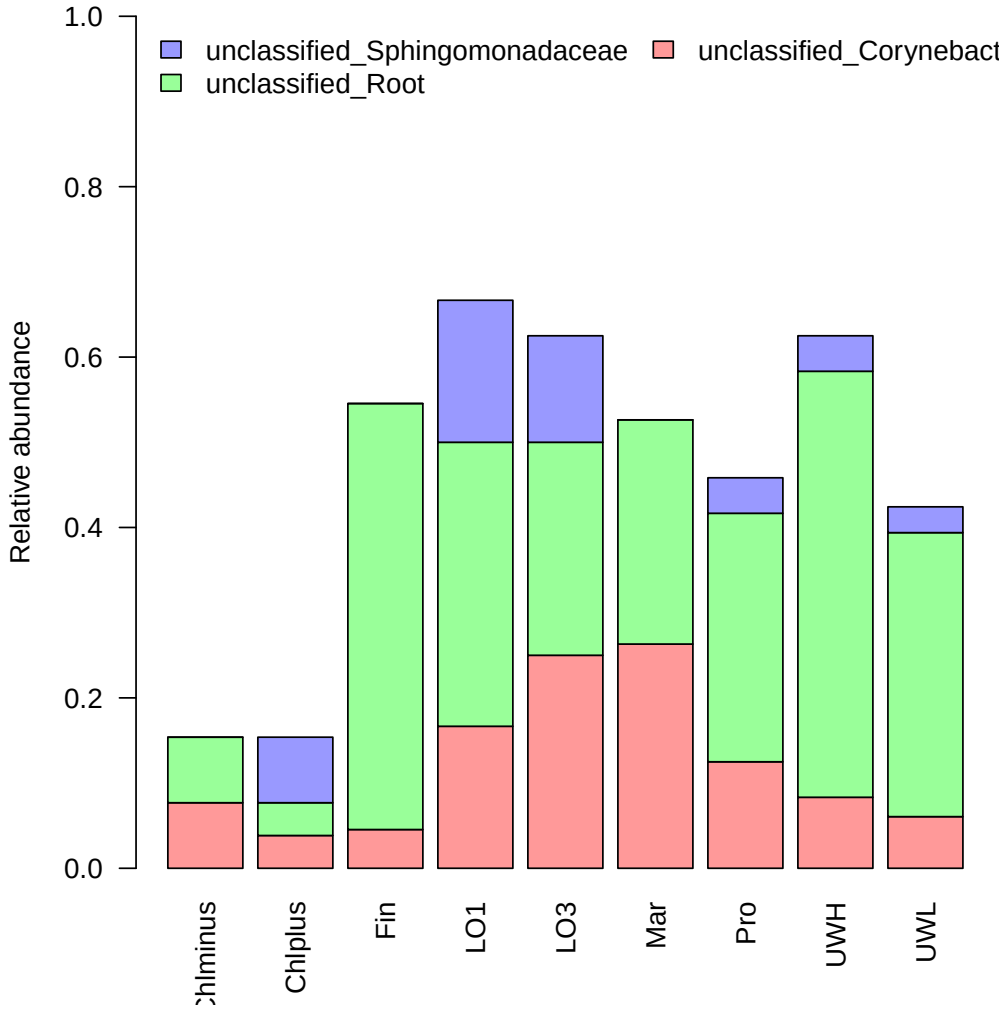


Figure 3: Barplot of taxonomic assignments by sample.

### 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 (44.9%); unclassified_Root (44.9%)"
 uncultured bacterium; UWH_CL-01079_OTU-21.
  "Root (38.5%); unclassified_Root (38.5%)"
  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 (49.6%); unclassified_Root (49.6%)"
uncultured bacterium; UWL_CL-110548_OTU-32.
  "Root (53.9%); unclassified_Root (53.9%)"
> #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 Session Information

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

- R version 3.5.2 (2018-12-20), `x86_64-pc-linux-gnu`

- Running under: `Ubuntu 16.04.5 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.8-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.8-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.28.0, Biostrings 2.50.2, DECIPHER 2.10.1, IRanges 2.16.0, RSQLite 2.1.1, S4Vectors 0.20.1, XVector 0.22.0
- Loaded via a namespace (and not attached): DBI 1.0.0, Rcpp 1.0.0, bit 1.1-14, bit64 0.9-7, blob 1.1.1, compiler 3.5.2, digest 0.6.18, memoise 1.1.0, pkgconfig 2.0.2, tools 3.5.2, zlibbioc 1.28.0