

# Treemendous: Example

Andrea Paz, Felix Specker, Dan Maynard

2023-07-17

## Package Installation

```
library(devtools)
install_github("speckerf/treemendous")
```

## Species List Preparation

All functions of *Treemendous* require the species name to be split into two columns, **Genus** and **Species**, with the former being capitalized. Assume you have two species, *Acer platanoides* and *Fagus sylvatica*, you can create the input tibble by calling:

```
### Species list preparation
library(tidyverse)
species <- c('Acer platanoides', 'Fagus sylvatica')
input <- species %>%
  tibble::as_tibble_col(column_name = 'binomial') %>%
  tidyr::separate(col = 'binomial', into = c('Genus', 'Species'))
input
```

```
## # A tibble: 2 x 2
##   Genus Species
##   <chr> <chr>
## 1 Acer  platanoides
## 2 Fagus sylvatica
```

Other useful functions for creating the input tibble include:

```
readr::read_csv('path') # import data
dplyr::select(Genus, Species) # select columns
dplyr::distinct(Genus, Species) # remove duplicate binomials
dplyr::rename('Genus' = 'old_genus_name',
              'Species' = 'old_species_name') # rename columns
dplyr::mutate(Genus = stringr::str_to_title(Genus)) # capitalize Genus
dplyr::mutate(Species = stringr::str_remove(Species, ".*?\\s")) # remove everything before first space
tidyr::drop_na(c('Genus', 'Species')) # remove rows with NA's
dplyr::arrange(Genus, Species) # sort names
dplyr::bind_rows(x, y) # concatenate two tibble's
```

## FIA: Standardize species names from the U.S. Forest Inventory and Analysis program.

Along with the package comes an example dataset `fia` with 2171 different tree species names. Assume that we want to standardize these species names according to a certain backbone (use the `backbone` argument). The function `summarize_output()` can be used to get a summary of the process.

```
library(treemendous)

result <- fia %>% matching(backbone = 'BGCI')
summarize_output(result)

## [1] "matched: 1822 / 2171 were matched with 1822 distinct matched names."
## [2] "direct_match: 1779 / 2171"
## [3] "indirectly matched: 43 / 392"
## [4] "      genus_match: 313 / 392"
## [5] "      fuzzy_match_genus: 2 / 79"
## [6] "      direct_match_species_within_genus: 1 / 315"
## [7] "      suffix_match_species_within_genus: 11 / 314"
## [8] "      fuzzy_match_species_within_genus: 31 / 303"
```

From 2171 species names in total, we were able to match 1822 according to the backbone BGCI, with 1779 names matching exactly, and 43 species names matching using fuzzy- and suffix-matching. Besides information about the matching process, the output contains the old names (prefix `Orig.`) as well as the matched names (prefix `Matched.`) as follows:

```
result %>%
  dplyr::slice_head(n=3) %>%
  dplyr::select(1:5)

## # A tibble: 3 x 5
##   Orig.Genus Orig.Species Matched.Genus Matched.Species matched
##   <chr>      <chr>      <chr>      <chr>      <lgl>
## 1 Abies      amabilis    Abies      amabilis    TRUE
## 2 Abies      balsamea    Abies      balsamea    TRUE
## 3 Abies      bracteata   Abies      bracteata   TRUE
```

We can further increase the number of matched species by using the functions `matching()` followed by `enforce_matching()`. Here, we specify the backbone BGCI.

```
result <- fia %>%
  matching(backbone = 'BGCI') %>%
  enforce_matching(backbone = 'BGCI')
result %>% summarize_output()

## [1] "matched: 2097 / 2171 were matched with 2036 distinct matched names."
## [2] "direct_match: 1779 / 2171"
## [3] "indirectly matched: 43 / 392"
## [4] "      genus_match: 93 / 117"
## [5] "      fuzzy_match_genus: 2 / 24"
## [6] "      direct_match_species_within_genus: 1 / 95"
```

```
## [7] "      suffix_match_species_within_genus: 11 / 94"
## [8] "      fuzzy_match_species_within_genus: 31 / 83"
## [9] "number of species matched via enforce_matching(): 275 / 349"
```

Now, we are able to match 2097 species names in total, with 275 species being matched via `enforce_matching()`. Note that the number of matched distinct species names is lower with 2044, because several input species were matched to the same species in the target database BGCI.

Note that if we choose a different backbone than BGCI, then species can matched names that are not accepted (synonyms), we can further resolve synonyms after matching the species names with the function `resolve_synonyms()`. Now, the output contains additionally the accepted species names (prefix `Accepted.`), as well as a column `Accepted.Backbone`, which states according to which backbone the synonym was resolved.

```
result <- fia %>%
  matching('WFO') %>%
  resolve_synonyms('WFO')
```

```
result %>%
  dplyr::slice_head(n=3) %>%
  dplyr::select(dplyr::matches('Orig|Matched|Accepted'), -'matched')
```

```
## # A tibble: 3 x 7
##   Orig.Genus Orig.Species Matched.Genus Matched.Species Accepted.Genus
##   <chr>      <chr>      <chr>      <chr>      <chr>
## 1 Abies      amabilis    Abies      amabilis    Abies
## 2 Abies      balsamea    Abies      balsamea    Abies
## 3 Abies      bracteata   Abies      bracteata   Abies
## # i 2 more variables: Accepted.Species <chr>, Accepted.Backbone <chr>
```

Note that a warning message is produced “Please consider calling `highlight_flags()` to investigate potential ambiguities upon resolving synonyms to accepted names”. Potential ambiguities could have been resolved in your dataset and it is suggested to use `highlight_flags()` to know more and decide if you want to check them manually. The `highlight_flags()` function should be used separately from the others as it will only return species that have some flag and not the full dataset.

```
flags <- result %>% highlight_flags('WFO')
```

## In summary, 574 out of 2171 matched species have raised a flag.

```
flags %>%
  dplyr::slice_head(n=3) %>%
  dplyr::select(dplyr::matches('Acc|ambiguity|link'))
```

```
## # A tibble: 3 x 6
##   Accepted.Genus Accepted.Species Accepted.Backbone WFO_authorship_ambiguity
##   <chr>          <chr>          <chr>          <lgl>
## 1 Abies          amabilis        WFO            TRUE
## 2 Abies          balsamea        WFO            FALSE
## 3 Abies          concolor        WFO            FALSE
## # i 2 more variables: WFO_infraspecific_ambiguity <lgl>,
## #   WFO_infraspecific_link <lgl>
```

Instead of using a single backbone, the user can decide to use any subset of the backbones `c('BGCI', 'WFO', 'WCVP', 'GBIF')` or use all of them by simply calling `matching()` without any argument. While `matching()` considers all backbones being equally important, the function `sequential_matching()` can be used to call `matching()` for individual backbones sequentially. For every species, the matched backbone is provided in the column `Matched.Backbone`.

```
result <- fia %>%
  sequential_matching(sequential_backbones = c('BGCI', 'WFO', 'WCVP'))
```

Remember that `matching()` and `sequential_matching()` match any species in the database and thus can provide matches to synonyms rather than accepted species. To get only accepted species returned use `resolve_synonyms()` after the matching function.

## Translate species names between two databases.

Oftentimes, researches require integrating multi-modal data from different sources for their analyses. Here, we demonstrate the use of the function `translate_trees()`, which allows a user directly translate names from an input database to a target database. First, we resolve both databases individually according to the single backbone (WFO) and compare the resolved names. Then, we use `translate_trees` to translate the input species names into the target names.

```
input <- tibble::tibble(
  Genus = c('Aria', 'Ardisia', 'Malus'),
  Species = c('umbellata', 'japonica', 'sylvestris')
)
target <- tibble::tibble(
  Genus = c('Sorbus', 'Ardisia', 'Malus'),
  Species = c('umbellata', 'montana', 'orientalis')
)
```

```
input %>%
  matching(backbone = 'WFO') %>%
  resolve_synonyms('WFO') %>%
  dplyr::select(1:6)
```

```
## # A tibble: 3 x 6
##   Orig.Genus Orig.Species Matched.Genus Matched.Species Accepted.Genus
##   <chr>      <chr>      <chr>      <chr>      <chr>
## 1 Ardisia   japonica   Ardisia   japonica   Ardisia
## 2 Aria      umbellata  Aria      umbellata  Aria
## 3 Malus     sylvestris Malus     sylvestris Malus
## # i 1 more variable: Accepted.Species <chr>
```

```
target %>%
  matching(backbone = 'WFO') %>%
  resolve_synonyms('WFO') %>%
  dplyr::select(1:6)
```

```
## # A tibble: 3 x 6
##   Orig.Genus Orig.Species Matched.Genus Matched.Species Accepted.Genus
##   <chr>      <chr>      <chr>      <chr>      <chr>
## 1 Ardisia   montana    Ardisia   montana    Ardisia
## 2 Malus     orientalis Malus     <NA>       <NA>
```

```
## 3 Sorbus      umbellata      Sorbus      umbellata      Sorbus
## # i 1 more variable: Accepted.Species <chr>
```

Resolving both sets individually leads to a mismatch - *Malus orientalis* and *Malus sylvestris* were resolved to two different names. Now let's see whether `translate_trees` can be used to match all three species:

```
translate_trees(df = input, target = target) %>%
  dplyr::select(1:4)
```

```
## # A tibble: 3 x 4
##   Orig.Genus Orig.Species Matched.Genus Matched.Species
##   <chr>      <chr>      <chr>      <chr>
## 1 Ardisia    japonica    Ardisia    montana
## 2 Aria      umbellata   Sorbus     umbellata
## 3 Malus     sylvestris  Malus      orientalis
```

Essentially, all three species names can be translated from the input set to the target set. Incorporating the knowledge of the desired target names, the function leverages the information about synonym-accepted relations in the three backbones WFO, WCVF and GBIF and is able to translate *Malus sylvestris* into *Malus orientalis*.