Package: multilink (via r-universe)

October 24, 2024

Title Multifile Record Linkage and Duplicate Detection

Version 0.1.1

Description Implementation of the methodology of Aleshin-Guendel & Sadinle (2022) <doi:10.1080/01621459.2021.2013242>. It handles the general problem of multifile record linkage and duplicate detection, where any number of files are to be linked, and any of the files may have duplicates.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

URL https://github.com/aleshing/multilink

BugReports https://github.com/aleshing/multilink/issues

Imports igraph, RecordLinkage, Rcpp, utils, mcclust, geosphere, stringr

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Author Serge Aleshin-Guendel [aut, cre]

Maintainer Serge Aleshin-Guendel <saleshinguendel@gmail.com>

Repository CRAN

Date/Publication 2023-06-09 14:20:07 UTC

Contents

`create_comparison_data` *Create Comparison Data*

Description

Create comparison data for all pairs of records, except for those records in files which are assumed to have no duplicates.

Usage

```
create_comparison_data(
  records,
  types,
  breaks,
  file_sizes,
  duplicates,
  verbose = TRUE
)
```

Arguments

`records`	A `data.frame` containing the records to be linked, where each column of `records` is a field to be compared. If there are multiple files, `records` should be obtained by stacking the files on top of each other so that `records[1:file_sizes[1], ]` contains the records for file 1, `records[(file_sizes[1] + 1):(file_sizes[1] + file_sizes[2]), ]` contains the records for file 2, and so on. Missing values should be coded as `NA`.
`types`	A `character` vector, indicating the comparison to be used for each field (i.e. each column of `records`). The options are: `"bi"` for binary comparisons, `"nu"` for numeric comparisons (absolute difference), `"lv"` for string comparisons (normalized Levenshtein distance), `"lv_sep"` for string comparisons (normalized Levenshtein distance) where each string may contain multiple spellings separated by the "|" character. We assume that fields using options `"bi"`, `"lv"`, and `"lv_sep"` are of class `character`, and fields using the `"nu"` option are of class `numeric`. For fields using the `"lv_sep"` option, for each record pair the normalized Levenshtein distance is computed between each possible spelling, and the minimum normalized Levenshtein distance between spellings is then used as the comparison for that record pair.

breaks	A list, the same length as types, indicating the break points used to compute disagreement levels for each fields' comparisons. If types[f]="bi", breaks[[f]] is ignored (and thus can be set to NA). See Details for more information on specifying this argument.
file_sizes	A numeric vector indicating the size of each file.
duplicates	A numeric vector indicating which files are assumed to have duplicates. duplicates[k] should be 1 if file k has duplicates, and duplicates[k] should be 0 if file k has no duplicates. If any files do not have duplicates, we strongly recommend that the largest such file is organized to be the first file.
verbose	A logical indicator of whether progress messages should be print (default TRUE).

Details

The purpose of this function is to construct comparison vectors for each pair of records. In order to construct these vectors, one needs to specify the types and breaks arguments. The types argument specifies how each field should be compared, and the breaks argument specifies how to discretize these comparisons.

Currently, the types argument supports three types of field comparisons: binary, absolute difference, and the normalized Levenshtein distance. Please contact the package maintainer if you need a new type of comparison to be supported.

The breaks argument should be a list, with with one element for each field. If a field is being compared with a binary comparison, i.e. types[f]="bi", then the corresponding element of breaks should be NA, i.e. breaks[[f]]=NA. If a field is being compared with a numeric or string comparison, then the corresponding element of breaks should be a vector of cut points used to discretize the comparisons. To give more detail, suppose you pass in cut points breaks[[f]]=c(cut_1, ...,cut_L). These cut points discretize the range of the comparisons into L+1 intervals: $I_0 = (-\infty, cut_1], I_1 = (cut_1, cut_2], ..., I_L = (cut_L, \infty]$. The raw comparisons, which lie in $[0, \infty)$ for numeric comparisons and $[0, 1]$ for string comparisons, are then replaced with indicators of which interval the comparisons lie in. The interval I_0 corresponds to the lowest level of disagreement for a comparison, while the interval I_L corresponds to the highest level of disagreement for a comparison.

Value

a list containing:

record_pairs A data.frame, where each row contains the pair of records being compared in the corresponding row of comparisons. The rows are sorted in ascending order according to the first column, with ties broken according to the second column in ascending order. For any given row, the first column is less than the second column, i.e. record_pairs[i, 1] < record_pairs[i, 2] for each row i.

comparisons A logical matrix, where each row contains the comparisons for the record pair in the corresponding row of record_pairs. Comparisons are in the same order as the columns of records, and are represented by L + 1 columns of TRUE/FALSE indicators, where L + 1 is the number of disagreement levels for the field based on breaks.

K The number of files, assumed to be of class numeric.

file_sizes A numeric vector of length K, indicating the size of each file.

`duplicates` A `numeric` vector of length `K`, indicating which files are assumed to have duplicates. `duplicates[k]` should be `1` if file `k` has duplicates, and `duplicates[k]` should be `0` if file `k` has no duplicates. If any files do not have duplicates, we strongly recommend that the largest such file is organized to be the first file.

`field_levels` A `numeric` vector indicating the number of disagreement levels for each field.

`file_labels` An `integer` vector of length `sum(file_sizes)`, where `file_labels[i]` indicates which file record `i` is in.

`fp_matrix` An `integer` matrix, where `fp_matrix[k1, k2]` is a label for the file pair `(k1, k2)`. Note that `fp_matrix[k1, k2] = fp_matrix[k2, k1]`.

`rp_to_fp` A `logical` matrix that indicates which record pairs belong to which file pairs. `rp_to_fp[fp, rp]` is `TRUE` if the records `record_pairs[rp, ]` belong to the file pair `fp`, and is FALSE otherwise. Note that `fp` is given by the labeling in `fp_matrix`.

`ab` An `integer` vector, of length `ncol(comparisons) * K * (K + 1) / 2` that indicates how many record pairs there are with a given disagreement level for a given field, for each file pair.

`file_sizes_not_included` A `numeric` vector of `0`s. This element is non-zero when `reduce_comparison_data` is used.

`ab_not_included` A `numeric` vector of `0`s. This element is non-zero when `reduce_comparison_data` is used.

`labels` `NA`. This element is not `NA` when `reduce_comparison_data` is used.

`pairs_to_keep` `NA`. This element is not `NA` when `reduce_comparison_data` is used.

`cc` `0`. This element is non-zero when `reduce_comparison_data` is used.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242][arXiv]

Examples

```
## Example with small no duplicate dataset
data(no_dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(no_dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = no_dup_data_small$file_sizes,
 duplicates = c(0, 0, 0))

## Example with small duplicate dataset
data(dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
```

```
                c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
file_sizes = dup_data_small$file_sizes,
duplicates = c(1, 1, 1))
```

---

dup_data *Duplicate Dataset*

Description

A dataset containing 867 simulated records from 3 files with no duplicate records in each file.

Usage

```
dup_data
```

Format

A list with three elements:

records A `data.frame` with the records, containing 7 fields, from all three files, in the format used for input to `create_comparison_data`.

file_sizes The size of each file.

IDs The true partition of the records, represented as an `integer` vector of arbitrary labels of length `sum(file_sizes)`.

Source

Extracted from the datasets used in the simulation study of the paper. The datasets were generated using code from Peter Christen's group https://dmm.anu.edu.au/geco/index.php.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242][arXiv]

Examples

```
data(dup_data)

# There are 500 entities represented in the records
length(unique(dup_data$IDs))
```

dup_data_small *Small Duplicate Dataset*

Description

A dataset containing 96 simulated records from 3 files with no duplicate records in each file, subset from dup_data.

Usage

```
dup_data_small
```

Format

A list with three elements:

records A data.frame with the records, containing 7 fields, from all three files, in the format used for input to create_comparison_data.

file_sizes The size of each file.

IDs The true partition of the records, represented as an integer vector of arbitrary labels of length sum(file_sizes).

Source

Extracted from the datasets used in the simulation study of the paper. The datasets were generated using code from Peter Christen's group https://dmm.anu.edu.au/geco/index.php.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242][arXiv]

Examples

```
data(dup_data_small)

# There are 96 entities represented in the records
length(unique(dup_data_small$IDs))
```

`find_bayes_estimate` *Find the Bayes Estimate of a Partition*

Description

Find the (approximate) Bayes estimate of a partition based on MCMC samples of the partition and a specified loss function.

Usage

```
find_bayes_estimate(
  partitions,
  burn_in,
  L_FNM = 1,
  L_FM1 = 1,
  L_FM2 = 2,
  L_A = Inf,
  max_cc_size = nrow(partitions),
  verbose = TRUE
)
```

Arguments

`partitions`	Posterior samples of the partition, where each column is one sample and the partition is represented as an `integer` vector of arbitrary labels, as produced by the output of a call to `gibbs_sampler`.
`burn_in`	The number of samples to discard for burn in.
`L_FNM`	Positive loss for a false non-match. Default is `1`.
`L_FM1`	Positive loss for a type 1 false match. Default is `1`.
`L_FM2`	Positive loss for a type 2 false match. Default is `2`.
`L_A`	Positive loss for abstaining from making a decision for a record. Default is `Inf`, i.e. decisions are made for all records.
`max_cc_size`	The maximum allowable connected component size over which the posterior expected loss is minimized. Default is `nrow(partitions)`, i.e. no approximation is used. When `is.infinite(L_A)`, we recommend setting this argument to `50`, then increasing based on a computational budget. When `!is.infinite(L_A)`, we recommend setting this argument to `10-12`, then increasing based on a computational budget (although an increase of `1` in this argument can in the worst case lead to a doubling in computation time).
`verbose`	A `logical` indicator of whether progress messages should be print (default `TRUE`).

Value

A vector, the same length of a column of `partitions` containing the (approximate) Bayes estimate of the partition. If `!is.infinite(L_A)` the output may be a partial estimate. A positive number `l` in index `i` indicates that record `i` is in the same cluster as every other record `j` with `l` in index `j`. A value of `-1` in index `i` indicates that the Bayes estimate abstained from making a decision for record `i`.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242][arXiv]

Examples

```
# Example with small no duplicate dataset
data(no_dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(no_dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = no_dup_data_small$file_sizes,
 duplicates = c(0, 0, 0))

# Specify the prior
prior_list <- specify_prior(comparison_list, mus = NA, nus = NA, flat = 0,
 alphas = rep(1, 7), dup_upper_bound = c(1, 1, 1),
 dup_count_prior_family = NA, dup_count_prior_pars = NA,
 n_prior_family = "uniform", n_prior_pars = NA)

# Find initialization for the matching (this step is optional)
# The following line corresponds to only keeping pairs of records as
# potential matches in the initialization for which neither gname nor fname
# disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
Z_init <- initialize_partition(comparison_list, pairs_to_keep, seed = 42)

# Run the Gibbs sampler
results <- gibbs_sampler(comparison_list, prior_list, n_iter = 1000,
 Z_init = Z_init, seed = 42)

# Find the full Bayes estimate

full_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = Inf, max_cc_size = 50)

# Find the partial Bayes estimate
partial_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = 0.1, max_cc_size = 12)
```

```
# Example with small duplicate dataset
data(dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = dup_data_small$file_sizes,
 duplicates = c(1, 1, 1))

# Reduce the comparison data
# The following line corresponds to only keeping pairs of records for which
# neither gname nor fname disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
reduced_comparison_list <- reduce_comparison_data(comparison_list,
 pairs_to_keep, cc = 1)

# Specify the prior
prior_list <- specify_prior(reduced_comparison_list, mus = NA, nus = NA,
 flat = 0, alphas = rep(1, 7), dup_upper_bound = c(10, 10, 10),
 dup_count_prior_family = c("Poisson", "Poisson", "Poisson"),
 dup_count_prior_pars = list(c(1), c(1), c(1)), n_prior_family = "uniform",
 n_prior_pars = NA)

# Run the Gibbs sampler
results <- gibbs_sampler(reduced_comparison_list, prior_list, n_iter = 1000,
 seed = 42)

# Find the full Bayes estimate

full_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = Inf, max_cc_size = 50)

# Find the partial Bayes estimate
partial_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = 0.1, max_cc_size = 12)
```

`gibbs_sampler` *Gibbs Sampler for Posterior Inference*

Description

Run a Gibbs sampler to explore the posterior distribution of partitions of records.

Usage

```
gibbs_sampler(
  comparison_list,
  prior_list,
  n_iter = 2000,
  Z_init = 1:sum(comparison_list$file_sizes),
  seed = 70,
  single_likelihood = FALSE,
  chaperones_info = NA,
  verbose = TRUE
)
```

Arguments

comparison_list	The output from a call to create_comparison_data or reduce_comparison_data.
prior_list	The output from a call to specify_prior.
n_iter	The number of iterations of the Gibbs sampler to run.
Z_init	Initialization of the partition of records, represented as an integer vector of arbitrary labels of length sum(comparison_list$file_sizes). The default initialization places each record in its own cluster. See initialize_partition for an alternative initialization when there are no duplicates in each file.
seed	The seed to use while running the Gibbs sampler.
single_likelihood	A logical indicator of whether to use a single likelihood for comparisons for all file pairs, or whether to use a separate likelihood for comparisons for each file pair. When single_likelihood=TRUE, a single likelihood is used, and the prior hyperparameters for m and u from the first file pair are used. We do not recommend using a single likelihood in general.
chaperones_info	If chaperones_info is set to NA, then Gibbs updates to the partition are used during the Gibbs sampler, as described in Aleshin-Guendel & Sadinle (2022). Else, Chaperones updates, as described in Miller et al. (2015) and Betancourt et al. (2016), are used and chaperones_info should be a list with five elements controlling Chaperones updates to the partition during the Gibbs sampler: chap_type, num_chap_iter, nonuniform_chap_type, extra_gibbs, num_restrict. chap_type is 0 if using a uniform Chaperones distribution, and 1 if using a nonuniform Chaperones distribution. num_chap_iter is the number of Chaperones updates to the partition that are made during each iteration of the Gibbs sampler. When using a nonuniform Chaperones distribution, nonuniform_chap_type is 0 if using the exact version, or 1 if using the partial version. extra_gibbs is a logical indicator of whether a Gibbs update to the partition should be done after the Chaperones updates, at each iteration of the Gibbs sampler. num_restrict is the number of restricted Gibbs steps to take during each Chaperones update to the partition.
verbose	A logical indicator of whether progress messages should be print (default TRUE).

Details

Given the prior specified using specify_prior, this function runs a Gibbs sampler to explore the posterior distribution of partitions of records, conditional on the comparison data created using create_comparison_data or reduce_comparison_data.

Value

a list containing:

`m` Posterior samples of the `m` parameters. Each column is one sample.

`u` Posterior samples of the `u` parameters. Each column is one sample.

`partitions` Posterior samples of the partition. Each column is one sample. Note that the partition is represented as an `integer` vector of arbitrary labels of length `sum(comparison_list$file_sizes)`.

`contingency_tables` Posterior samples of the overlap table. Each column is one sample. This incorporates counts of records determined not to be candidate matches to any other records using reduce_comparison_data.

`cluster_sizes` Posterior samples of the size of each cluster (associated with an arbitrary label from 1 to `sum(comparison_list$file_sizes)`). Each column is one sample.

`sampling_time` The time in seconds it took to run the sampler.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242][arXiv]

Jeffrey Miller, Brenda Betancourt, Abbas Zaidi, Hanna Wallach, & Rebecca C. Steorts (2015). Microclustering: When the cluster sizes grow sublinearly with the size of the data set. *NeurIPS Bayesian Nonparametrics: The Next Generation Workshop Series*. [arXiv]

Brenda Betancourt, Giacomo Zanella, Jeffrey Miller, Hanna Wallach, Abbas Zaidi, & Rebecca C. Steorts (2016). Flexible Models for Microclustering with Application to Entity Resolution. *Advances in neural information processing systems*. [Published] [arXiv]

Examples

```
# Example with small no duplicate dataset
data(no_dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(no_dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = no_dup_data_small$file_sizes,
 duplicates = c(0, 0, 0))

# Specify the prior
prior_list <- specify_prior(comparison_list, mus = NA, nus = NA, flat = 0,
 alphas = rep(1, 7), dup_upper_bound = c(1, 1, 1),
 dup_count_prior_family = NA, dup_count_prior_pars = NA,
```

```
 n_prior_family = "uniform", n_prior_pars = NA)

# Find initialization for the matching (this step is optional)
# The following line corresponds to only keeping pairs of records as
# potential matches in the initialization for which neither gname nor fname
# disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
Z_init <- initialize_partition(comparison_list, pairs_to_keep, seed = 42)

# Run the Gibbs sampler
{
results <- gibbs_sampler(comparison_list, prior_list, n_iter = 1000,
 Z_init = Z_init, seed = 42)
}

# Example with small duplicate dataset
data(dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = dup_data_small$file_sizes,
 duplicates = c(1, 1, 1))

# Reduce the comparison data
# The following line corresponds to only keeping pairs of records for which
# neither gname nor fname disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
reduced_comparison_list <- reduce_comparison_data(comparison_list,
 pairs_to_keep, cc = 1)

# Specify the prior
prior_list <- specify_prior(reduced_comparison_list, mus = NA, nus = NA,
 flat = 0, alphas = rep(1, 7), dup_upper_bound = c(10, 10, 10),
 dup_count_prior_family = c("Poisson", "Poisson", "Poisson"),
 dup_count_prior_pars = list(c(1), c(1), c(1)), n_prior_family = "uniform",
 n_prior_pars = NA)

# Run the Gibbs sampler
{
results <- gibbs_sampler(reduced_comparison_list, prior_list, n_iter = 1000,
 seed = 42)
}
```

`initialize_partition` *Initialize the Partition*

Description

Generate an initialization for the partition in the case when it is assumed there are no duplicates in all files (so that the partition is a matching).

Usage

```
initialize_partition(comparison_list, pairs_to_keep, seed = NA)
```

Arguments

comparison_list	the output from a call to create_comparison_data or reduce_comparison_data. Note that in order to correctly specify the initialization, if reduce_comparison_data is used to the reduce the number of record pairs that are candidate matches, then the output of reduce_comparison_data (not create_comparison_data) should be used for this argument.
pairs_to_keep	A logical vector, the same length as comparison_list$record_pairs, indicating which record pairs are potential matches in the initialization.
seed	The seed to use to generate the initialization.

Details

When it is assumed that there are no duplicates in all files, and reduce_comparison_data is not used to reduce the number of potential matches, the Gibbs sampler used for posterior inference may experience slow mixing when using an initialization for the partition where each record is in its own cluster (the default option for the Gibbs sampler). The purpose of this function is to provide an alternative initialization scheme.

To use this initialization scheme, the user passes in a logical vector that indicates which record pairs are potential matches according to an indexing method (as in reduce_comparison_data). Note that this indexing is only used to generate the initialization, it is not used for inference. The initialization scheme first finds the transitive closure of the potential matches, which partitions the records into blocks. Within each block of records, the scheme randomly selects a record from each file, and these selected records are then placed in the same cluster for the partition initialization. All other records are placed in their own clusters.

Value

an integer vector of arbitrary labels of length sum(comparison_list$file_sizes), giving an initialization for the partition.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242][arXiv]

Examples

```
# Example with small no duplicate dataset
data(no_dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(no_dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = no_dup_data_small$file_sizes,
 duplicates = c(0, 0, 0))

# Find initialization for the matching
# The following line corresponds to only keeping pairs of records as
# potential matches in the initialization for which neither gname nor fname
# disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
Z_init <- initialize_partition(comparison_list, pairs_to_keep, seed = 42)
```

`multilink` *Multifile Record Linkage and Duplicate Detection*

Description

The multilink package implements the methodology of Aleshin-Guendel & Sadinle (2022). It handles the general problem of multifile record linkage and duplicate detection, where any number of files are to be linked, and any of the files may have duplicates.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242] [arXiv]

Examples

```
# Here we demonstrate an example workflow with the small no duplicate dataset
data(no_dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(no_dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = no_dup_data_small$file_sizes,
 duplicates = c(0, 0, 0))

# Specify the prior
```

```
prior_list <- specify_prior(comparison_list, mus = NA, nus = NA, flat = 0,
 alphas = rep(1, 7), dup_upper_bound = c(1, 1, 1),
 dup_count_prior_family = NA, dup_count_prior_pars = NA,
 n_prior_family = "uniform", n_prior_pars = NA)

# Find initialization for the matching (this step is optional)
# The following line corresponds to only keeping pairs of records as
# potential matches in the initialization for which neither gname nor fname
# disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
Z_init <- initialize_partition(comparison_list, pairs_to_keep, seed = 42)

# Run the Gibbs sampler
results <- gibbs_sampler(comparison_list, prior_list, n_iter = 1000,
 Z_init = Z_init, seed = 42)

# Find the full Bayes estimate

full_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = Inf, max_cc_size = 50)

# The number of clusters in the full estimate
length(unique(full_estimate))
# The number of entities represented in the records
length(unique(no_dup_data_small$IDs))

# Find which record pairs are truly coreferent based on IDs
true_links <- no_dup_data_small$IDs[comparison_list$record_pairs[, 1]] ==
no_dup_data_small$IDs[comparison_list$record_pairs[, 2]]

# Find which record pairs are in the same clusters in the full estimate
full_estimate_links <- full_estimate[comparison_list$record_pairs[, 1]] ==
full_estimate[comparison_list$record_pairs[, 2]]

# Find the number of true matches in the full estimate
true_matches <- sum(full_estimate_links & true_links)

# Precision of the full estimate
true_matches / sum(full_estimate_links)

# Recall of the full estimate
true_matches / sum(true_links)

# Find the partial Bayes estimate
partial_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = 0.1, max_cc_size = 12)

# The partial estimate abstains from making decisions for how many records?
sum(partial_estimate == -1)

# For the records which decisions were made for in the partial estimate,
# there are how many clusters?
```

```
length(unique(partial_estimate))

# Abstain rate of partial_estimate
sum(partial_estimate == -1) / length(partial_estimate)

# Relabel records where we abstained
partial_estimate[which(partial_estimate == -1)] <- length(partial_estimate) +
which(partial_estimate == -1)

# Find which record pairs are in the same clusters in the full estimate
partial_estimate_links <-
 partial_estimate[comparison_list$record_pairs[, 1]] ==
 partial_estimate[comparison_list$record_pairs[, 2]]

# Find the number of true matches in the partial estimate
true_matches_A <- sum(partial_estimate_links & true_links)

# Precision of the partial estimate
true_matches_A / sum(partial_estimate_links)


# Here we demonstrate an example workflow with the small duplicate dataset
data(dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = dup_data_small$file_sizes,
 duplicates = c(1, 1, 1))

# Reduce the comparison data
# The following line corresponds to only keeping pairs of records for which
# neither gname nor fname disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
reduced_comparison_list <- reduce_comparison_data(comparison_list,
 pairs_to_keep, cc = 1)

# Specify the prior
prior_list <- specify_prior(reduced_comparison_list, mus = NA, nus = NA,
 flat = 0, alphas = rep(1, 7), dup_upper_bound = c(10, 10, 10),
 dup_count_prior_family = c("Poisson", "Poisson", "Poisson"),
 dup_count_prior_pars = list(c(1), c(1), c(1)), n_prior_family = "uniform",
 n_prior_pars = NA)

# Run the Gibbs sampler
results <- gibbs_sampler(reduced_comparison_list, prior_list, n_iter = 1000,
 seed = 42)

# Find the full Bayes estimate
```

```
full_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = Inf, max_cc_size = 50)

# The number of  clusters in the full estimate (including records records
# determined not to be candidate matches to any other records using
# reduce_comparison_data)
length(unique(full_estimate)) +
sum(reduced_comparison_list$file_sizes_not_included)
# The number of entities represented in the records
length(unique(dup_data_small$IDs))

# Find which record pairs are truly coreferent based on IDs
true_links <- dup_data_small$IDs[comparison_list$record_pairs[, 1]] ==
dup_data_small$IDs[comparison_list$record_pairs[, 2]]

# Focus on the record pairs that were candidate matches
true_links_reduced <- true_links[reduced_comparison_list$pairs_to_keep]

# Calculate the number of prior false non-matches based on the indexing
# scheme used
prior_fnm <-
 nrow(comparison_list$record_pairs[true_links &
 (!reduced_comparison_list$pairs_to_keep), ])

# Find which record pairs are in the same clusters in the full estimate
full_estimate_links <-
 full_estimate[reduced_comparison_list$record_pairs[, 1]] ==
 full_estimate[reduced_comparison_list$record_pairs[, 2]]

# Find the number of true matches in the full estimate
true_matches <- sum(full_estimate_links & true_links_reduced)

# Precision of the full estimate
true_matches / sum(full_estimate_links)

# Recall of the full estimate
true_matches / (sum(true_links_reduced) + prior_fnm)

# Find the partial Bayes estimate
partial_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = 0.1, max_cc_size = 12)

# The partial estimate abstains from making decisions for how many records?
sum(partial_estimate == -1)

# For the records which decisions were made for in the partial estimate,
# there are how many clusters? (including records determined not to be
# candidate matches to any other records using reduce_comparison_data)
length(unique(partial_estimate)) +
 sum(reduced_comparison_list$file_sizes_not_included)

# Abstain rate of partial_estimat (excluding records determined not
# to be candidate matches to any other records using reduce_comparison_data)
```

```
sum(partial_estimate == -1) / length(partial_estimate)

# Relabel records where we abstained
partial_estimate[which(partial_estimate == -1)] <- length(partial_estimate) +
which(partial_estimate == -1)

# Find which record pairs are in the same clusters in the full estimate
partial_estimate_links <-
 partial_estimate[reduced_comparison_list$record_pairs[, 1]] ==
 partial_estimate[reduced_comparison_list$record_pairs[, 2]]

# Find the number of true matches in the partial estimate
true_matches_A <- sum(partial_estimate_links & true_links_reduced)

# Precision of the partial estimate
true_matches_A / sum(partial_estimate_links)

# Relabel the full and partial Bayes estimates
full_estimate_relabel <- relabel_bayes_estimate(reduced_comparison_list,
 full_estimate)

partial_estimate_relabel <- relabel_bayes_estimate(reduced_comparison_list,
 partial_estimate)

# Add columns to the records corresponding to their full and partial
# Bayes estimates
dup_data_small$records <- cbind(dup_data_small$records,
 full_estimate_id = full_estimate_relabel$link_id,
 partial_estimate_id = partial_estimate_relabel$link_id)
```

`no_dup_data` *No Duplicate Dataset*

Description

A dataset containing `730` simulated records from `3` files with no duplicate records in each file.

Usage

```
no_dup_data
```

Format

A list with three elements:

records A `data.frame` with the records, containing `7` fields, from all three files, in the format used for input to `create_comparison_data`.

file_sizes The size of each file.

IDs The true partition of the records, represented as an `integer` vector of arbitrary labels of length `sum(file_sizes)`.

Source

Extracted from the datasets used in the simulation study of the paper. The datasets were generated using code from Peter Christen's group https://dmm.anu.edu.au/geco/index.php.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242] [arXiv]

Examples

```
data(no_dup_data)

# There are 500 entities represented in the records
length(unique(no_dup_data$IDs))
```

---

no_dup_data_small *Small No Duplicate Dataset*

Description

A dataset containing 71 simulated records from 3 files with no duplicate records in each file, subset from `no_dup_data`.

Usage

```
no_dup_data_small
```

Format

A list with three elements:

records A `data.frame` with the records, containing 7 fields, from all three files, in the format used for input to `create_comparison_data`.

file_sizes The size of each file.

IDs The true partition of the records, represented as an `integer` vector of arbitrary labels of length `sum(file_sizes)`.

Source

Extracted from the datasets used in the simulation study of the paper. The datasets were generated using code from Peter Christen's group https://dmm.anu.edu.au/geco/index.php.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242] [arXiv]

Examples

```
data(no_dup_data_small)

# There are 71 entities represented in the records
length(unique(no_dup_data_small$IDs))
```

`reduce_comparison_data` *Reduce Comparison Data Size*

Description

Use indexing to reduce the number of record pairs that are potential matches.

Usage

```
reduce_comparison_data(comparison_list, pairs_to_keep, cc = 1)
```

Arguments

`comparison_list`	The output of a call to `create_comparison_data`.
`pairs_to_keep`	A `logical` vector, the same length as `comparison_list$record_pairs`, indicating which record pairs should be kept as potential matches. These potential matches do not have to be transitive (see the argument `cc`).
`cc`	A `numeric` indicator of whether to find the transitive closure of `pairs_to_keep`, and use these potential matches instead of just those from `pairs_to_keep`. `cc` should be `1` if the transitive closure is being used, and `cc` should be `0` if the transitive closure is not being used. We recommend setting `cc` to `1`.

Details

When using comparison-based record linkage methods, scalability is a concern, as the number of record pairs is quadratic in the number of records. In order to address these concerns, it's common to declare certain record pairs to not be potential matches a priori, using indexing methods. The user is free to index using any method they like, as long as they can produce a `logical` vector that indicates which record pairs are potential matches according to their indexing method. We recommend, if the user chosen indexing method does not output potential matches that are transitive, to set the `cc` argument to `1`. By transitive we mean, for any three records i, j, and k, if i and j are potential matches, and j and k are potential matches, then i and k are potential matches. Non-transitive

indexing schemes can lead to poor mixing of the Gibbs sampler used for posterior inference, and suggests that the indexing method used may have been too stringent.

If indexing is used, it may be the case that some records are declared to not be potential matches to any other records. In this case, the indexing method has made the decision that these records have no matches, and thus we can remove them from the data set and relabel the remaining records; see the documentation for `labels` for information on how to go between the original labeling and the new labeling.

If indexing is used, comparisons for record pairs that aren't potential matches are still used during inference, where they're used to inform the distribution of comparisons for non-matches.

Value

a list containing:

`record_pairs` A `data.frame`, where each row contains the pair of records being compared in the corresponding row of `comparisons`. The rows are sorted in ascending order according to the first column, with ties broken according to the second column in ascending order. For any given row, the first column is less than the second column, i.e. `record_pairs[i, 1] < record_pairs[i, 2]` for each row `i`. If according to `pairs_to_keep` there are records which are not potential matches to any other records, the remaining records are relabeled (see `labels`).

`comparisons` A `logical` matrix, where each row contains the comparisons between the record pair in the corresponding row of `record_pairs`. Comparisons are in the same order as the columns of `records`, and are represented by `L + 1` columns of `TRUE/FALSE` indicators, where `L + 1` is the number of disagreement levels for the field based on `breaks`.

`K` The number of files, assumed to be of class `numeric`.

`file_sizes` A `numeric` vector of length `K`, indicating the size of each file. If according to `pairs_to_keep` there are records which are not potential matches to any other records, the remaining records are relabeled (see `labels`), and `file_sizes` now represents the sizes of each file after removing such records.

`duplicates` A `numeric` vector of length `K`, indicating which files are assumed to have duplicates. `duplicates[k]` should be `1` if file `k` has duplicates, and `duplicates[k]` should be `0` if file `k` has no duplicates.

`field_levels` A `numeric` vector indicating the number of disagreement levels for each field.

`file_labels` An `integer` vector of length `sum(file_sizes)`, where `file_labels[i]` indicated which file record `i` is in.

`fp_matrix` An `integer` matrix, where `fp_matrix[k1, k2]` is a label for the file pair `(k1, k2)`. Note that `fp_matrix[k1, k2] = fp_matrix[k2, k1]`.

`rp_to_fp` A `logical` matrix that indicates which record pairs belong to which file pairs. `rp_to_fp[fp, rp]` is `TRUE` if the records `record_pairs[rp, ]` belong to the file pair `fp`, and is FALSE otherwise. Note that `fp` is given by the labeling in `fp_matrix`.

`ab` An `integer` vector, of length `ncol(comparisons) * K * (K + 1) / 2` that indicates how many record pairs there are with a given disagreement level for a given field, for each file pair.

`file_sizes_not_included` If according to `pairs_to_keep` there are records which are not potential matches to any other records, the remaining records are relabeled (see `labels`), and `file_sizes_not_included` indicates, for each file, the number of such records that were removed.

`ab_not_included` For record pairs not included according to `pairs_to_keep`, this is an `integer` vector, of length `ncol(comparisons) * K * (K + 1) / 2` that indicates how many record pairs there are with a given disagreement level for a given field, for each file pair.

`labels` If according to `pairs_to_keep` there are records which are not potential matches to any other records, the remaining records are relabeled. `labels` provides a dictionary that indicates, for each of the new labels, which record in the original labeling the new label corresponds to. In particular, the first column indicates the record in the original labeling, and the second column indicates the new labeling.

`pairs_to_keep` A `logical` vector, the same length as `comparison_list$record_pairs`, indicating which record pairs were kept as potential matches. This may not be the same as the input `pairs_to_keep` if `cc` was set to 1.

`cc` A `numeric` indicator of whether the connected components of the potential matches are closed under transitivity.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242][arXiv]

Examples

```
# Example with small duplicate dataset
data(dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = dup_data_small$file_sizes,
 duplicates = c(1, 1, 1))

# Reduce the comparison data
# The following line corresponds to only keeping pairs of records for which
# neither gname nor fname disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
reduced_comparison_list <- reduce_comparison_data(comparison_list,
 pairs_to_keep, cc = 1)
```

`relabel_bayes_estimate`

Relabel the Bayes Estimate of a Partition

Description

Relabel the Bayes estimate of a partition, for use after using indexing to reduce the number of record pairs that are potential matches.

Usage

```
relabel_bayes_estimate(reduced_comparison_list, bayes_estimate)
```

Arguments

reduced_comparison_list	The output from a call to reduce_comparison_data.
bayes_estimate	The output from a call to find_bayes_estimate.

Details

When the function reduce_comparison_data is used to reduce the number of record pairs that are potential matches, it may be the case that some records are declared to not be potential matches to any other records. In this case, the indexing method has made the decision that these records have no matches, and thus we can remove them from the data set and relabel the remaining records; see the documentation for labels in reduce_comparison_data for information on how to go between the original labeling and the new labeling. The purpose of this function is to relabel the output of find_bayes_estimate when the function reduce_comparison_data is used, so that the user doesn't have to do this relabeling themselves.

Value

A data.frame, with as many rows as sum(reduced_comparison_list$file_sizes + reduced_comparison_list$file_s
i.e. the number of records originally input to create_comparison_data, before indexing occurred. This data.frame has two columns, "original_labels" and "link_id". Given row i of records originally input to create_comparison_data, the linkage id according to bayes_estimate is given by the ith row of the link_id column. See the documentation for find_bayes_estimate for information on how to interpret this linkage id.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242][arXiv]

Examples

```
# Example with small duplicate dataset
data(dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = dup_data_small$file_sizes,
 duplicates = c(1, 1, 1))

# Reduce the comparison data
# The following line corresponds to only keeping pairs of records for which
# neither gname nor fname disagree at the highest level
```

```
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
reduced_comparison_list <- reduce_comparison_data(comparison_list,
 pairs_to_keep, cc = 1)

# Specify the prior
prior_list <- specify_prior(reduced_comparison_list, mus = NA, nus = NA,
 flat = 0, alphas = rep(1, 7), dup_upper_bound = c(10, 10, 10),
 dup_count_prior_family = c("Poisson", "Poisson", "Poisson"),
 dup_count_prior_pars = list(c(1), c(1), c(1)), n_prior_family = "uniform",
 n_prior_pars = NA)

# Run the Gibbs sampler
{
results <- gibbs_sampler(reduced_comparison_list, prior_list, n_iter = 1000,
 seed = 42)

# Find the full Bayes estimate
full_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = Inf, max_cc_size = 50)

# Find the partial Bayes estimate
partial_estimate <- find_bayes_estimate(results$partitions, burn_in = 100,
 L_FNM = 1, L_FM1 = 1, L_FM2 = 2, L_A = 0.1, max_cc_size = 12)

# Relabel the full and partial Bayes estimates
full_estimate_relabel <- relabel_bayes_estimate(reduced_comparison_list,
 full_estimate)

partial_estimate_relabel <- relabel_bayes_estimate(reduced_comparison_list,
 partial_estimate)

# Add columns to the records corresponding to their full and partial
# Bayes estimates
dup_data_small$records <- cbind(dup_data_small$records,
 full_estimate_id = full_estimate_relabel$link_id,
 partial_estimate_id = partial_estimate_relabel$link_id)
}
```

`specify_prior` *Specify the Prior Distributions*

Description

Specify the prior distributions for the m and u parameters of the models for comparison data among matches and non-matches, and the partition.

Usage

```
specify_prior(
```

```
  comparison_list,
  mus = NA,
  nus = NA,
  flat = 0,
  alphas = NA,
  dup_upper_bound = NA,
  dup_count_prior_family = NA,
  dup_count_prior_pars = NA,
  n_prior_family = NA,
  n_prior_pars = NA
)
```

Arguments

`comparison_list`
: the output from a call to `create_comparison_data` or `reduce_comparison_data`. Note that in order to correctly specify the prior, if `reduce_comparison_data` is used to the reduce the number of record pairs that are potential matches, then the output of `reduce_comparison_data` (not `create_comparison_data`) should be used for this argument.

`mus`, `nus`
: The hyperparameters of the Dirichlet priors for the m and u parameters for the comparisons among matches and non-matches, respectively. These are positive `numeric` vectors which have length equal to the number of columns of `comparison_list$comparisons` times the number of file pairs (`comparison_list$K * (comparison_list$K + 1) / 2`). If set to `NA`, flat priors are used. We recommend using flat priors for m and u.

`flat`
: A `numeric` indicator of whether a flat prior for partitions should be used. `flat` should be `1` if a flat prior is used, and `flat` should be `0` if a structured prior is used. If a flat prior is used, the remaining arguments should be set to `NA`. Otherwise, the remaining arguments should be specified. We do not recommend using a flat prior for partitions in general.

`alphas`
: The hyperparameters for the Dirichlet-multinomial overlap table prior, a positive `numeric` vector of length `2 ^ comparison_list$K - 1`. The indexing of these hyperparameters is based on the the `comparison_list$K`-bit binary representation of the inclusion patterns of the overlap table. To give a few examples, suppose `comparison_list$K` is `3`. `1` in `3`-bit binary is `001`, so `alphas[1]` is the hyperparameter for the `001` cell of the overlap table, representing clusters containing only records from the third file. `2` in `3`-bit binary is `010`, so `alphas[2]` is the hyperparameter for the `010` cell of the overlap table, representing clusters containing only records from the second file. `3` in `3`-bit binary is `011`, so `alphas[3]` is the hyperparameter for the `011` cell of the overlap table, representing clusters containing only records from the second and third files. If set to `NA`, the hyperparameters will all be set to `1`.

`dup_upper_bound`
: A `numeric` vector indicating the maximum number of duplicates, from each file, allowed in each cluster. For a given file `k`, `dup_upper_bound[k]` should be between `1` and `comparison_list$file_sizes[k]`, i.e. even if you don't want to impose an upper bound, you have to implicitly place an upper bound: the

number of records in a file. If set to NA, the upper bound for file k will be set to 1 if no duplicates are allowed for that file, or comparison_list$file_sizes[k] if duplicates are allowed for that file.

dup_count_prior_family
: A character vector indicating the prior distribution family used for the number of duplicates in each cluster, for each file. Currently the only option is "Poisson" for a Poisson prior, truncated to lie between 1 and dup_upper_bound[k]. The mean parameter of the Poisson distribution is specified using the dup_count_prior_pars argument. If set to NA, a Poisson prior with mean 1 will be used.

dup_count_prior_pars
: A list containing the parameters for the prior distribution for the number of duplicates in each cluster, for each file. For file k, when dup_count_prior_family[k]="Poisson", dup_count_prior_pars[[k]] is a positive constant representing the mean of the Poisson prior.

n_prior_family
: A character indicating the prior distribution family used for n, the number of clusters represented in the records. Note that this includes records determined not to be potential matches to any other records using reduce_comparison_data. Currently the there are two options: "uniform" for a uniform prior for n, i.e. $p(n) \propto 1$, and "scale" for a scale prior for n, i.e. $p(n) \propto 1/n$. If set to NA, a uniform prior will be used.

n_prior_pars
: Currently set to NA. When more prior distribution families for n are implemented, this will be a vector of parameters for those priors.

Details

The purpose of this function is to specify prior distributions for all parameters of the model. Please note that if reduce_comparison_data is used to the reduce the number of record pairs that are potential matches, then the output of reduce_comparison_data (not create_comparison_data) should be used as input.

For the hyperparameters of the Dirichlet priors for the m and u parameters for the comparisons among matches and non-matches, respectively, we recommend using a flat prior. This is accomplished by setting mus=NA and nus=NA. Informative prior specifications are possible, but in practice they will be overwhelmed by the large number of comparisons.

For the prior for partitions, we do not recommend using a flat prior. Instead we recommend using our structure prior for partitions. By setting flat=0 and the remaining arguments to NA, one obtains the default specification for the structured prior that we have found to perform well in simulation studies. The structured prior for partitions is specified as follows:

- Specify a prior for n, the number of clusters represented in the records. Note that this includes records determined not to be potential matches to any other records using reduce_comparison_data. Currently, a uniform prior and a scale prior for n are supported. Our default specification uses a uniform prior.
- Specify a prior for the overlap table (see the documentation for alphas for more information). Currently a Dirichlet-multinomial prior is supported. Our default specification sets all hyperparameters of the Dirichlet-multinomial prior to 1.
- For each file, specify a prior for the number of duplicates in each cluster. As a part of this prior, we specify the maximum number of records in a cluster for each file, through

dup_upper_bound. When there are assumed to be no duplicates in a file, the maximum number of records in a cluster for that file is set to 1. When there are assumed to be duplicates in a file, we recommend setting the maximum number of records in a cluster for that file to be less than the file size, if prior knowledge allows. Currently, a Poisson prior for the the number of duplicates in each cluster is supported. Our default specification uses a Poisson prior with mean 1.

Please contact the package maintainer if you need new prior families for n or the number of duplicates in each cluster to be supported.

Value

a list containing:

mus The hyperparameters of the Dirichlet priors for the m parameters for the comparisons among matches.

nus The hyperparameters of the Dirichlet priors for the u parameters for the comparisons among non-matches. Includes data from comparisons of record pairs that were declared to not be potential matches using reduce_comparison_data.

flat A numeric indicator of whether a flat prior for partitions should be used. flat is 1 if a flat prior is used, and flat is 0 if a structured prior is used.

no_dups A numeric indicator of whether no duplicates are allowed in all of the files.

alphas The hyperparameters for the Dirichlet-multinomial overlap table prior, a positive numeric vector of length 2 ^ comparison_list$K, where the first element is 0.

alpha_0 The sum of alphas.

dup_upper_bound A numeric vector indicating the maximum number of duplicates, from each file, allowed in each cluster. For a given file k, dup_upper_bound[k] should be between 1 and comparison_list$file_sizes[k], i.e. even if you don't want to impose an upper bound, you have to implicitly place an upper bound: the number of records in a file.

log_dup_count_prior A list containing the log density of the prior distribution for the number of duplicates in each cluster, for each file.

log_n_prior A numeric vector containing the log density of the prior distribution for the number of clusters represented in the records.

nus_specified The nus before data from comparisons of record pairs that were declared to not be potential matches using reduce_comparison_data are added. Used for input checking.

References

Serge Aleshin-Guendel & Mauricio Sadinle (2022). Multifile Partitioning for Record Linkage and Duplicate Detection. *Journal of the American Statistical Association*. [doi: 10.1080/01621459.2021.2013242] [arXiv]

Examples

```
# Example with small no duplicate dataset
data(no_dup_data_small)
```

```
# Create the comparison data
comparison_list <- create_comparison_data(no_dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = no_dup_data_small$file_sizes,
 duplicates = c(0, 0, 0))

# Specify the prior
prior_list <- specify_prior(comparison_list, mus = NA, nus = NA, flat = 0,
 alphas = rep(1, 7), dup_upper_bound = c(1, 1, 1),
 dup_count_prior_family = NA, dup_count_prior_pars = NA,
 n_prior_family = "uniform", n_prior_pars = NA)

# Example with small duplicate dataset
data(dup_data_small)

# Create the comparison data
comparison_list <- create_comparison_data(dup_data_small$records,
 types = c("bi", "lv", "lv", "lv", "lv", "bi", "bi"),
 breaks = list(NA,  c(0, 0.25, 0.5),  c(0, 0.25, 0.5),
               c(0, 0.25, 0.5), c(0, 0.25, 0.5),  NA, NA),
 file_sizes = dup_data_small$file_sizes,
 duplicates = c(1, 1, 1))

# Reduce the comparison data
# The following line corresponds to only keeping pairs of records for which
# neither gname nor fname disagree at the highest level
pairs_to_keep <- (comparison_list$comparisons[, "gname_DL_3"] != TRUE) &
 (comparison_list$comparisons[, "fname_DL_3"] != TRUE)
reduced_comparison_list <- reduce_comparison_data(comparison_list,
 pairs_to_keep, cc = 1)

# Specify the prior
prior_list <- specify_prior(reduced_comparison_list, mus = NA, nus = NA,
 flat = 0, alphas = rep(1, 7), dup_upper_bound = c(10, 10, 10),
 dup_count_prior_family = c("Poisson", "Poisson", "Poisson"),
 dup_count_prior_pars = list(c(1), c(1), c(1)), n_prior_family = "uniform",
 n_prior_pars = NA)
```

Index