Package 'reclin2'

October 14, 2022

Type Package

Title Record Linkage Toolkit

Version 0.2.0

Description Functions to assist in performing probabilistic record linkage and deduplication: generating pairs, comparing records, em-algorithm for estimating m- and u-probabilities
(I. Fellegi & A. Sunter (1969) <doi:10.1080/01621459.1969.10501049>,
T.N. Herzog, F.J. Scheuren, & W.E. Winkler (2007),
``Data Quality and Record Linkage Techniques", ISBN:978-0-387-69502-0),
forcing one-to-one matching. Can also be
used for pre- and post-processing for machine learning methods for record linkage. Focus is on memory, CPU performance and flexibility.

BugReports https://github.com/djvanderlaan/reclin2/issues

URL https://github.com/djvanderlaan/reclin2

Depends data.table, R (>= 3.6.0)

Imports stringdist, stats, utils, lpSolve, Rcpp, parallel

Suggests simplermarkdown

LinkingTo Rcpp

VignetteBuilder simplermarkdown

SystemRequirements C++11

License GPL-3

LazyLoad yes

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation yes

Author Jan van der Laan [aut, cre] (<https://orcid.org/0000-0002-0693-1514>)

Maintainer Jan van der Laan <r@eoos.dds.nl>

Repository CRAN

Date/Publication 2022-09-24 20:30:02 UTC

R topics documented:

`add_from_x` *Add a variable from one of the data sets to pairs*

Description

Add a variable from one of the data sets to pairs

Usage

```
add_from_x(pairs, variable, new_variable = variable, ...)

add_from_y(pairs, variable, new_variable = variable, ...)
```

Arguments

pairs	data.table with pairs. Should contain the columns .x and .y.
variable	name of the variable that should be added
new_variable	optional variable name of the new variable in pairs. When omitted variable is used.
...	other parameters are passed on to compare_vars. Especially inplace, x and y might be of interest.

Value

Returns the pairs with the column added. When inplace = TRUE pairs is returned invisibly and the original pairs is modified.

---

cluster_call *Call a function on each of the worker nodes and pass it the pairs*

Description

Call a function on each of the worker nodes and pass it the pairs

Usage

```
cluster_call(pairs, fun, ...)
```

Arguments

pairs	an object or type cluster_pairs as created for example by cluster_pair.
fun	a function to call on each of the worker nodes. See details on the arguments of this function.
...	additional arguments are passed on to fun.

Details

The function will have to accept the following arguments as its first three arguments:

pairs the data.table with the pairs of the worker node.

x a data.table with the portion of x present on the worker node.

y a data.table with y.

Value

The function will return a list with for each worker the result of the function call. When the functions return NULL the result is returned invisibly. Because the result is returned to main node, make sure you don't accidentally return all pairs. If you don't want to return anything end your function with NULL.

Examples

```
# Generate some pairs
library(parallel)
data("linkexample1", "linkexample2")
cl <- makeCluster(2)
pairs <- cluster_pair(cl, linkexample1, linkexample2)
compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))

# Add a new column to pairs
cluster_call(pairs, function(pairs, ...) {
  pairs[, name := firstname & lastname]
  # we don't want to return the pairs; so make sure to return something
  # else
  NULL
})

# Get the number of pairs on each node
lenghts <- cluster_call(pairs, function(pairs, ...) {
  nrow(pairs)
})
lengths <- unlist(lenghts)
lenghts

# Cleanup
stopCluster(cl)
```

`cluster_collect` *Collect pairs from cluster nodes*

Description

Collect pairs from cluster nodes

Usage

```
cluster_collect(pairs, select = NULL, clear = FALSE)
```

Arguments

`pairs`	an object or type `cluster_pairs` as created for example by `cluster_pair`.
`select`	the name of a logical column that is used to select the pairs that will be collected
`clear`	remove the pairs from the cluster nodes

Value

Returns an object of type `pairs` which is a `data.table`. This object can be used as a regular (non-cluster) set of pairs

Examples

```
library(parallel)
data("linkexample1", "linkexample2")
cl <- makeCluster(2)

pairs <- cluster_pair(cl, linkexample1, linkexample2)
local_pairs <- cluster_collect(pairs, clear = FALSE)

compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
model <- problink_em(~ lastname + firstname + address + sex, data = pairs)
predict(model, pairs, type = "mpost", add = TRUE, binary = TRUE)
# Select pairs with a mpost > 0.5
select_threshold(pairs, "selected", "mpost", 0.5)
# Collect the selected pairs
local_pairs <- cluster_collect(pairs, "selected")

stopCluster(cl)
```

`cluster_modify_pairs` *Call a function on each of the worker nodes to modify the pairs on the node*

Description

Call a function on each of the worker nodes to modify the pairs on the node

Usage

```
cluster_modify_pairs(pairs, fun, ..., new_name = NULL)
```

Arguments

`pairs`	an object or type `cluster_pairs` as created for example by `cluster_pair`.
`fun`	a function to call on each of the worker nodes. See details on the arguments of this function.
`...`	additional arguments are passed on to `fun`.
`new_name`	name of new object to assign the pairs to on the cluster nodes.

Details

The function will have to accept the following arguments as its first three arguments:

pairs the `data.table` with the pairs of the worker node.

x a `data.table` with the portion of `x` present on the worker node.

y a `data.table` with `y`.

The function should either return a data.table with the new pairs, or NULL. When a data.table is returned this values will replace the pairs when new_name is missing or create new pairs in the environment new_name. When the function returns NULL it is assumed that the function modified the pairs by reference (e.g. using pairs[, new_var := new_val]). Note that this also means that new_name is ignored.

Value

Will return a cluster_pairs object. When new_name is not given it will return the input pairs invisibly. Otherwise it will return a new cluster_pairs object.

Examples

```
# Generate some pairs
library(parallel)
data("linkexample1", "linkexample2")
cl <- makeCluster(2)
pairs <- cluster_pair(cl, linkexample1, linkexample2)
compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))

# Create a new set of pairs containing a random sample of the original
# pairs.
sample <-  cluster_call(pairs, new_name = "sample", function(pairs, ...) {
  sel <- sample(nrow(pairs), round(nrow(pairs)*0.1))
  pairs[sel, ]
})

# Cleanup
stopCluster(cl)
```

---

cluster_pair *Generate all possible pairs using multiple processes*

Description

Generates all combinations of records from x and y.

Usage

```
cluster_pair(cluster, x, y, deduplication = FALSE, name = "default")
```

Arguments

cluster	a cluster object as created by makeCluster from parallel or from the snow package.
x	first data.frame
y	second data.frame. Ignored when deduplication = TRUE.
deduplication	generate pairs from only x. Ignore y. This is usefull for deduplication of x.
name	the name of the resulting object to create locally on the different R processes.

Details

Generating (all) pairs of the records of two data sets, is usually the first step when linking the two data sets.

`x` is split into `length{cluster}` parts which are distributed over the worker nodes. `y` is copied to each of the nodes. On the nodes then `pair` is called. The pairs are stored in the global object `reclin_env` on the nodes in the variable `name`. The pairs can then be further processes using functions such as `compare_pairs`, and `tabulate_patterns`. The function `cluster_collect` collects the pairs from each of the nodes.

Value

A object of type `cluster_pairs` which is a `list` containing the cluster and the name of the pairs object on the cluster nodes. For the pairs objects created on the nodes see the documentation of `pair`.

See Also

`cluster_pair_blocking` and `cluster_pair_minsim` are other methods to generate pairs.

Examples

```
library(parallel)
data("linkexample1", "linkexample2")
cl <- makeCluster(2)
pairs <- cluster_pair(cl, linkexample1, linkexample2)
stopCluster(cl)
```

---

cluster_pair_blocking *Generate pairs using simple blocking using multiple processes*

Description

Generates all combinations of records from `x` and `y` where the blocking variables are equal.

Usage

```
cluster_pair_blocking(
  cluster,
  x,
  y,
  on,
  deduplication = FALSE,
  name = "default"
)
```

Arguments

cluster	a cluster object as created by makeCluster from parallel or from the snow package.
x	first data.frame
y	second data.frame. Ignored when deduplication = TRUE.
on	the variables defining the blocks or strata for which all pairs of x and y will be generated.
deduplication	generate pairs from only x. Ignore y. This is usefull for deduplication of x.
name	the name of the resulting object to create locally on the different R processes.

Details

Generating (all) pairs of the records of two data sets, is usually the first step when linking the two data sets. However, this often results in a too large number of records. Therefore, blocking is usually applied.

x is split into length{cluster} parts which are distributed over the worker nodes. y is copied to each of the nodes. On the nodes then pair_blocking is called. The pairs are stored in the global object reclin_env on the nodes in the variable name. The pairs can then be further processes using functions such as compare_pairs, and tabulate_patterns. The function cluster_collect collects the pairs from each of the nodes.

Value

A object of type cluster_pairs which is a list containing the cluster and the name of the pairs object on the cluster nodes. For the pairs objects created on the nodes see the documentation of pair.

See Also

cluster_pair and cluster_pair_minsim are other methods to generate pairs.

Examples

```
library(parallel)
data("linkexample1", "linkexample2")
cl <- makeCluster(2)
pairs <- cluster_pair_blocking(cl, linkexample1, linkexample2, "postcode")
stopCluster(cl)
```

`cluster_pair_minsim` *Generate pairs with a minimal similarity using multiple processes*

Description

Generates all combinations of records from `x` and `y` where the blocking variables are equal.

Usage

```
cluster_pair_minsim(
  cluster,
  x,
  y,
  on,
  minsim = 0,
  comparators = list(default_comparator),
  default_comparator = identical(),
  keep_simsum = TRUE,
  deduplication = FALSE,
  name = "default"
)
```

Arguments

`cluster`	a cluster object as created by `makeCluster` from `parallel` or `makeCluster` from `snow`.
`x`	first `data.frame`
`y`	second `data.frame`. Ignored when `deduplication = TRUE`.
`on`	the variables defining the blocks or strata for which all pairs of `x` and `y` will be generated.
`minsim`	minimal similarity score.
`comparators`	named list of functions with which the variables are compared. This function should accept two vectors. Function should either return a vector or a `data.table` with multiple columns.
`default_comparator`	variables for which no comparison function is defined using `comparators` is compares with the function `default_comparator`.
`keep_simsum`	add a variable `minsim` to the result with the similarity score of the pair.
`deduplication`	generate pairs from only `x`. Ignore `y`. This is usefull for deduplication of `x`.
`name`	the name of the resulting object to create locally on the different R processes.

Details

Generating (all) pairs of the records of two data sets, is usually the first step when linking the two data sets. However, this often results in a too large number of records. `pair_minsim` will only keep pairs with a similarity score equal or larger than `minsim`. The similarity score is calculated by summing the results of the comparators for all variables of `on`.

`x` is split into `length{cluster}` parts which are distributed over the worker nodes. `y` is copied to each of the nodes. On the nodes then `cluster_pair_minsim` is called. The pairs are stored in the global object `reclin_env` on the nodes in the variable `name`. The pairs can then be further processes using functions such as `compare_pairs`, and `tabulate_patterns`. The function `cluster_collect` collects the pairs from each of the nodes.

Value

A object of type `cluster_pairs` which is a `list` containing the cluster and the name of the pairs object on the cluster nodes. For the pairs objects created on the nodes see the documentation of `pair`.

See Also

`cluster_pair` and `cluster_pair_blocking` are other methods to generate pairs.

Examples

```
library(parallel)
data("linkexample1", "linkexample2")
cl <- makeCluster(2)
# Either address or postcode has to match to keep a pair
pairs <- cluster_pair_minsim(cl, linkexample1, linkexample2,
   on = c("postcode", "address"), minsim = 1)
stopCluster(cl)
```

---

compare_pairs.cluster_pairs

Compare pairs on a set of variables common in both data sets

Description

Compare pairs on a set of variables common in both data sets

Usage

```
## S3 method for class 'cluster_pairs'
compare_pairs(
  pairs,
  on,
  comparators = list(default_comparator),
```

```
  default_comparator = identical(),
  new_name = NULL,
  ...
)

compare_pairs(
  pairs,
  on,
  comparators = list(default_comparator),
  default_comparator = identical(),
  ...
)

## S3 method for class 'pairs'
compare_pairs(
  pairs,
  on,
  comparators = list(default_comparator),
  default_comparator = identical(),
  x = attr(pairs, "x"),
  y = attr(pairs, "y"),
  inplace = FALSE,
  ...
)
```

Arguments

pairs	data.table with pairs. Should contain the columns .x and .y.
on	character vector of variables that should be compared.
comparators	named list of functions with which the variables are compared. This function should accept two vectors. Function should either return a vector or a data.table with multiple columns.
default_comparator	variables for which no comparison function is defined using comparators is compares with the function default_comparator.
new_name	name of new object to assign the pairs to on the cluster nodes.
...	Ignored for now
x	data.table with one half of the pairs.
y	data.table with the other half of the pairs.
inplace	logical indicating whether pairs should be modified in place. When pairs is large this can be more efficient.

Details

It is assumed the variables in on are present in both x and y. Variables with the same names are added to pairs. When the comparator returns a data.table multiple columns are added to pairs. The names of these columns are variable pasted together with the names of the data.table returned by comparator (separated by "_").

Value

Returns the `data.table` `pairs` with one or more columns added in case of `compare_pairs.pairs`.

In case of `compare_pairs.cluster_pairs`, `compare_pair.pairs` is called on each cluster node and the resulting pairs are assigned to `new_name` in the environment `reclin_env`. When `new_name` is not given (or equal to NULL) the original pairs on the nodes are overwritten.

`compare_vars.cluster_pairs` *Compare pairs on given variables*

Description

Compare pairs on given variables

Usage

```
## S3 method for class 'cluster_pairs'
compare_vars(
  pairs,
  variable,
  on_x = variable,
  on_y = on_x,
  comparator = identical(),
  new_name = NULL,
  ...
)

compare_vars(
  pairs,
  variable,
  on_x = variable,
  on_y = on_x,
  comparator = identical(),
  ...
)

## S3 method for class 'pairs'
compare_vars(
  pairs,
  variable,
  on_x = variable,
  on_y = on_x,
  comparator = identical(),
  x = attr(pairs, "x"),
  y = attr(pairs, "y"),
  inplace = FALSE,
```

```
  ...
)
```

Arguments

pairs	data.table with pairs. Should contain the columns .x and .y.
variable	character vector with name of resulting column name that is added to pairs.
on_x	character vector with the column names from x on which to compare.
on_y	character vector with the column names from y on which to compare.
comparator	function with which the variables are compared. When on_x and on_y have length 1, this function should accept two vectors. Otherwise it will receive two data.tables. Function should either return a vector or a data.table with multiple columns.
new_name	name of new object to assign the pairs to on the cluster nodes.
...	Used to pass additional arguments to methods
x	data.table with one half of the pairs.
y	data.table with the other half of the pairs.
inplace	logical indicating whether pairs should be modified in place. When pairs is large this can be more efficient.

Details

When comparator returns a data.table multiple columns are added to pairs. The names of these columns are variable pasted together with the names of the data.table returned by comparator (separated by "_").

Value

Returns the data.table pairs with one or more columns added.

---

deduplicate_equivalence

Deduplication using equivalence groups

Description

Deduplication using equivalence groups

Usage

```
deduplicate_equivalence(pairs, variable, selection, x = attr(pairs, "x"))
```

Arguments

pairs	a pairs object, such as generated by pair_blocking
variable	name of the variable to create in x that will contain the group labels.
selection	a logical variable with the same length as pairs has rows, or the name of such a variable in pairs. Pairs are only selected when select is TRUE. When missing it is assumed all pairs are selected.
x	the first data set; when missing attr(pairs, "x") is used.

Value

Returns x with a variable containing the group labels. Records with the same group label (should) correspond to the same entity.

---

get_inspect_pairs *Get a subset of pairs to inspect*

Description

Get a subset of pairs to inspect

Usage

```
get_inspect_pairs(
  pairs,
  variable,
  threshold,
  position = NULL,
  n = 11,
  x = attr(pairs, "x"),
  y = attr(pairs, "y")
)
```

Arguments

pairs	data.table with pairs.
variable	name of variable to base the selection on; should be a variable with the similarity score of the pairs.
threshold	the threshold around which to select pairs. Used when position is not given.
position	select pairs around this position (based on order of variable), e.g. position = 1 will select the pairs with the highest similarity score.
n	number of pairs to select. Pairs are selected symmetric around the theshold.
x	data.table with one half of the pairs.
y	data.table with the other half of the pairs.

Value

Returns a list with elements `pairs` with the selected pairs; `x` records from `x` corresponding to the pairs; `y` records from `y` corresponding to the pairs; `position` position of the selected pairs; `index` index of the pairs in `pairs`.

`greedy` *Greedy one-to-one matching of pairs*

Description

Greedy one-to-one matching of pairs

Usage

```
greedy(x, y, weight)
```

Arguments

`x`	id's of lhs of pairs
`y`	id's of rhs of pairs
`weight`	weight of pair

Details

Pairs with the highest weight are selected as long a neither the lhs as the rhs are already selected in a pair with a higher weight.

Value

A logical vector with the same length as `x`.

`identical` *Comparison functions*

Description

Comparison functions

Usage

```
identical()

jaro_winkler(threshold = 0.95)

lcs(threshold = 0.8)

jaccard(threshold = 0.8)
```

Arguments

threshold	threshold to use for the Jaro-Winkler string distance when creating a binary result.

Details

A comparison function should accept two arguments: both vectors. When the function is called with both arguments it should compare the elements in the first vector to those in the second. When called in this way, both vectors have the same length. What the function should return depends on the methods used to score the pairs. Usually the comparison functions return a similarity score with a value of 0 indication complete difference and a value > 0 indicating similarity (often a value of 1 will indicate perfect similarity).

Some methods, such as problink_em, can handle similarity scores, but also need binary values (0/FALSE = complete dissimilarity; 1/TRUE = complete similarity). In order to allow for this the comparison function is called with one argument.

When the comparison is called with one argument, it is passed the result of a previous comparison. The function should translate that result to a binary (TRUE/FALSE or 1/0) result. The result should not contain missing values.

The jaro_winkler, lcs and jaccard functions use the corresponding methods from stringdist except that they are transformed from a distance to a similarity score.

Value

The functions return a comparison function (see details).

Examples

```
cmp <- identical()
x <- cmp(c("john", "mary", "susan", "jack"),
         c("johan", "mary", "susanna", NA))
# Applying the comparison function to the result of the comparison results
# in a logical result, with NA's and values of FALSE set to FALSE
cmp(x)

cmp <- jaro_winkler(0.95)
x <- cmp(c("john", "mary", "susan", "jack"),
         c("johan", "mary", "susanna", NA))
# Applying the comparison function to the result of the comparison results
# in a logical result, with NA's and values below the threshold FALSE
cmp(x)
```

`link` *Use the selected pairs to generate a linked data set*

Description

Use the selected pairs to generate a linked data set

Usage

```
link(
  pairs,
  selection = NULL,
  all = FALSE,
  all_x = all,
  all_y = all,
  x = attr(pairs, "x"),
  y = attr(pairs, "y"),
  suffixes = c(".x", ".y"),
  keep_from_pairs = c(".x", ".y")
)
```

Arguments

`pairs`	a `pairs` object, such as generated by `pair_blocking`
`selection`	a logical variable with the same length as `pairs` has rows, or the name of such a variable in `pairs`. Pairs are only selected when `select` is `TRUE`. When missing `attr(pairs, "selection")` is used when available.
`all`	return all records from `x` and `y`; even those that don't match.
`all_x`	return all records from `x`.
`all_y`	return all records from `y`.
`x`	the first data set; when missing `attr(pairs, "x")` is used.
`y`	the second data set; when missing `attr(pairs, "y")` is used.
`suffixes`	a character vector of length 2 specifying the suffixes to be used for making unique the names of columns in the result.
`keep_from_pairs`	character vector with names of variables in `pairs` that should be included in the output.

Details

Uses the selected pairs to link the two data sets to each other. Renames variables that are in both data sets.

Value

Returns a `data.table` containing records from `x` and `y` and `pairs`. Columns that occur both in `x` and `y` gain a suffix indicating from which data set they are.

`linkexample1` *Tiny example dataset for probabilistic linkage*

Description

Contains fictional records of 7 persons.

Format

Two data frames with resp. 6 and 5 records and 6 columns.

Details

- `id` the id of the person; this contains no errors and can be used to validate the linkage.
- `lastname` the last name of the person; contains errors.
- `firstname` the first name of the persons; contains errors.
- `address` the address; contains errors.
- `sex` the sex; contains errors and missing values.
- `postcode` the postcode; contains no errors.

`match_n_to_m` *Force n to m matching on a set of pairs*

Description

Force n to m matching on a set of pairs

Usage

```
match_n_to_m(x, y, w, n = 1, m = 1)
```

Arguments

`x`	a vector of identifiers for each x in each pair This vector should have a unique value for each element in x.
`y`	a vector of identifiers for each y in each pair This vector should have a unique value for each element in y.
`w`	a vector with weights for each pair. The algorithm will try to maximise the total weight of the selected pairs.
`n`	an integer. Each element of x can be linked to at most n elements of y.
`m`	an integer. Each element of y can be linked to at most m elements of y.

Details

The algorithm will try to select pairs in such a way each element of `x` is matched to at most `n` elements of `y` and that each element of `y` is matched at most `m` elements of `x`. It tries to select elements in such a way that the total weight `w` of the selected elements is maximised.

Value

A logical vector with the same length as `x` indicating the selected records.

Examples

```
d <- data.frame(x=c(1,1,1,2,2,3,3), y=c(1,2,3,4,5,6,7), w=1:7)
# One-to-one matching:
d[match_n_to_m(d$x, d$y, d$w), ]

# N-to-one matching:
d[match_n_to_m(d$x, d$y, d$w, n=999), ]

# One-to-m matching:
d[match_n_to_m(d$x, d$y, d$w, m=999), ]

# N-to-M matching, e.g. select all pairs
d[match_n_to_m(d$x, d$y, d$w, n=999, m=999), ]
```

---

merge_pairs.cluster_pairs

Merge two sets of pairs into one

Description

Merge two sets of pairs into one

Usage

```
## S3 method for class 'cluster_pairs'
merge_pairs(
  pairs1,
  pairs2,
  name = paste(pairs1$name, pairs2$name, sep = "+"),
  ...
)

## S3 method for class 'cluster_pairs'
rbind(...)
```

```
merge_pairs(pairs1, pairs2, ...)

## S3 method for class 'pairs'
merge_pairs(pairs1, pairs2, ...)

## S3 method for class 'pairs'
rbind(...)
```

Arguments

pairs1	the first set of pairs
pairs2	the second set of pairs
name	name of new object to assign the pairs to on the cluster nodes.
...	for rbind the pairs or cluster_pairs objects the need to be combined; for merge_pairs these are passed on to other methods.

Details

The function will give an error when the two sets of pairs have different values for attr(pairs1, "x") and attr(pairs1, "y"). When there attributes are missing the code will execute; the user is then responsible for ensuring that the indices in pairs1 and pairs2 refer to the same datasets.

Value

Returns a pairs or cluster_pairs object where both sets of pairs are combined. Duplicate pairs are removed.

In case of merge_pairs.cluster_pairs, merge_pairs.pairs is called on each cluster node and the resulting pairs are assigned to name in the environment reclin_env.

---

pair *Generate all possible pairs*

Description

Generates all combinations of records from x and y.

Usage

```
pair(x, y, deduplication = FALSE, add_xy = TRUE)
```

Arguments

x	first data.frame
y	second data.frame. Ignored when deduplication = TRUE.
deduplication	generate pairs from only x. Ignore y. This is usefull for deduplication of x.
add_xy	add x and y as attributes to the returned pairs. This makes calling some subsequent operations that need x and y (such as compare_pairs easier.

Details

Generating (all) pairs of the records of two data sets, is usually the first step when linking the two data sets.

Value

A data.table with two columns, .x and .y, is returned. Columns .x and .y are row numbers from data.frames .x and .y respectively.

See Also

pair_blocking and pair_minsim are other methods to generate pairs.

Examples

```
data("linkexample1", "linkexample2")
pairs <- pair(linkexample1, linkexample2)
```

---

pair_blocking *Generate pairs using simple blocking*

Description

Generates all combinations of records from x and y where the blocking variables are equal.

Usage

```
pair_blocking(x, y, on, deduplication = FALSE, add_xy = TRUE)
```

Arguments

x	first data.frame
y	second data.frame. Ignored when deduplication = TRUE.
on	the variables defining the blocks or strata for which all pairs of x and y will be generated.
deduplication	generate pairs from only x. Ignore y. This is usefull for deduplication of x.
add_xy	add x and y as attributes to the returned pairs. This makes calling some subsequent operations that need x and y (such as compare_pairs easier.

Details

Generating (all) pairs of the records of two data sets, is usually the first step when linking the two data sets. However, this often results in a too large number of records. Therefore, blocking is usually applied.

Value

A data.table with two columns, .x and .y, is returned. Columns .x and .y are row numbers from data.frames .x and .y respectively.

See Also

pair and pair_minsim are other methods to generate pairs.

Examples

```
data("linkexample1", "linkexample2")
pairs <- pair_blocking(linkexample1, linkexample2, "postcode")
```

---

pair_minsim *Generate pairs with a minimal similarity*

Description

Generates all combinations of records from x and y where the blocking variables are equal.

Usage

```
pair_minsim(
  x,
  y,
  on,
  minsim = 0,
  comparators = list(default_comparator),
  default_comparator = identical(),
  keep_simsum = TRUE,
  deduplication = FALSE,
  add_xy = TRUE
)
```

Arguments

x	first data.frame
y	second data.frame. Ignored when deduplication = TRUE.
on	the variables defining on which the pairs of records from x and y are compared.
minsim	minimal similarity score.
comparators	named list of functions with which the variables are compared. This function should accept two vectors. Function should either return a vector or a data.table with multiple columns.

default_comparator
: variables for which no comparison function is defined using comparators is compares with the function default_comparator.

keep_simsum
: add a variable minsim to the result with the similarity score of the pair.

deduplication
: generate pairs from only x. Ignore y. This is usefull for deduplication of x.

add_xy
: add x and y as attributes to the returned pairs. This makes calling some subsequent operations that need x and y (such as compare_pairs easier.

Details

Generating (all) pairs of the records of two data sets, is usually the first step when linking the two data sets. However, this often results in a too large number of records. pair_minsim will only keep pairs with a similarity score equal or larger than minsim. The similarity score is calculated by summing the results of the comparators for all variables of on.

Missing values in the variables on which the pairs are compared count as a similarity of 0.

Value

A data.table with two columns, .x and .y, is returned. Columns .x and .y are row numbers from data.frames .x and .y respectively.

See Also

pair and pair_blocking are other methods to generate pairs.

Examples

```
data("linkexample1", "linkexample2")
pairs <- pair_minsim(linkexample1, linkexample2,
   on = c("postcode", "address"), minsim = 1)
# Either address or postcode has to match to keep a pair
```

---

predict.problink_em *Calculate weights and probabilities for pairs*

Description

Calculate weights and probabilities for pairs

Usage

```
## S3 method for class 'problink_em'
predict(
  object,
  pairs = newdata,
  newdata = NULL,
```

```
  type = c("weights", "mpost", "probs", "all"),
  binary = FALSE,
  add = FALSE,
  comparators,
  new_name = NULL,
  ...
)
```

Arguments

object	an object of type problink_em as produced by problink_em.
pairs	a object with pairs for which to calculate weights.
newdata	an alternative name for the pairs argument. Specify newdata or pairs.
type	a character vector of length one specifying what to calculate. See results for more information.
binary	convert comparison vectors to binary vectors using the comparison function in comparators.
add	add the predictions to the original pairs object.
comparators	a list of comparison functions (see compare_pairs). When missing attr(pairs, 'comparators') is used.
new_name	name of new object to assign the pairs to on the cluster nodes (only relevant when pairs is of type cluster_pairs.
...	unused.

Value

When pairs is of type pairs, returns a data.table with either the .x and .y columns from pairs (when add = FALSE) or all columns of pairs. To these columns are added:

- In case of type = "weights" a column weights with the calculated weights.
- In case of type = "mpost" a column mpost with the calculated posterior probabilities (probability that pair is a match given comparison vector.
- In case of type = "prob" the columns mprob and uprob with the m and u-probabilites and mpost and upost with the posterior m- and u-probabilities.
- In case of type = "all" all of the above.

In case of compare_pairs.cluster_pairs, compare_pair.pairs is called on each cluster node and the resulting pairs are assigned to new_name in the environment reclin_env. When new_name is not given (or equal to NULL) the original pairs on the nodes are overwritten.

problink_em *Calculate EM-estimates of m- and u-probabilities*

Description

Calculate EM-estimates of m- and u-probabilities

Usage

```
problink_em(
  formula,
  data,
  patterns,
  mprobs0 = list(0.95),
  uprobs0 = list(0.02),
  p0 = 0.05,
  tol = 1e-05,
  mprob_max = 0.999,
  uprob_min = 1e-04
)
```

Arguments

formula	a formula object with the variables for which to calculate the m- and u-probabilities. Should be of the form ~ var1 + var2.
data	data set with pairs on which to estimate the model. Alternatively one can use the patterns argument.
patterns	table of patterns (as output by tabulate_patterns).
mprobs0, uprobs0	initial values of the m- and u-probabilities. These should be lists with numeric values. The names of the elements in the list should correspond to the names in by_x in compare_pairs.
p0	the initial estimate of the probability that a pair is a match.
tol	when the change in the m and u-probabilities is smaller than tol the algorithm is stopped.
mprob_max	maximum values of the estimated m-probabilities. Values equal to one can lead to numerical instabilities.
uprob_min	maximum values of the estimated m-probabilities. Values equal to zero can lead to numerical instabilities.

Value

Returns an object of type problink_em. This is a list containing the estimated mprobs, uprobs and overall linkage probability p. It also contains the table of comparison patterns.

References

Fellegi, I. and A. Sunter (1969). "A Theory for Record Linkage", *Journal of the American Statistical Association*. 64 (328): pp. 1183-1210. doi:10.2307/2286061.

Herzog, T.N., F.J. Scheuren and W.E. Winkler (2007). *Data Quality and Record Linkage Techniques*, Springer.

Examples

```
data("linkexample1", "linkexample2")
pairs <- pair_blocking(linkexample1, linkexample2, "postcode")
pairs <- compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
model <- problink_em(~ lastname + firstname + address + sex, data = pairs)
summary(model)
```

---

select_greedy.cluster_pairs

Select matching pairs enforcing one-to-one linkage

Description

Select matching pairs enforcing one-to-one linkage

Usage

```
## S3 method for class 'cluster_pairs'
select_greedy(
  pairs,
  variable,
  score,
  threshold,
  preselect = NULL,
  id_x = NULL,
  id_y = NULL,
  ...
)

## S3 method for class 'cluster_pairs'
select_n_to_m(
  pairs,
  variable,
  score,
  threshold,
  preselect = NULL,
  id_x = NULL,
  id_y = NULL,
  ...
```

```
)

select_greedy(
  pairs,
  variable,
  score,
  threshold,
  preselect = NULL,
  id_x = NULL,
  id_y = NULL,
  ...
)

## S3 method for class 'pairs'
select_greedy(
  pairs,
  variable,
  score,
  threshold,
  preselect = NULL,
  id_x = NULL,
  id_y = NULL,
  x = attr(pairs, "x"),
  y = attr(pairs, "y"),
  inplace = FALSE,
  ...
)

select_n_to_m(
  pairs,
  variable,
  score,
  threshold,
  preselect = NULL,
  id_x = NULL,
  id_y = NULL,
  ...
)

## S3 method for class 'pairs'
select_n_to_m(
  pairs,
  variable,
  score,
  threshold,
  preselect = NULL,
  id_x = NULL,
  id_y = NULL,
```

```
  x = attr(pairs, "x"),
  y = attr(pairs, "y"),
  inplace = FALSE,
  ...
)
```

Arguments

pairs	a pairs object, such as generated by pair_blocking
variable	the name of the new variable to create in pairs. This will be a logical variable with a value of TRUE for the selected pairs.
score	name of the score/weight variable of the pairs. When not given and attr(pairs, "score") is defined, that is used.
threshold	the threshold to apply. Pairs with a score above the threshold are selected.
preselect	a logical variable with the same length as pairs has rows, or the name of such a variable in pairs. Pairs are only selected when preselect is TRUE. This interacts with threshold (pairs have to be selected with both conditions).
id_x	a integer vector with the same length a the number of rows in pairs, or the name of a column in x. This vector should identify unique objects in x. When not specified it is assumed that each element in x is unique.
id_y	a integer vector with the same length a the number of rows in pairs, or the name of a column in y. This vector should identify unique objects in y. When not specified it is assumed that each element in y is unique.
...	Used to pass additional arguments to methods
x	data.table with one half of the pairs.
y	data.table with the other half of the pairs.
inplace	logical indicating whether pairs should be modified in place. When pairs is large this can be more efficient.

Details

Both methods force one-to-one matching. select_greedy uses a greedy algorithm that selects the first pair with the highest weight. select_n_to_m tries to optimise the total weight of all of the selected pairs. In general this will result in a better selection. However, select_n_to_m uses much more memory and is much slower and, therefore, can only be used when the number of possible pairs is not too large.

Value

Returns the pairs with the variable given by variable added. This is a logical variable indicating which pairs are selected a matches.

Examples

```
data("linkexample1", "linkexample2")
pairs <- pair_blocking(linkexample1, linkexample2, "postcode")
```

```
pairs <- compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
model <- problink_em(~ lastname + firstname + address + sex, data = pairs)
pairs <- predict(model, pairs, type = "mpost", add = TRUE, binary = TRUE)

# Select pairs with a mpost > 0.5 and force one-to-one linkage
pairs <- select_n_to_m(pairs, "ntom", "mpost", 0.5)
pairs <- select_greedy(pairs, "greedy", "mpost", 0.5)
table(pairs$ntom, pairs$greedy)

# The same example as above using a cluster;
library(parallel)
cl <- makeCluster(2)
pairs <- cluster_pair_blocking(cl, linkexample1, linkexample2, "postcode")
compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
model <- problink_em(~ lastname + firstname + address + sex, data = pairs)
predict(model, pairs, type = "mpost", add = TRUE, binary = TRUE)
# Select pairs with a mpost > 0.5 and force one-to-one linkage
# select_n_to_m and select_greedy only work on pairs that are local;
# therefore we first collect the pairs
select_threshold(pairs, "selected", "mpost", 0.5)
local_pairs <- cluster_collect(pairs, "selected")
local_pairs <- select_n_to_m(local_pairs, "ntom", "mpost", 0.5)
local_pairs <- select_greedy(local_pairs, "greedy", "mpost", 0.5)
table(local_pairs$ntom, local_pairs$greedy)

stopCluster(cl)
```

`select_threshold.cluster_pairs` *Select matching pairs with a score above a threshold*

Description

Select matching pairs with a score above a threshold

Usage

```
## S3 method for class 'cluster_pairs'
select_threshold(pairs, variable, score, threshold, new_name = NULL, ...)

select_threshold(pairs, variable, score, threshold, ...)

## S3 method for class 'pairs'
select_threshold(pairs, variable, score, threshold, inplace = FALSE, ...)
```

Arguments

`pairs`	a `pairs` object, such as generated by `pair_blocking`

variable	the name of the new variable to create in pairs. This will be a logical variable with a value of TRUE for the selected pairs.
score	name of the score/weight variable of the pairs. When not given and attr(pairs, "score") is defined, that is used.
threshold	the threshold to apply. Pairs with a score above the threshold are selected.
new_name	name of new object to assign the pairs to on the cluster nodes.
...	ignored
inplace	logical indicating whether pairs should be modified in place. When pairs is large this can be more efficient.

Value

Returns the pairs with the variable given by variable added. This is a logical variable indicating which pairs are selected a matches.

Examples

```
data("linkexample1", "linkexample2")
pairs <- pair_blocking(linkexample1, linkexample2, "postcode")
pairs <- compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
model <- problink_em(~ lastname + firstname + address + sex, data = pairs)
pairs <- predict(model, pairs, type = "mpost", add = TRUE, binary = TRUE)
# Select pairs with a mpost > 0.5
select_threshold(pairs, "selected", "mpost", 0.5, inplace = TRUE)

# Example using cluster;
# In general the syntax is exactly the same except for the first call to
# to cluster_pair. Note the in general `inplace = TRUE` is implied when
# working with a cluster; therefore the assignment back to pairs can be
# omitted (also not a problem if it is not).
library(parallel)
data("linkexample1", "linkexample2")
cl <- makeCluster(2)
pairs <- cluster_pair(cl, linkexample1, linkexample2)
compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
model <- problink_em(~ lastname + firstname + address + sex, data = pairs)
predict(model, pairs, type = "mpost", add = TRUE, binary = TRUE)
# Select pairs with a mpost > 0.5
# Unlike the regular pairs: inplace = TRUE is implied here
select_threshold(pairs, "selected", "mpost", 0.5)
stopCluster(cl)
```

---

summary.problink_em *Summarise the results from* problink_em

Description

Summarise the results from problink_em

Usage

```
## S3 method for class 'problink_em'
summary(object, ...)
```

Arguments

object	the problink_em object.
...	ignored;

Value

Returns the original object with a data.frame with the patterns and corresponding m-, u-probabilities and weights added.

---

tabulate_patterns.cluster_pairs

Create a table of comparison patterns

Description

Create a table of comparison patterns

Usage

```
## S3 method for class 'cluster_pairs'
tabulate_patterns(pairs, on, comparators, complete = TRUE, ...)

tabulate_patterns(pairs, on, comparators, complete = TRUE, ...)

## S3 method for class 'pairs'
tabulate_patterns(pairs, on, comparators, complete = TRUE, ...)
```

Arguments

pairs	a pairs object, such as generated by pair_blocking
on	variables from pairs defining the comparison patterns. When missing names(comparators) is used.
comparators	a list with comparison functions for each of the columns. When missing or NULL, the function looks for columns in pairs with a comparator attribute.
complete	add patterns that do not occur in the dataset to the result (with n = 0).
...	passed on to other methods.

Details

Since comparison vectors can contain continuous numbers (usually between 0 and 1), this could result in a very large number of possible comparison vectors. Therefore, the comparison vectors are passed on to the comparators in order to threshold them. This usually results in values 0 or 1. Missing values are usually codes as 0. However, this all depends on the comparison functions used. For more information see the documentation on the comparison functions.

Value

Returns a `data.frame` with all unique comparison patterns that exist in `pairs`, with a column `n` added with the number of times each pattern occurs.

Examples

```
data("linkexample1", "linkexample2")
pairs <- pair_blocking(linkexample1, linkexample2, "postcode")
pairs <- compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
tabulate_patterns(pairs)
```

---

town_names *Spelling variations of a set of town names*

Description

Contains spelling variations found in various files of a set of town/village names. Names were selected that contain 'rdam' or 'rdm'. The correct/official names are also given. This data set can be used as an example data set for deduplication

Format

Data frames with 584 records and two columns.

Details

- name the name of the town/village as found in the files
- official_name the official/correct name

Index