# Package ‘reclin’

August 9, 2018

**Type** Package  
**Title** Record Linkage Toolkit  
**Version** 0.1.1  
**Date** 2018-08-09  
**Author** Jan van der Laan  
**Maintainer** Jan van der Laan <r@eoos.dds.nl>  
**Description** Functions to assist in performing probabilistic record linkage and deduplication: generating pairs, comparing records, em-algorithm for estimating m- and u-probabilities, forcing one-to-one matching. Can also be used for pre- and post-processing for machine learning methods for record linkage.

**URL** [https://github.com/djvanderlaan/reclin](https://github.com/djvanderlaan/reclin)

**Depends** stats, lvec, ldat, R (>= 3.4.0)  
**Imports** dplyr, stringdist, utils, methods, lpSolve, Rcpp  
**Suggests** testthat, knitr, rmarkdown  
**LinkingTo** Rcpp  
**SystemRequirements** C++11  
**License** GPL-3  
**LazyLoad** yes  
**RoxygenNote** 6.0.1  
**VignetteBuilder** knitr  
**NeedsCompilation** yes  
**Repository** CRAN  
**Date/Publication** 2018-08-09 14:30:03 UTC

## R topics documented:

- `add_from_x`  
- `compare_pairs`
**add_from_x**

**Description**

Add variables from data sets to pairs

**Usage**

```r
add_from_x(pairs, ...)

add_from_y(pairs, ...)
```

**Arguments**

- `pairs`:
  A *pairs* object, such as generated by `pair Blocking`

- `...`:
  A set of option of the form `newvarname = "varname"`, where `varname` is a column in `x` or `y`.

**Value**

A *pairs* object which contains all column of the original *pairs* with the new columns added to it. An error is generated when it is attempted to add variables that already exist in *pairs*.
**compare_pairs**

_Compare all pairs of records_

**Description**

Compare all pairs of records

**Usage**

```r
compare_pairs(pairs, by, comparators = list(default_comparator), x, y, 
  default_comparator = identical(), overwrite = FALSE)
```

**Arguments**

- `pairs`: a pairs object, such as generated by `pair_blocking`
- `by`: variables from `x` and `y` on which to compare the records.
- `comparators`: a names list of comparator functions, for the named variables the given functions will be used to compare the records. For the remaining variables the default_comparator will be used.
- `x`: the first data.frame, when missing attr(pairs, "x") is used.
- `y`: the second data.frame, when missing attr(pairs, "y") is used.
- `default_comparator`: the default comparison function.
- `overwrite`: overwrite exiting variables in `pairs`

**Value**

Returns the pairs object with a column added for each variable in `by`. The value is the column is given by the return value of the corresponding comparison function.
Examples

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

deduplicate_equivalence

Deduplication using equivalence groups

Description

Deduplication using equivalence groups

Usage

deduplicate_equivalence(pairs, var = "duplicate_groups", selection, x)

Arguments

pairs a pairs object, such as generated by pair_blocking
var 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 attr(pairs, "selection") is used when available.
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.

filter_pairs_for_deduplication

Remove pairs which do not have to be compared for deduplication

Description

In case of deduplication one tries to link a data set to itself. Therefore, comparisons only have to be made for records for which the index of the records from the first data set is larger than the index from the record from the second data set.
greedy

**Usage**

```
filter_pairs_for_deduplication(pairs)
```

**Arguments**

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

---

**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 as neither the lhs as the rhs are already selected in a pair with a higher weight.

---

**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 indicating complete difference and a value > 0 indicating similarity (often a value of 1 will indicate perfect similarity).

Some methods, such as `score_problink` and `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

```r
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)
```
Use the selected pairs to generate a linked data set

**Usage**

```r
link(pairs, selection = NULL, x = NULL, y = NULL, all_x = TRUE, all_y = TRUE, ...)
```

**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.
- `x` the first data set; when missing `attr(pairs, "x")` is used.
- `y` the second data set; when missing `attr(pairs, "y")` is used.
- `all_x` return all records from `x`.
- `all_y` return all records from `y`.
- `...` ignored.

**Details**

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

---

**linkeexample1** 

*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**

```r
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.

**Examples**

```r
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), ]
```
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, blocking_var = NULL, large = TRUE, add_xy = TRUE,
chunk_size = 1e+07)

Arguments

x first data.frame
y second data.frame
blocking_var the variables defining the blocks or strata for which all pairs of x and y will be generated.
large should the pairs be returned as a ldat object.
add_xy add x and y as attributes to the returned pairs. This makes calling some subse-
quent operations that need x and y (such as compare_pairs easier.
chunk_size used when large = TRUE to specify the approximate number of pairs that are kept in memory.

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

When large is FALSE, a data.frame with two columns, x and y, is returned. Columns x and y are row numbers from data.frames x and y respectively. When large is TRUE, an object of type ldat is returned.
predict.problink_em

Examples

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

predict.problink_em  Calculate weights and probabilities for pairs

Description

Calculate weights and probabilities for pairs

Usage

```r
## S3 method for class 'problink_em'
predict(object, pairs = newdata, newdata = NULL,
     type = c("weights", "mpost", "probs", "all"), binary = FALSE, comparators,
     ...)
```

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`.
- `comparators`: a list of comparison functions (see `compare_pairs`). When missing `attr(pairs, 'comparators')` is used.
- `...`: unused.

Value

In case of `type == "weights"` returns a vector (lvec or regular R-vector depending on the type of `pairs`), with the linkage weights. In case of `type == "mpost"` returns a vector with the posterior m-probabilities (probability that a pair is a match). In case of `type == "probs"` returns a data.frame or `ldat` with the m- and u-probabilities and posterior m- and u probabilities. In case `type == "all"` returns a data.frame or `ldat` with both probabilities and weights.
problink_em  
Calculate EM-estimates of m- and u-probabilities

Description
Calculate EM-estimates of m- and u-probabilities

Usage
problink_em(patterns, mprobs0 = list(0.95), uprobs0 = list(0.02),
  p0 = 0.05, tol = 1e-05)

Arguments
- patterns: either a table of patterns (as output by `tabulate_patterns`) or pairs with comparison columns (as output by `compare_pairs`).
- 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.

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

Examples
data("linkexample1", "linkexample2")
pairs <- pair_blocking(linkexample1, linkexample2, "postcode")
pairs <- compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
model <- problink_em(pairs)
summary(model)
score_problink

Score comparison patterns of pairs using the probabilistic linkage framework

Description

Score comparison patterns of pairs using the probabilistic linkage framework

Usage

score_problink(pairs, model = NULL, var = "weight", add = TRUE, ...)

Arguments

pairs a pairs object, such as generated by pair_blocking
model an object of type problink_em containing the estimated m- and u-probabilities. When NULL or missing a model is estimated.
var the name of the new variable that will be created (also see details).
add add the estimated score to the pairs object and return the pairs object. Otherwise, just the scores are returned.
... passed on to predict.problink_em.

Value

When add = TRUE, the pairs object is returned with the scores added to it. The new column will have the name var unless additional arguments are passed on to predict.problink_em using the ... argument that causes the calculation of multiple scores (such are type = "all"). In that case the text given by var is prepended to the names of the variables returned by predict.problink_em (with a separator ".").

When add = FALSE the scores are returned as is.

Examples

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

# is the same as
model <- problink_em(pairs)
pairs <- score_problink(pairs, model = model)
Score pairs by summing the similarity vectors

Description

Score pairs by summing the similarity vectors

Usage

score_simsum(pairs, var = "simsum", by, add = TRUE, na_value = 0, ...)

Arguments

pairs a pairs object, such as generated by pair_blocking
var a character vector of length 1 with the name of the variable that will be created.
by a character vector with the column names from pairs that should be summed. When missing the by attribute from pairs is used.
add add the variable to the pairs object and return the pairs object. Otherwise, return a vector with the scores.
na_value the value to use for missing values
... passed on to other methods.

Details

The scores are calculated by summing the columns given by by. Missing values are counted as zeros.

Value

When add = TRUE the original pairs object is returned with the column given by var added to it. Otherwise a vector with scores is returned.

Examples

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

Select matching pairs enforcing one-to-one linkage

Description
Select matching pairs enforcing one-to-one linkage

Usage
```
select_greedy(pairs, threshold = NULL, weight, var = "select",
preselect = NULL, id_x = NULL, id_y = NULL, ...)

select_n_to_m(pairs, threshold = NULL, weight = NULL, var = "select",
preselect = NULL, n = 1, m = 1, id_x = NULL, id_y = NULL, ...)
```

Arguments
- **pairs**: A pairs object, such as generated by `pair_blocking`
- **threshold**: The threshold to apply. Pairs with a score above the threshold are selected.
- **weight**: Name of the score/weight variable of the pairs. When not given and `attr(pairs, "score")` is defined, that is used.
- **var**: 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.
- **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 as `pairs` has rows, 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 as `pairs` has rows, 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.
- **n**: The number of records from `x` that can at most be linked to a record in `y`.
- **m**: The number of records from `y` that can at most be linked to a record in `x`.

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.
select_threshold

Value

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

Examples

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

# Select pairs with a simsum > 5 and force one-to-one linkage
pairs <- select_n_to_m(pairs, 0, var = "ntom")
pairs <- select_greedy(pairs, 0, var = "greedy")
table(pairs[, c("ntom", "greedy")])

Description

Select pairs for linkage using a threshold

Usage

select_threshold(pairs, threshold, weight, var = "select")

Arguments

pairs a pairs object, such as generated by pair_blocking
threshold the threshold to apply. Pairs with a score above the threshold are selected.
weight name of the score/weight variable of the pairs. When not given and attr(pairs, "score") is defined, that is used.
var 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.

Value

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

data("linkexample1", "linkexample2")
pairs <- pair_blocking(linkexample1, linkexample2, "postcode")
pairs <- compare_pairs(pairs, c("lastname", "firstname", "address", "sex"))
pairs <- score_simsum(pairs)
# Select pairs with a simsum > 5 as matches
pairs <- select_threshold(pairs, 5)

---

**summary.problink_em**  
*Summarise the results from problink_em*

Description

Summarise the results from problink_em

Usage

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

Arguments

- `object`  
  the problink_em object.
- `...`  
  ignored;

---

**tabulate_patterns**  
*Create a table of comparison patterns*

Description

Create a table of comparison patterns

Usage

```r
tabulate_patterns(pairs, ..., comparators = NULL, by = NULL)
```
Arguments

- **pairs**: a pairs object, such as generated by `pair_blocking` passed on to other methods.
- **comparators**: a list with comparison functions for each of the columns. When missing or `NULL`, `attr(pairs, "comparators")` is used. Therefore, this parameter usually does not need to be specified.
- **by**: the columns that should be used for the comparison vectors. When missing or `NULL`, `attr(pairs, "by")` is used. Therefore, this parameter usually does not need to be specified.

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

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

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

*Topic datasets
  linkexample1, 7
town_names, 17

add_from_x, 2
add_from_y(add_from_x), 2

comparator functions, 3
compare_pairs, 3, 9–11
comparison function, 3
comparison functions, 17

deduplicate_equivalence, 4

filter_pairs_for_deduplication, 4
greedy, 5

identical, 5

jaccard(identical), 5
jaro_winkler(identical), 5

lcs(identical), 5
ldat, 9, 10
link, 7
linkexample1, 7
linkexample2(linkexample1), 7
lvec, 10

match_n_to_m, 8

pair_blocking, 2–5, 7, 9, 12–15, 17
predict.problink_em, 10, 12
problink_em, 6, 10, 11, 12, 16

score_problink, 6, 12
score_simsum, 13

select_greedy, 14
select_n_to_m(select_greedy), 14
select_threshold, 15
stringdist, 6