Package: RecordLinkage (via r-universe)

September 18, 2024

Version 0.4-12.4

Title Record Linkage Functions for Linking and Deduplicating Data Sets

Description Provides functions for linking and deduplicating data sets. Methods based on a stochastic approach are implemented as well as classification algorithms from the machine learning domain. For details, see our paper ``The RecordLinkage Package: Detecting Errors in Data" Sariyar M / Borg A (2010) <doi:10.32614/RJ-2010-017>.

License GPL (>= 2)

Encoding UTF-8

LazyData true

VignetteBuilder knitr

Depends R (>= 3.5.0), DBI, RSQLite(>= 1.0.0), ff

Imports e1071, rpart, ada, ipred, stats, evd, methods, data.table (>= 1.7.8), nnet, xtable

Suggests RUnit, knitr

Collate register-S3-classes.r RLBigData-classes.r RLResult-class.r accessor-methods.r evt.r classify.r classifySupv-methods.r genSamples.r strcmp.r compare.r getPairs.r summary.r em-methods.r internals.r em.r mygllm.r epilink-methods.r phonetics.r onAttach.r getPairs-methods.r serialization.r tools.r stochastic.r

NeedsCompilation yes

Author Murat Sariyar [aut, cre], Andreas Borg [aut]

Maintainer Murat Sariyar <murat.sariyar@bfh.ch>

Date/Publication 2022-11-08 14:10:15 UTC

Repository https://sym33.r-universe.dev

RemoteUrl https://github.com/cran/RecordLinkage

RemoteRef HEAD

RemoteSha b32452149857b15849e9c82fb81e812df6e921fc

Contents

`classifySupv` *Supervised Classification*

Description

Supervised classification of record pairs based on a trained model.

Usage

```
  classifySupv(model, newdata, ...)

  ## S4 method for signature 'RecLinkClassif,RecLinkData'
classifySupv(model, newdata,
    convert.na = TRUE, ...)

  ## S4 method for signature 'RecLinkClassif,RLBigData'
classifySupv(model, newdata,
    convert.na = TRUE, withProgressBar = (sink.number()==0), ...)
```

Arguments

`model`	Object of class `RecLinkClassif`. The calibrated model. See `trainSupv`.
`newdata`	Object of class `"RecLinkData"` or `"RLBigData"`. The data to classify.
`convert.na`	Logical. Whether to convert missing values in the comparison patterns to 0.
`withProgressBar`	Whether to display a progress bar
`...`	Further arguments for the `predict` method.

Details

The record pairs in `newdata` are classified by calling the appropriate `predict` method for `model$model`.

By default, the `"RLBigDataDedup"` method displays a progress bar unless output is diverted by `sink`, e.g. when processing a Sweave file.

Value

For the `"RecLinkData"` method, a S3 object of class `"RecLinkResult"` that represents a copy of `newdata` with element `rpairs$prediction`, which stores the classification result, as addendum.

For the `"RLBigData"` method, a S4 object of class `"RLResult"`.

Author(s)

Andreas Borg, Murat Sariyar

See Also

trainSupv for training of classifiers, classifyUnsup for unsupervised classification.

Examples

```
# Split data into training and validation set, train and classify with rpart
data(RLdata500)
pairs=compare.dedup(RLdata500, identity=identity.RLdata500,
                    blockfld=list(1,3,5,6,7))
l=splitData(pairs, prop=0.5, keep.mprop=TRUE)
model=trainSupv(l$train, method="rpart", minsplit=5)
result=classifySupv(model=model, newdata=l$valid)
summary(result)
```

---

classifyUnsup *Unsupervised Classification*

Description

Classify record pairs with unsupervised clustering methods.

Usage

```
classifyUnsup(rpairs, method, ...)
```

Arguments

rpairs	Object of type RecLinkData. The data to classify.
method	The classification method to use. One of "kmeans", "bclust".
...	Further arguments for the classification method

Details

A clustering algorithm is applied to find clusters in the comparison patterns. In the case of two clusters (the default), the cluster further from the origin (i.e. representing higher similarity values) is interpreted as the set of links, the other as the set of non-links.

Supported methods are:

kmeans K-means clustering, see kmeans.

bclust Bagged clustering, see bclust.

Value

An object of class "RecLinkResult" that represents a copy of newdata with element rpairs$prediction, which stores the classification result, as addendum.

Author(s)

Andreas Borg, Murat Sariyar

See Also

`trainSupv` and `classifySupv` for supervised classification.

Examples

```
# Classification with bclust
data(RLdata500)
rpairs=compare.dedup(RLdata500, identity=identity.RLdata500,
                     blockfld=list(1,3,5,6,7))
result=classifyUnsup(rpairs,method="bclust")
summary(result)
```

---

clone *Serialization of record linkage object.*

Description

Saving, loading and deep copying of record linkage objects for big data sets.

Usage

```
clone(object, ...)
saveRLObject(object, file, ...)
loadRLObject(file)
```

Arguments

`object`	Object of class "`RLBigData`". The object to save.
`file`	The name of the file to save to or load from.
`...`	Optional arguments for possible additions, currently not used.

Details

The classes for big data sets make use of file-backed data structures from the **ff** package, therefore the `load` and `save` mechanism of R is not sufficient for persistent storage of these objects. Also, assignment via `<-` does not duplicate the **ff** data structures.

`clone` makes a deep copy of an object by duplicating the underlying files.

`saveRLObject` saves an object to zip file containing a dump of the R object as well as the associated **ff** files.

`loadRLObject` loads an object from a file saved by `saveRLObject`.

`clone` and `saveRLObject` are generic functions with methods for "`RLBigData`" and "`RLResult`".

If `loadRLObject` is called with `inPlace = FALSE` (the default), a working copy of the database is made in a temporary file and the original file left untouched. Calling with `inPlace = TRUE` sets the provided file as working copy and changes made to the database are persistent. This option is useful when working with large files in order to prevent disk usage overhead.

`saveRLObject` and `loadRLObject` require working zip / unzip programs.

Value

`clone` returns a deep copy of `object`.

`loadRLObject` returns the loaded object.

`saveRLObject` is used for its side effects.

Note

Objects loaded with `inPlace = TRUE` must be saved again after changes have been made to the object (e.g. calculation of weights).

Author(s)

Andreas Borg, Murat Sariyar

---

compare *Compare Records*

Description

Builds comparison patterns of record pairs for deduplication or linkage.

Usage

```
compare.dedup (dataset, blockfld = FALSE, phonetic = FALSE,
  phonfun = soundex, strcmp = FALSE, strcmpfun = jarowinkler, exclude = FALSE,
  identity = NA, n_match = NA, n_non_match = NA)

compare.linkage (dataset1, dataset2, blockfld = FALSE,
  phonetic = FALSE, phonfun = soundex, strcmp = FALSE,
  strcmpfun = jarowinkler, exclude = FALSE, identity1 = NA, identity2 = NA,
  n_match = NA, n_non_match = NA)
```

Arguments

`dataset`	Table of records to be deduplicated. Either a data frame or a matrix.
`dataset1, dataset2`	Two data sets to be linked.
`blockfld`	Blocking field definition. A list of integer or character vectors with column indices or `FALSE` to disable blocking. See details and examples.

`phonetic` Determines usage of a phonetic code. If `FALSE`, no phonetic code will be used; if `TRUE`, the phonetic code will be used for all columns; if a numeric or character vector is given, the phonetic code will be used for the specified columns.

`phonfun` Function for phonetic code. See details.

`strcmp` Determines usage of a string metric. Used in the same manner as `phonetic`

`strcmpfun` User-defined function for string metric. See details.

`exclude` Columns to be excluded. A numeric or character vector specifying the columns which should be excluded from comparison

`identity`, `identity1`, `identity2`
Optional numerical vectors for identifying matches and non-matches. In a deduplication process, two records `dataset[i,]` and `dataset[j,]` are a true match if and only if `identity[i,]==identity[j,]`. In a linkage process, two records `dataset1[i,]` and `dataset2[j,]` are a true match if and only if `identity1[i,]==identity2[j,]`.

`n_match`, `n_non_match`
Number of desired matches and non-matches in the result.

Details

These functions build record pairs and finally comparison patterns by which these pairs are later classified as links or non-links. They make up the initial stage in a Record Linkage process after possibly normalizing the data. Two general scenarios are reflected by the two functions: `compare.dedup` works on a single data set which is to be deduplicated, `compare.linkage` is intended for linking two data sets together.

Data sets are represented as data frames or matrices (typically of type character), each row representing one record, each column representing one field or attribute (like first name, date of birth...). Row names are not retained in the record pairs. If an identifier other than row number is needed, it should be supplied as a designated column and excluded from comparison (see note on `exclude` below).

Each element of `blockfld` specifies a set of columns in which two records must agree to be included in the output. Each blocking definition in the list is applied individually, the sets obtained thereby are combined by a union operation. If `blockfld` is `FALSE`, no blocking will be performed, which leads to a large number of record pairs ($\frac{n(n-1)}{2}$ where n is the number of records).

As an alternative to blocking, a determined number of `n_match` matches and `n_non_match` non-matches can be drawn if `identity` or `identity1` and `identity2` are supplied. This is relevant for generating training sets for the supervised classificators (see `trainSupv`).

Fields can be excluded from the linkage process by supplying their column index in the vector `exclude`, which is especially useful for external identifiers. Excluded fields can still be used for blocking, also with phonetic code.

Phonetic codes and string similarity measures are supported for enhanced detection of misspellings. Applying a phonetic code leads to a binary values, where 1 denotes equality of the generated phonetic code. A string comparator leads to a similarity value in the range $[0, 1]$.

String comparison is not allowed on a field for which a phonetic code is generated. For phonetic encoding functions included in the package, see phonetics. For the included string comparators, see `jarowinkler` and `levenshteinSim`.

Please note that phonetic code and string metrics can slow down the generation of comparison patterns significantly.

User-defined functions for phonetic code and string comparison can be supplied via the arguments `phonfun` and `strcmpfun`. `phonfun` is expected to have a single character argument (the string to be transformed) and must return a character value with the encoded string.

`strcmpfun` must have as arguments the two strings to be compared and return a similarity value in the range $[0, 1]$, with 0 denoting the lowest and 1 denoting the highest degree of similarity. Both functions must be fully vectorized to work on matrices.

Value

An object of class `RecLinkPairs` with the following components:

`data`	Copy of the records, converted to a data frame.
`pairs`	Generated comparison patterns.
`frequencies`	For each column included in `pairs`, the average frequency of values (reciprocal of number of distinct values).

Author(s)

Andreas Borg, Murat Sariyar

See Also

`RecLinkData` for the format of returned objects.

Examples

```
data(RLdata500)
data(RLdata10000)

# deduplication without blocking, use string comparator on names
## Not run: rpairs=compare.dedup(RLdata500,strcmp=1:4)
# linkage with blocking on first name and year of birth, use phonetic
# code on first components of first and last name

## Not run: rpairs=compare.linkage(RLdata500,RLdata10000,blockfld=c(1,7),phonetic=c(1,3))
# deduplication with blocking on either last name or complete date of birth,
# use string comparator on all fields, include identity information
## Not run: rpairs=compare.dedup(RLdata500, identity=identity.RLdata500, strcmp=TRUE,
  blockfld=list(1,c(5,6,7)))
## End(Not run)

# Draw 100 matches and 1000 non-matches
## Not run: rpairs=compare.dedup(RLdata10000,identity=identity.RLdata10000,n_match=100,
  n_non_match=10000)
## End(Not run)
```

deleteNULLs *Remove NULL Values*

Description

Removes all NULL elements from a list or vector.

Usage

```
deleteNULLs(x)
```

Arguments

x	A vector or list

Value

A copy of x with NULL values removed.

Note

This function is internally used for simple lists and vectors. The behaviour for nested lists and vectors embedded in lists is not thoroughly tested.

References

Taken from a posting by Jim Holtman on the R-help mailing list, https://stat.ethz.ch/pipermail/r-help/2006-August/111896.html

editMatch *Edit Matching Status*

Description

Allows editing the matching status of record pairs.

Usage

```
editMatch(rpairs)
```

Arguments

rpairs	A "RecLinkData" or "RLBigData" object. The record pairs to edit.

Details

This function pops up an editor (via edit) where each record pair in rpairs is printed in two consecutive lines, pairs separated by blank lines. The matching status is printed and can be edited in the last column following the first respective record. A match is denoted by 1, a non-match by 0. NAs are possible to mark pairs with unknown status. Changes in other fields are ignored.

Manual editing of the matching status is useful for clerical review in general and in particular to label training sets. In conjunction with getMinimalTrain, good results can be obtained with a manageable effort of manual review.

Value

A copy of rpairs with edited matching status.

Author(s)

Andreas Borg

See Also

getMinimalTrain

---

emClassify *Weight-based Classification of Data Pairs*

Description

Classifies data pairs to which weights were assigned by emWeights. Based on user-defined thresholds or predefined error rates.

Usage

```
  emClassify(rpairs, threshold.upper = Inf,
    threshold.lower = threshold.upper, my = Inf, ny = Inf, ...)

  ## S4 method for signature 'RecLinkData,ANY,ANY'
emClassify(rpairs, threshold.upper = Inf,
    threshold.lower = threshold.upper, my = Inf, ny = Inf)

  ## S4 method for signature 'RLBigData,ANY,ANY'
emClassify(rpairs, threshold.upper = Inf,
    threshold.lower = threshold.upper, my = Inf, ny = Inf,
    withProgressBar = (sink.number()==0))
```

Arguments

rpairs	RecLinkData object with weight information.
my	A probability. Error bound for false positives.
ny	A probability. Error bound for false negatives.
threshold.upper	A numeric value. Threshold for links.
threshold.lower	A numeric value. Threshold for possible links.
withProgressBar	Whether to display a progress bar
...	Placeholder for method-specific arguments.

Details

Two general approaches are implemented. The classical procedure by Fellegi and Sunter (see references) minimizes the number of possible links with given error levels for false links (my) and false non-links (ny).

The second approach requires thresholds for links and possible links to be set by the user. A pair with weight w is classified as a link if $w \geq$ *threshold.upper*, as a possible link if *threshold.upper* $\geq w \geq$ *threshold.lower* and as a non-link if $w <$ *threshold.lower*.

If threshold.upper or threshold.lower is given, the threshold-based approach is used, otherwise, if one of the error bounds is given, the Fellegi-Sunter model. If only my is supplied, links are chosen to meet the error bound and all other pairs are classified as non-links (the equivalent case holds if only ny is specified). If no further arguments than rpairs are given, a single threshold of 0 is used.

Value

For the "RecLinkData" method, a S3 object of class "RecLinkResult" that represents a copy of newdata with element rpairs$prediction, which stores the classification result, as addendum.

For the "RLBigData" method, a S4 object of class "RLResult".

Note

The quality of classification of the Fellegi-Sunter method relies strongly on reasonable estimations of m- and u-probabilities. The results should be evaluated critically.

Author(s)

Andreas Borg, Murat Sariyar

References

Ivan P. Fellegi, Alan B. Sunter: A Theory for Record Linkage, in: Journal of the American Statistical Association Vol. 64, No. 328 (Dec., 1969), pp. 1183–1210.

See Also

getPairs to produce output from which thresholds can be determined conveniently.

---

emWeights *Calculate weights*

Description

Calculates weights for Record Linkage based on an EM algorithm.

Usage

```
  emWeights(rpairs, cutoff = 0.95, ...)

  ## S4 method for signature 'RecLinkData'
emWeights(rpairs, cutoff = 0.95, ...)

  ## S4 method for signature 'RLBigData'
emWeights(rpairs, cutoff = 0.95,
    verbose = TRUE, ...)
```

Arguments

rpairs	The record pairs for which to compute weights. See details.
cutoff	Either a numeric value in the range [0,1] or a vector with the same length as the number of attributes in the data. Cutoff value for string comparator.
verbose	Logical. Whether to print progress messages.
...	Additional arguments passed to mygllm.

Details

Since package version 0.3, this is a generic functions with methods for S3 objects of class RecLinkData as well as S4 objects of classes "RLBigDataDedup" and "RLBigDataLinkage".

The weight of a record pair is calculated by $\log_2 \frac{M}{U}$, where M and U are estimated m- and u-probabilities for the present comparison pattern. If a string comparator is used, weights are first calculated based on a binary table where all comparison values greater or equal cutoff are set to one, all other to zero. The resulting weight is adjusted by adding for every pair $\log_2 \left(\prod_{j:s_j^i \geq cutoff} s_j^i \right)$, where s_j^i is the value of the string metric for attribute j in data pair i.

The appropriate value of cutoff depends on the choice of string comparator. The default is adjusted to jarowinkler, a lower value (e.g. 0.7) is recommended for levenshteinSim.

Estimation of M and U is done by an EM algorithm, implemented by mygllm. For every comparison pattern, the estimated numbers of matches and non-matches are used to compute the corresponding probabilities. Estimations based on the average frequencies of values and given error rates are taken as initial values. In our experience, this increases stability and performance of the EM algorithm.

Some progress messages are printed to the message stream (see message if verbose == TRUE. This includes progress bars, but these are suppressed if output is diverted by sink to avoid cluttering the output file.

Value

A copy of rpairs with the weights attached. See the class documentation (RecLinkData, "RLBigDataDedup" and "RLBigDataLinkage") on how weights are stored.

Side effects

The "RLBigData" method writes to a disk file containing a ffvector that contains the calculated weights. belonging to object

Author(s)

Andreas Borg, Murat Sariyar

References

William E. Winkler: Using the EM Algorithm for Weight Computation in the Fellegi-Sunter Model of Record Linkage, in: Proceedings of the Section on Survey Research Methods, American Statistical Association 1988, pp. 667–671.

See Also

emClassify for classification of weighted pairs. epiWeights for a different approach for weight calculation.

---

epiClassify *Classify record pairs with EpiLink weights*

Description

Classifies record pairs as link, non-link or possible link based on weights computed by epiWeights and the thresholds passed as arguments.

Usage

```
epiClassify(rpairs, threshold.upper, threshold.lower = threshold.upper,
  ...)

## S4 method for signature 'RecLinkData'
epiClassify(rpairs, threshold.upper, threshold.lower = threshold.upper)

## S4 method for signature 'RLBigData'
epiClassify(rpairs, threshold.upper, threshold.lower = threshold.upper,
  e = 0.01, f = getFrequencies(rpairs), withProgressBar = (sink.number()==0))
```

Arguments

rpairs	RecLinkData object. Record pairs to be classified.
threshold.upper	A numeric value between 0 and 1.
threshold.lower	A numeric value between 0 and 1 lower than threshold.upper
e	Numeric vector. Estimated error rate(s).
f	Numeric vector. Average frequency of attribute values.
withProgressBar	Logical. Whether to display a progress bar.
...	Placeholder for optional arguments

Details

All record pairs with weights greater or equal threshold.upper are classified as links. Record pairs with weights smaller than threshold.upper and greater or equal threshold.lower are classified as possible links. All remaining records are classified as non-links.

For the "RecLinkData" method, weights must have been calculated for rpairs using epiWeights.

A progress bar is displayed by the "RLBigData" method only if weights are calculated on the fly and, by default, unless output is diverted by sink (e.g. in a Sweave script).

Value

For the "RecLinkData" method, a S3 object of class "RecLinkResult" that represents a copy of newdata with element rpairs$prediction, which stores the classification result, as addendum.

For the "RLBigData" method, a S4 object of class "RLResult".

Author(s)

Andreas Borg, Murat Sariyar

See Also

epiWeights

Examples

```
# generate record pairs
data(RLdata500)
p=compare.dedup(RLdata500,strcmp=TRUE ,strcmpfun=levenshteinSim,
  identity=identity.RLdata500, blockfld=list("by", "bm", "bd"))

# calculate weights
p=epiWeights(p)

# classify and show results
summary(epiClassify(p,0.6))
```

epiWeights *Calculate EpiLink weights*

Description

Calculates weights for record pairs based on the EpiLink approach (see references).

Usage

```
  epiWeights(rpairs, e = 0.01, f, ...)

  ## S4 method for signature 'RecLinkData'
epiWeights(rpairs, e = 0.01, f = rpairs$frequencies)

  ## S4 method for signature 'RLBigData'
epiWeights(rpairs, e = 0.01, f = getFrequencies(rpairs),
    withProgressBar = (sink.number()==0))
```

Arguments

rpairs	The record pairs for which to compute weights. See details.
e	Numeric vector. Estimated error rate(s).
f	Numeric vector. Average frequency of attribute values.
withProgressBar	Whether to display a progress bar
...	Placeholder for method-specific arguments.

Details

This function calculates weights for record pairs based on the approach used by Contiero et alia in the EpiLink record linkage software (see references).

Since package version 0.3, this is a generic function with methods for S3 objects of class RecLinkData as well as S4 objects of classes "RLBigDataDedup" and "RLBigDataLinkage".

The weight for a record pair (x^1, x^2) is computed by the formula

$$\frac{\sum_i w_i s(x_i^1, x_i^2)}{\sum_i w_i}$$

where $s(x_i^1, x_i^2)$ is the value of a string comparison of records x^1 and x^2 in the i-th field and w_i is a weighting factor computed by

$$w_i = \log_2(1 - e_i)/f_i$$

, where f_i denotes the average frequency of values and e_i the estimated error rate for field i.

String comparison values are taken from the record pairs as they were generated with compare.dedup or compare.linkage. The use of binary patterns is possible, but in general yields poor results.

The average frequency of values is by default taken from the object `rpairs`. Both frequency and error rate `e` can be set to a single value, which will be recycled, or to a vector with distinct error rates for every field.

The error rate(s) and frequencie(s) must satisfy $e_i \leq 1 - f_i$ for all i, otherwise the functions fails. Also, some other rare combinations can result in weights with illegal values (NaN, less than 0 or greater than 1). In this case a warning is issued.

By default, the `"RLBigDataDedup"` method displays a progress bar unless output is diverted by `sink`, e.g. when processing a Sweave file.

Value

A copy of `rpairs` with the weights attached. See the class documentation (`RecLinkData`, `"RLBigDataDedup"` and `"RLBigDataLinkage"`) on how weights are stored.

For the `"RLBigData"` method, the returned object is only a shallow copy in the sense that it links to the same ff data files as database file as `rpairs`.

Side effects

The `"RLBigData"` method creates a `"ffvector"` object, for which a disk file is created.

Author(s)

Andreas Borg, Murat Sariyar

References

P. Contiero et al., The EpiLink record linkage software, in: Methods of Information in Medicine 2005, 44 (1), 66–71.

See Also

`epiClassify` for classification based on EpiLink weights. `emWeights` for a different approach for weight calculation.

Examples

```
# generate record pairs
data(RLdata500)
p=compare.dedup(RLdata500,strcmp=TRUE ,strcmpfun=levenshteinSim,
  identity=identity.RLdata500, blockfld=list("by", "bm", "bd"))

# calculate weights
p=epiWeights(p)

# classify and show results
summary(epiClassify(p,0.6))
```

ffdf-class *Class* "ffdf"

Description

S4 representation of S3 class "ffdf", created by setOldClass. See ffdf for documentation of the underlying S3 class.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

.S3Class: Object of class "character"

Extends

Class "oldClass", directly.

Methods

No methods defined with class "ffdf" in the signature.

ff_vector-class *Class* "ff_vector"

Description

S4 representation of S3 class "ff_vector", created by setOldClass. See ff for documentation of the underlying S3 class.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

.S3Class: Object of class "character"

Extends

Class "oldClass", directly.

Methods

No methods defined with class "ff_vector" in the signature.

genSamples *Generate Training Set*

Description

Generates training data by unsupervised classification.

Usage

```
genSamples(dataset, num.non, des.mprop = 0.1)
```

Arguments

dataset	Object of class RecLinkData. Data pairs from which to sample.
num.non	Positive Integer. Number of desired non-links in the training set.
des.mprop	Real number in the range [0,1]. Ratio of number of links to number of non-links in the training set.

Details

The application of supervised classifiers (via classifySupv) requires a training set of record pairs with known matching status. Where no such data are available, genSamples can be used to generate training data. The matching status is determined by unsupervised clustering with bclust. Subsequently, the desired number of links and non-links are sampled.

If the requested numbers of matches or non-matches is not feasible, a warning is issued and the maximum possible number is considered.

Value

A list of "RecLinkResult" objects.

train	The sampled training data.
valid	All other record pairs

Record pairs are split into the respective pairs components. The prediction components represent the clustering result. If weights are present in dataset, the corresponding values of Wdata are stored to train and valid. All other components are copied from dataset.

Note

Unsupervised clustering may lead to a poor quality of classification, all subsequent results should be evaluated critically.

Author(s)

Andreas Borg, Murat Sariyar

See Also

splitData for splitting data sets without clustering.

`getErrorMeasures-methods`

Calculate Error Measures

Description

Computes various error measures for the classification of a data set.

Details

Let TP be the number of correctly classified matches (true positives), TN the number of correctly classified non-matches (true negatives), FP and FN the number of misclassified non-matches and matches (false positives and false negatives). The calculated error measures are:

alpha error $\frac{FN}{TP+FN}$

beta error $\frac{FP}{TN+FP}$

accuracy $\frac{TP+TN}{TP+TN+FP+FN}$

precision $\frac{TP}{TP+FP}$

sensitivity $\frac{TP}{TP+FN}$

specificity $\frac{TN}{TN+FP}$

ppv Positive predictive value: $\frac{TP}{TP+FP}$

npv Negative predictive value: $\frac{TN}{TN+FN}$

Value

A list with components `alpha`, `beta`, `accuracy`, `precision`, `sensitivity`, `specificity`, `ppv` and `npv`, each a number in the range $[0, 1]$.

Methods

`signature(object = "RecLinkResult")` Method for S3 result objects of class `"RecLinkResult"`

`signature(object = "RLResult")` Method for S4 objects of class `"RLResult"`, from classification of big data objects (see `"RLBigData"`, `"RLBigDataDedup"`, `"RLBigDataLinkage"`)

A wrapper function `errorMeasures(result)` exists for compatibility with package version 0.2.

Note

Record pairs with unknown true matching status (e.g. due to missing values in the argument `identity` to `RLBigDataDedup`) and possible links are not counted, which can distort the values returned by this function.

Author(s)

Murat Sariyar, Andreas Borg

`getExpectedSize` *Estimate number of record pairs.*

Description

Estimates the total number of record pairs generated by a dataset and specified blocking conditions.

Usage

```
  getExpectedSize(object, ...)

  ## S4 method for signature 'RLBigDataDedup'
getExpectedSize(object)

  ## S4 method for signature 'RLBigDataLinkage'
getExpectedSize(object)

  ## S4 method for signature 'data.frame'
getExpectedSize(object, blockfld = list())
```

Arguments

`object`	Either a record linkage object or a dataset.
`blockfld`	A blocking definition, such as in `compare.dedup`
`...`	Placeholder for additional arguments.

Details

The `"RLBigData*"` methods are only left for backward compatibility. Since version 0.4, all record pairs for such objects are generated and stored in a disk file. The methods return the true number of record pairs.

For the `"data.frame"` method, estimation is based on the assumption that agreement or disagreement of one attribute is independent of the other attributes.

`blockfld` is a blocking definition such as for `RLBigDataDedup`.

Value

The expected number of record pairs.

Author(s)

Andreas Borg, Murat Sariyar

getFrequencies-methods *Get attribute frequencies*

Description

Returns the average frequencies of attribute values for a Record Linkage object, which is `1 / unique(c)`, for every data column `c`.

Methods

`signature(x = "RLBigData")`

getMinimalTrain *Create a minimal training set*

Description

Samples a subset of the provided data (comparison patterns) so that every comparison pattern in `rpairs` is represented in the subset at least once.

Usage

```
getMinimalTrain(rpairs, nEx = 1)
```

Arguments

`rpairs`	A `"RecLinkData"` or `"RLBigData"` object. The data set from which to create a minimal training set.
`nEx`	The desired number of examples per comparison pattern.

Details

Our internal research has given indication that in the context of Record Linkage with supervised classification procedures small training sets are often sufficient, provided they cover the whole range of present comparison patterns.

By default, this function creates a minimal training set that is a subset of the record pairs to be classified in which every present comparison pattern is represented by exactly one training example. By this approach, the work to classify a training set by clerical review can be minimized while keeping a good classification performance.

Larger training sets can be obtained by setting `nEx` to a higher number. Up to `nEx` examples for every comparison pattern are randomly selected, limited by the total number of record pairs with that pattern.

Value

An object of the same class as `rpairs`, representing a minimal comprehensive training set. The appropriate subset of comparison patterns (and weights, if present) is taken, all other components are copied.

Note

Application is only advisable for binary comparison patterns (i.e. only 0 and 1 appear as agreement values). For patterns with string comparison values, the size of the returned set can be too large for a manual review. A warning is issued if fuzzy agreement values (> 0 and < 1) are present in the data.

Note

Due to the small size of the resulting training set, outliers can have a relatively high impact on further classification results. Stable methods such as Bagging or Support-Vector-Machines should be used in conjunction with minimal training sets to minimize this risk.

Author(s)

Andreas Borg, Murat Sariyar

See Also

`editMatch` for manually setting the matching status of the training pairs.

Examples

```
data(RLdata500)
p <- compare.dedup(RLdata500,blockfld=list(1,3),identity=identity.RLdata500)
train <- getMinimalTrain(p)
classif <- trainSupv(train,method="bagging")
summary(classifySupv(classif,newdata=p))
```

---

getPairs *Extract Record Pairs*

Description

Extracts record pairs from data and result objects.

Usage

```
## S4 method for signature 'RecLinkData'
getPairs(object, max.weight = Inf, min.weight = -Inf,
         single.rows = FALSE, show = "all", sort = !is.null(object$Wdata))

## S4 method for signature 'RLBigData'
getPairs(object, max.weight = Inf, min.weight = -Inf,
    filter.match = c("match", "unknown", "nonmatch"),
    withWeight = hasWeights(object), withMatch = TRUE, single.rows = FALSE,
    sort = withWeight)

## S4 method for signature 'RLResult'
getPairs(object, filter.match = c("match", "unknown", "nonmatch"),
    filter.link = c("nonlink", "possible", "link"), max.weight = Inf,
    min.weight = -Inf, withMatch = TRUE, withClass = TRUE,
   withWeight = hasWeights(object@data), single.rows = FALSE, sort = withWeight)

getFalsePos(object, single.rows = FALSE)
getFalseNeg(object, single.rows = FALSE)
getFalse(object, single.rows = FALSE)
```

Arguments

`object`	The data or result object from which to extract record pairs.
`max.weight, min.weight`	Real numbers. Upper and lower weight threshold.
`filter.match`	Character vector, a nonempty subset of `c("match", "nonmatch", "unkown")` denoting which pairs to allow in the output.
`filter.link`	Character vector, a nonempty subset of `c("link", "nonlink", "unkown")` denoting which pairs to allow in the output.
`withWeight`	Logical. Whether to include linkage weights in the output.
`withMatch`	Logical. Whether to include matching status in the output.
`withClass`	Logical. Whether to include classification result in the output.
`single.rows`	Logical. Whether to print record pairs in one row instead of two consecutive rows.
`show`	Character. Selects which records to show, one of `"links"`, `"nonlinks"`, `"possible"`, `"all"`.
`sort`	Logical. Whether to sort descending by weight.

Details

These methods extract record pairs from `"RecLinkData"`, or `"RecLinkResult"`, `"RLBigData"` and `"RLResult"` objects. Possible applications are retrieving a linkage result for further processing, conducting a manual review in order to determine classification thresholds or inspecting misclassified pairs.

The various arguments can be grouped by the following purposes:

1. Controlling which record pairs are included in the output: `min.weight` and `max.weight`, `filter.match`, `filter.link`, `show`.
2. Controlling which information is shown: `withWeight`, `withMatch`, `withClass`
3. Controlling the overall structure of the result: `sort`, `single.rows`.

The weight limits are inclusive, i.e. a record pair with weight `w` is included only if `w >= min.weight && w <= max.weight`.

If `single.rows` is not `TRUE`, pairs are output on two consecutive lines in a more readable format. All data are converted to character, which can lead to a loss of precision for numeric values. Therefore, this format should be used for printing only.

`getFalsePos`, `getFalseNeg` and `getFalse` are shortcuts (currently for objects of class `"RLResult"` only) to retrieve false positives (links that are non-matches in fact), false negatives (non-links that are matches in fact) or all falsely classified pairs, respectively.

Value

A data frame. If `single.rows` is `TRUE`, each row holds (in this order) id and data fields of the first record, id and data fields of the second record and possibly matching status, classification result and/or weight.

If `single.rows` is not `TRUE`, the result holds for each resulting record pair consecutive rows of the following format:

1. ID and data fields of the first record followed by as many empty fields to match the length of the following line.
2. ID and data fields of the second record, possibly followed by matching status, classification result and/or weight.
3. A blank line to separate record pairs.

Note

When non-matches are included in the output and blocking is permissive, the result object can be very large, possibly leading to memory problems.

Author(s)

Andreas Borg, Murat Sariyar

Examples

```
data(RLdata500)

# create record pairs and calculate epilink weights
rpairs <- RLBigDataDedup(RLdata500, identity = identity.RLdata500,
  blockfld=list(1,3,5,6,7))
rpairs <- epiWeights(rpairs)

# show all record pairs with weights between 0.5 and 0.6
```

```
getPairs(rpairs, min.weight=0.5, max.weight=0.6)

# show only matches with weight <= 0.5
getPairs(rpairs, max.weight=0.5, filter.match="match")

# classify with one threshold
result <- epiClassify(rpairs, 0.5)

# show all links, do not show classification in the output
getPairs(result, filter.link="link", withClass = FALSE)

# see wrongly classified pairs
getFalsePos(result)
getFalseNeg(result)
```

`getParetoThreshold` *Estimate Threshold from Pareto Distribution*

Description

Calculates a classification threshold based on a generalized Pareto distribution (GPD) fitted to the weights distribution of the given data pairs.

Usage

```
getParetoThreshold(rpairs, quantil = 0.95, interval = NA)
## S4 method for signature 'RecLinkData'
getParetoThreshold(rpairs, quantil = 0.95, interval = NA)
## S4 method for signature 'RLBigData'
getParetoThreshold(rpairs, quantil = 0.95, interval = NA)
```

Arguments

`rpairs`	A "RecLinkData" or "RLBigData" object with weights.The data for which to compute a threshold.
`quantil`	A real number between 0 and 1. The quantile which to compute.
`interval`	A numeric vector denoting the interval on which to fit a GPD.

Details

This threshold calculation is based on the assumption that the distribution of weights exhibit a 'fat tail' which can be fitted by a generalized Pareto distribution (GPD). The limits of the interval which is subject to the fitting are usually determined by reviewing a mean residual life plot of the data. If the limits are not externally supplied, a MRL plot is displayed from which the endpoints can be selected by mouse input. If only one endpoint is selected or supplied, the greater endpoint is set to the maximum weight. A suitable interval is characterized by a relatively long, approximately linear segment of the plot.

Value

A classification threshold.

Note

The quality of matching varies, poor results can occur in some cases. Evaluate carefully before applying to a real case.

Author(s)

Andreas Borg, Murat Sariyar

References

Sariyar M., Borg A. and Pommerening M.: Controlling false match rates in record linkage using extreme value theory. Journal of Biomedical Informatics, doi:10.1016/j.jbi.2011.02.008.

See Also

`emWeights` and `epiWeights` for calculating weights, `emClassify` and `epiClassify` for classifying with the returned threshold.

Examples

```
data(RLdata500)
rpairs=compare.dedup(RLdata500, identity=identity.RLdata500, strcmp=TRUE,
  blockfld=list(1,3,5:7))
rpairs=epiWeights(rpairs)
# leave out argument interval to choose from plot
## Not run: threshold=getParetoThreshold(rpairs,interval=c(0.68, 0.79))
## Not run: summary(epiClassify(rpairs,threshold))
```

---

getTable-methods *Build contingency table*

Description

Builds a contingency table for a linkage result with counts for each combination of real matching status and predicted result.

Methods

`signature(object = "RecLinkResult")` Method for S3 result sets.

`signature(object = "RLResult")` Method for S4 result sets (big data sets).

gpdEst *Estimate Threshold from Pareto Distribution*

Description

Fits a Pareto distribution to the distribution of weights and calculates a quantile on the fitted model as classification threshold.

Usage

```
gpdEst(Wdata, thresh = -Inf, quantil = 0.95)
```

Arguments

Wdata	A numeric vector representing weights of record pairs.
thresh	Threshold for exceedances.
quantil	A real number between 0 and 1. The desired quantile.

Details

The weights that exceed thresh are fitted to a generalized Pareto distribution (GPD). The estimated parameters shape and scale are used to calculate a classification threshold by the formula

$$thresh + \frac{scale}{shape}((\frac{n}{k}(1 - quantil))^{-shape} - 1)$$

where n is the total number of weights and k the number of exceedances.

Value

A real number representing the resulting classification threshold. It is assured that the threshold lies in a reasonable range.

Author(s)

Murat Sariyar

See Also

getParetoThreshold for user-level function

isFALSE *Check for FALSE*

Description

Shortcut for `identical(x,FALSE)`

Usage

```
isFALSE(x)
```

Arguments

x	Any object

Value

Returns `TRUE` if `x` is identical to `FALSE`, `FALSE` otherwise.

Author(s)

Andreas Borg

mygllm *Generalized Log-Linear Fitting*

Description

Fits a log-linear model for collapsed contingency tables.

Usage

```
mygllm(y, s, X, maxit = 1000, tol = 1e-05, E = rep(1, length(s)))
```

Arguments

y	Vector of observed cell frequencies.
s	Scatter matrix. s[i] is the cell in the observed array that corresponds to cell i in the full array.
X	Design matrix.
maxit	Maximum number of iterations.
tol	Convergence parameter.
E	Full contingency table. Should be initialized with either ones or a priori estimates.

Details

This is an implementation and extension of the algorithm published by Haber (1984). It also incorporates ideas of David Duffy (see references).

A priori estimates of the full contingency table can be given as start values by argument `E`. This can reduce execution time significantly.

Value

Estimated full contingency table.

Author(s)

Andreas Borg, Murat Sariyar

References

Michael Haber, Algorithm AS 207: Fitting a General Log-Linear Model, in: Applied Statistics 33 (1984) No. 3, 358–362.

David Duffy: gllm: Generalised log-linear model. R package version 0.31. https://cran.r-project.org/package=gllm

See Also

`emWeights`, which makes use of log-linear fitting for weight calculation.

---

optimalThreshold *Optimal Threshold for Record Linkage*

Description

Calculates the optimal threshold for weight-based Record Linkage.

Usage

```
optimalThreshold(rpairs, my = NaN, ny = NaN)
## S4 method for signature 'RecLinkData'
optimalThreshold(rpairs, my = NaN, ny = NaN)
## S4 method for signature 'RLBigData'
optimalThreshold(rpairs, my = NaN, ny = NaN)
```

Arguments

`rpairs`	Record pairs for which to calculate a threshold.
`my`	A real value in the range [0,1]. Error bound for false positives.
`ny`	A real value in the range [0,1]. Error bound for false negatives.

Details

Weights must have been calculated for rpairs, for example by emWeights or epiWeights. The true match result must be known for rpairs, mostly this is provided through the identity argument of compare.*

For the following, it is assumed that all records with weights greater than or equal to the threshold are classified as links, the remaining as non-links. If no further arguments are given, a threshold which minimizes the absolute number of misclassified record pairs is returned. If my is supplied (ny is ignored in this case), a threshold is picked which maximizes the number of correctly classified links while keeping the ratio of false links to the total number of links below or equal my. If ny is supplied, the number of correct non-links is maximized under the condition that the ratio of falsely classified non-links to the total number of non-links does not exceed ny.

Two separate runs of optimalThreshold with values for my and ny respectively allow for obtaining a lower and an upper threshold for a three-way classification approach (yielding links, non-links and possible links).

Value

A numeric value, the calculated threshold.

Author(s)

Andreas Borg, Murat Sariyar

See Also

emWeights emClassify epiWeights epiClassify

Examples

```
# create record pairs
data(RLdata500)
p=compare.dedup(RLdata500,identity=identity.RLdata500, strcmp=TRUE,
  strcmpfun=levenshteinSim)

# calculate weights
p=epiWeights(p)

# split record pairs in two sets
l=splitData(dataset=p, prop=0.5, keep.mprop=TRUE)

# get threshold from training set
threshold=optimalThreshold(l$train)

# classify remaining data
summary(epiClassify(l$valid,threshold))
```

phonetics *Phonetic Code*

Description

Interface to phonetic coding functions.

Usage

```
soundex(str)
```

Arguments

str	A character vector or matrix.

Details

soundex is a widespread algorithm for English names. This implementation can only handle common characters. It strips off non-alphabetical characters.

The C code for soundex was taken from PostgreSQL 8.3.6.

Value

A character vector or matrix with the same size and dimensions as str, containing its phonetic encoding.

Author(s)

Andreas Borg (R interface only)

References

see also https://www.codedrome.com/the-soundex-algorithm-in-c/

See Also

jarowinkler and levenshteinSim for string comparison.

RecLinkClassif-class *Class "RecLinkClassif"*

Description

S4 wrapper for S3 class with the same name, which has the same structure as a RecLinkData object plus the following components:

`prediction` Linkage result. Coded as a factor with levels `"N"` for non-links, `"P"` for possible links and `"L"` for links.

`attrNames` Column names of the set of comparison patterns.

Objects from the Class

Objects of the S3 class are created by classification functions, such as `classifySupv` or `emClassify`

Slots

`.S3Class`**:** Object of class `"character"`.

Extends

Class `"oldClass"`, directly.

Methods

classifySupv `signature(model = "RecLinkClassif", newdata = "RecLinkData")`

classifySupv `signature(model = "RecLinkClassif", newdata = "RLBigData")`

Author(s)

Andreas Borg, Murat Sariyar

Examples

```
showClass("RecLinkClassif")
```

RecLinkData-class *Class "RecLinkData"*

Description

S4 wrapper for S3 class "RecLinkData".

Objects from the Class

Objects of the S3 class are created by the comparison functions compare.*. The S4 class is virtual and exists solely for internal usage in method signatures.

Slots

.S3Class: Internal slot.

See "RecLinkData" for the structure of the S3 class.

Extends

Class "oldClass", directly.

Methods

Use getMethods(classes = "RecLinkData") to list the methods defined for this class.

Author(s)

Andreas Borg, Murat Sariyar

See Also

"RecLinkData" for the structure of the S3 class. compare.dedup, which creates objects of this class. "RLBigData", an alternative data structure suitable for big data sets.

Examples

```
showClass("RecLinkData")
```

RecLinkData.object *Record Linkage Data Object*

Description

S3 class representing information about record pairs for Record Linkage, as returned by functions compare.dedup and compare.linkage.

Value

A list with at least the following elements:

data **(for** type = "deduplication"**):** Object of class "data.frame". Data frame of original records.

data1**,** data2 **(for** type = "linkage"**):** Objects of class "data.frame". Data frames of original records.

pairs**:** Object of class "data.frame" Data frame of data pairs. Each row represents the comparison pattern of two records, identified by columns id1 and id2. The other columns contain for each considered attribute a real number in the range [0..1] representing the degree of similarity. These columns are named according to the respective columns in data. The last column contains the matching status of the pair, coded as 1 for a match or 0 for a non-match.

frequencies**:** Object of class "numeric" Numeric vector with average frequency of values for each column included in pairs (reciprocal of number of distinct values).

type**:** Object of class "character" Identifies whether a linkage ("linkage") or a deduplication ("deduplication") project is represented.

.S3class**:** Internal slot.

The following elements are optional:

M**:** Object of class "numeric" Vector of m-probabilities as calculated by emWeights.

U**:** Object of class "numeric" Vector of u-probabilities as calculated by emWeights.

W**:** Object of class "numeric" Vector of log-likelihood weights as calculated by emWeights, corresponding to binary comparison patterns as created by bincombinations.

Wdata**:** Object of class "numeric" Vector of log-likelihood weights as calculated by emWeights, corresponding to the rows of pairs.

Author(s)

Andreas Borg, Murat Sariyar

See Also

"RecLinkData" for the S4 representation. compare.dedup, which creates objects of this class. "RLBigData", an alternative data structure suitable for big data sets.

RecLinkResult-class *Class "RecLinkResult"*

Description

S4 wrapper for S3 class "RecLinkResult".

Objects from the Class

Object of the S3 class are created by classification functions, such as classifySupv and emClassify. The S4 class is virtual and exists solely for internal usage in method signatures.

Slots

.S3Class: Internal slot.

See "RecLinkResult" for the structure of the S3 class.

Extends

Class "RecLinkData", directly. Class "oldClass", by class "RecLinkData", distance 2.

Methods

%append% `signature(x = "RecLinkResult", y = "RecLinkResult")`

getErrorMeasures `signature(object="RecLinkResult")`

getTable `signature(object="RecLinkResult")`

Author(s)

Andreas Borg, Murat Sariyar

See Also

"RecLinkResult" for the structure of the S3 class. "RLResult", the equivalent data structure for big data sets.

Examples

```
showClass("RecLinkResult")
```

RecLinkResult.object *Record Linkage Result Object*

Description

An object representing information about the classification result of a Record Linkage procedure.

Value

`data`, `pairs`, `frequencies`
: Inherited from `RecLinkData`.

`prediction`
: Factor object indicating the classification of each record pair in `valid`. Levels are:
  - `"L"` for links,
  - `"P"` for possible links
  - `"N"` for non-links

Author(s)

Andreas Borg

See Also

`emClassify`. `RecLinkData`.

resample *Safe Sampling*

Description

Performs sampling without replacement while avoiding undesired behaviour if `x` has length 1. See documentation of `sample`.

Usage

```
resample(x, size, ...)
```

Arguments

`x`	A vector from which to sample.
`size`	A non-negative number giving the size of the sample.
`...`	Further arguments to `sample`.

RLBigData-class *Class "RLBigData"*

Description

Abstract class for big data sets.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots in "RLBigData"

`frequencies`: Object of class `"numeric"`. Average frequency of values for each column of the underlying data (1 / of number of distinct values)

`blockFld`: Object of class `"list"`. Blocking definition. See documentation for constructor functions for details.

`excludeFld`: Object of class `"numeric"`. Indices of attributes which are not considered for comparison.

`strcmpFld`: Object of class `"numeric"`. Indices of attributes on which a string comparator is executed.

`strcmpFun`: Object of class `"character"`. String representing the string comparison function.

`phoneticFld`: Object of class `"numeric"`. Indices of attributes on which a phonetic code is generated.

`phoneticFun`: Object of class `"character"`. String representing the phonetic coding function.

`drv`: Object of class `"DBIDriver"`. Database driver.

`con`: Object of class `"DBIConnection"`. Database connection.

Methods

begin `signature(x = "RLBigData")`

classifySupv `signature(model = "RecLinkClassif", newdata = "RLBigData")`

clear `signature(x = "RLBigData")`

clone `signature(object = "RLBigData")`

emClassify `signature(rpairs = "RLBigData")`

emWeights `signature(rpairs = "RLBigData")`

epiClassify `signature(rpairs = "RLBigData")`

epiWeights `signature(rpairs = "RLBigData")`

getDbFile `signature(object = "RLBigData")`

getFrequencies `signature(x = "RLBigData")`

getMatchCount `signature(object = "RLBigData")`

getNACount `signature(object = "RLBigData")`

getPairs `signature(object = "RLBigData")`

getPatternCounts `signature(x = "RLBigData")`

getSQLStatement `signature(object = "RLBigData")`

nextPairs `signature(x = "RLBigData")`

saveRLObject `signature(object = "RLBigData")`

show `signature(object = "RLBigData")`

Author(s)

Andreas Borg

See Also

Non-abstract subclasses "RLBigDataDedup" and "RLBigDataLinkage" with constructors RLBigDataDedup and RLBigDataLinkage.

Examples

```
showClass("RLBigData")
```

RLBigDataDedup *Constructors for big data objects.*

Description

These are constructors which initialize a record linkage setup for big datasets, either deduplication of one (RLBigDataDedup) or linkage of two datasets (RLBigDataLinkage).

Usage

```
RLBigDataDedup(dataset, identity = NA, blockfld = list(), exclude = numeric(0),
  strcmp = numeric(0), strcmpfun = "jarowinkler", phonetic = numeric(0),
  phonfun = "soundex")

RLBigDataLinkage(dataset1, dataset2, identity1 = NA, identity2 = NA,
  blockfld = list(), exclude = numeric(0), strcmp = numeric(0),
  strcmpfun = "jarowinkler", phonetic = numeric(0), phonfun = "soundex")
```

Arguments

`dataset`, `dataset1`, `dataset2`
: Table of records to be deduplicated or linked. Either a data frame or a matrix.

`identity`, `identity1`, `identity2`
: Optional vectors (are converted to factors) for identifying true matches and non-matches. In a deduplication process, two records `dataset[i,]` and `dataset[j,]` are a true match if and only if `identity[i,]==identity[j,]`. In a linkage process, two records `dataset1[i,]` and `dataset2[j,]` are a true match if and only if `identity1[i,]==identity2[j,]`.

`blockfld`
: Blocking field definition. A numeric or character vector or a list of several such vectors, corresponding to column numbers or names. See details and examples.

`exclude`
: Columns to be excluded. A numeric or character vector corresponding to columns of dataset or dataset1 and dataset2 which should be excluded from comparison

`strcmp`
: Determines usage of string comparison. If `FALSE`, no string comparison will be used; if `TRUE`, string comparison will be used for all columns; if a numeric or character vector is given, the string comparison will be used for the specified columns.

`strcmpfun`
: Character string representing the string comparison function. Possible values are `"jarowinkler"` and `"levenshtein"`.

`phonetic`
: Determines usage of phonetic code. Used in the same manner as `strcmp`

.

`phonfun`
: Character string representing the phonetic function. Currently, only `"soundex"` is supported (see `soundex`).

Details

These functions act as constructors for the S4 classes `"RLBigDataDedup"` and `"RLBigDataLinkage"`. They make up the initial stage in a Record Linkage process using large data sets (>= 1.000.000 record pairs) after possibly normalizing the data. Two general scenarios are reflected by the two functions: `RLBigDataDedup` works on a single data set which is to be deduplicated, `RLBigDataLinkage` is intended for linking two data sets together. Their usage follows the functions `compare.dedup` and `compare.linkage`, which are recommended for smaller amounts of data, e.g. training sets.

Datasets are represented as data frames or matrices (typically of type character), each row representing one record, each column representing one attribute (like first name, date of birth,...). Row names are not retained in the record pairs. If an identifier other than row number is needed, it should be supplied as a designated column and excluded from comparison (see note on `exclude` below).

In case of `RLBigDataLinkage`, the two datasets must have the same number of columns and it is assumed that their column classes and semantics match. If present, the column names of `dataset1` are assigned to `dataset2` in order to enforce a matching format. Therefore, column names used in `blockfld` or other arguments refer to `dataset1`.

Each element of `blockfld` specifies a set of columns in which two records must agree to be included in the output. Each blocking definition in the list is applied individually, the sets obtained thereby are combined by a union operation. If `blockfld` is `FALSE`, no blocking will be performed, which leads to a large number of record pairs ($\frac{n(n-1)}{2}$ where n is the number of records).

Fields can be excluded from the linkage process by supplying their column index in the vector `exclude`, which is especially useful for external identifiers. Excluded fields can still be used for blocking, also with phonetic code.

Phonetic codes and string similarity measures are supported for enhanced detection of misspellings. Applying a phonetic code leads to binary similarity values, where 1 denotes equality of the generated phonetic code. A string comparator leads to a similarity value in the range $[0, 1]$. Using string comparison on a field for which a phonetic code is generated is possible, but issues a warning.

In contrast to the `compare.*` functions, phonetic coding and string comparison is not carried out in R, but by database functions. Supported functions are `"soundex"` for phonetic coding and `"jarowinkler"` and `"levenshtein"` for string comparison. See the documentation for their R equivalents (phonetic functions, string comparison) for further information.

Value

An object of class `"RLBigDataDedup"` or `"RLBigDataLinkage"`, depending on the called function.

Side effects

The RSQLite database driver is initialized via `dbDriver("SQLite")` and a connection established and stored in the returned object. Extension functions for phonetic code and string comparison are loaded into the database. The records in `dataset` or `dataset1` and `dataset2` are stored in tables `"data"` or `"data1"` and `"data2"`, respectively, and indices are created on all columns involved in blocking.

Author(s)

Andreas Borg, Murat Sariyar

See Also

`"RLBigDataDedup"`, `"RLBigDataLinkage"`, `compare.dedup`, `compare.linkage`, the vignette "Classes for record linkage of big data sets".

Examples

```
data(RLdata500)
data(RLdata10000)
# deduplication without blocking, use string comparator on names
rpairs <- RLBigDataDedup(RLdata500, strcmp = 1:4)
# linkage with blocking on first name and year of birth, use phonetic
# code on first components of first and last name
rpairs <- RLBigDataLinkage(RLdata500, RLdata10000, blockfld = c(1, 7),
  phonetic = c(1, 3))
# deduplication with blocking on either last name or complete date of birth,
# use string comparator on all fields, include identity information
rpairs <- RLBigDataDedup(RLdata500, identity = identity.RLdata500, strcmp=TRUE,
  blockfld = list(1, c(5, 6, 7)))
```

RLBigDataDedup-class *Class "RLBigDataDedup"*

Description

Represents a record linkage setup where a single dataset is to be deduplicated.

Objects from the Class

Objects should be created using the constructor function RLBigDataDedup, which does some essential error checking, conversion and initialization.

Slots

See also "RLBigData" for inherited slots.

`data`: Object of class "`data.frame`" Data set.

`identity`: Object of class "`factor`" True ID of records in `data`

Extends

Class "RLBigData", directly.

Methods

getColumnNames `signature(object = "RLBigDataDedup")`

getExpectedSize `signature(object = "RLBigDataDedup")`

See also RLBigData-class for inherited methods.

Author(s)

Andreas Borg

See Also

RLBigDataDedup, RLBigData-class

Examples

```
showClass("RLBigDataDedup")
```

RLBigDataLinkage-class *Class "RLBigDataLinkage"*

Description

Represents a record linkage setup with two datasets which are to be linked together.

Objects from the Class

Objects should be created using the constructor function RLBigDataLinkage, which does some essential error checking, conversion and initialization.

Slots

See also "RLBigData" for inherited slots.

`data1`: Object of class "`data.frame`" First data set.

`data2`: Object of class "`data.frame`" Second data set.

`identity1`: Object of class "`factor`" True ID of records in `data1`

`identity2`: Object of class "`factor`" True ID of records in `data2`

Extends

Class "RLBigData", directly.

Methods

getColumnNames `signature(object = "RLBigDataLinkage")`

getExpectedSize `signature(object = "RLBigDataLinkage")`

See also RLBigData-class for inherited methods.

Author(s)

Andreas Borg

See Also

"RLBigData", RLBigDataLinkage

Examples

```
showClass("RLBigDataLinkage")
```

RLdata *Test data for Record Linkage*

Description

The RLdata tables contain artificial personal data for the evaluation of Record Linkage procedures. Some records have been duplicated with randomly generated errors. RLdata500 contains fifty duplicates, RLdata10000 thousand duplicates.

Usage

```
RLdata500
RLdata10000
identity.RLdata500
identity.RLdata10000
```

Format

RLdata500 and RLdata10000 are character matrices with 500 and 10000 records. Each row represents one record, with the following columns:

fname_c1 First name, first component

fname_c2 First name, second component

lname_c1 Last name, first component

lname_c2 Last name, second component

by Year of birth

bm Month of birth

bd Day of birth

identity.RLdata500 and identity.RLdata10000 are integer vectors representing the true record ids of the two data sets. Two records are duplicates, if and only if their corresponding values in the identity vector agree.

Author(s)

Andreas Borg, Murat Sariyar

Source

Generated with the data generation component of Febrl (Freely Extensible Biomedical Record Linkage), version 0.3 (https://sourceforge.net/projects/febrl/). The following data sources were used (all relate to Germany):

https://blog.beliebte-vornamen.de/2009/02/prozentuale-anteile-2008/, a list of the frequencies of the 20 most popular female names in 2008.

https://www.beliebte-vornamen.de/760-alle_jahre.htm, a list of the 100 most popular first names since 1890. The frequencies found in the source above were extrapolated to fit this list.

http://www.ahnenforschung-in-stormarn.de/geneal/nachnamen_100.htm, a list of the 100 most frequent family names with frequencies.

Age distribution as of Dec 31st, 2008, statistics of Statistisches Bundesamt Deutschland, taken from the GENESIS database (https://www-genesis.destatis.de/genesis/online/logon).

Web links as of August 2020.

RLResult-class *Class "RLResult"*

Description

A class that represents the result of a record linkage procedure with big data sets.

Objects from the Class

Objects from this class are created by the classification functions in this package, e.g. classifySupv. Directly creating instances by calling new is neither necessary nor recommended.

Slots

data: Object of class "RLBigData". The data set which was classified.

prediction: Object of class "ff". A vector with classification result for every record pair, coded by levels "N" for a non-link, "P" for a possible link and "L" for a link.

Methods

clone signature(object = "RLResult")

getDbFile signature(object = "RLResult")

getErrorMeasures signature(object = "RLResult")

getPairs signature(object = "RLResult")

getTable signature(object = "RLResult")

saveRLObject signature(object = "RLResult")

Note

The slot data uses a database to store data and create comparison patterns, thus assignment of a "RLResult" object to a different variable can lead to undesired results. Use clone to make a distinct copy. Similarly, the standard save mechanism does not work; saveRLObject and loadRLObject are provided to make objects persistent over different R sessions.

Author(s)

Andreas Borg, Murat Sariyar

See Also

classifySupv, emClassify and epiClassify create objects of this type.

show *Show a RLBigData object*

Description

Shows summarized information on a "RLBigData" object.

Usage

```
  ## S4 method for signature 'RLBigData'
show(object)
```

Arguments

object	The object for which to show a summary.

Details

The printed information consists of the type of linkage procedure (deduplication or linkage) and the number of records and the approximate number of record pairs as calculated by getExpectedSize. More information is obtained by the summary methods for these classes.

Value

show returns an invisible NULL and is used for its side effect.

Author(s)

Andreas Borg, Murat Sariyar

See Also

show

Examples

```
data(RLdata500)
rpairs <- RLBigDataDedup(RLdata500)
rpairs
```

splitData *Split Data*

Description

Splits a data set into two sets with desired proportions.

Usage

```
splitData(dataset, prop, keep.mprop = FALSE, num.non = 0, des.mprop = 0,
use.pred = FALSE)
```

Arguments

dataset	Object of class RecLinkData. Data pairs to split.
prop	Real number between 0 and 1. Proportion of data pairs to form the training set.
keep.mprop	Logical. Whether the ratio of matches should be retained.
num.non	Positive Integer. Desired number on non-matches in the training set.
des.mprop	Real number between 0 and 1. Desired proportion of matches to non-matches in the training set.
use.pred	Logical. Whether to apply match ratio to previous classification results instead of true matching status.

Value

A list of RecLinkData objects.

train	The sampled training data.
valid	All other record pairs

The sampled data are stored in the pairs attributes of train and valid. If present, the attributes prediction and Wdata are split and the corresponding values saved. All other attributes are copied to both data sets.

If the number of desired matches or non-matches is higher than the number actually present in the data, the maximum possible number is chosen and a warning issued.

Author(s)

Andreas Borg, Murat Sariyar

See Also

genSamples for generating training data based on unsupervised classification.

Examples

```
data(RLdata500)
pairs=compare.dedup(RLdata500, identity=identity.RLdata500,
  blockfld=list(1,3,5,6,7))

# split into halves, do not enforce match ratio
l=splitData(pairs, prop=0.5)
summary(l$train)
summary(l$valid)

# split into 1/3 and 2/3, retain match ration
l=splitData(pairs, prop=1/3, keep.mprop=TRUE)
summary(l$train)
summary(l$valid)

# generate a training set with 100 non-matches and 10 matches
l=splitData(pairs, num.non=100, des.mprop=0.1, keep.mprop=TRUE)
summary(l$train)
summary(l$valid)
```

---

stochastic *Stochastic record linkage.*

Description

Methods for stochastic record linkage following the framework of Fellegi and Sunter.

Usage

```
## S4 method for signature 'RecLinkData'
fsWeights(rpairs, m = 0.95, u = rpairs$frequencies, cutoff = 1)
## S4 method for signature 'RLBigData'
fsWeights(rpairs, m=0.95, u=getFrequencies(rpairs),
    cutoff=1, withProgressBar = (sink.number()==0))
## S4 method for signature 'RecLinkData'
fsClassify(rpairs, ...)
## S4 method for signature 'RLBigData'
fsClassify(rpairs, threshold.upper, threshold.lower=threshold.upper,
 m=0.95, u=getFrequencies(rpairs), withProgressBar = (sink.number()==0), cutoff=1)
```

Arguments

`rpairs`	The record pairs to be classified.
`threshold.upper`	A numeric value between 0 and 1.

threshold.lower
: A numeric value between 0 and 1 lower than threshold.upper.

m, u
: Numeric vectors. m- and u-probabilities of matching variables, see Details.

withProgressBar
: Logical. Whether to display a progress bar.

cutoff
: Numeric value. Threshold for converting string comparison values to binary values.

...
: Arguments passed to emClassify.

Details

These methods perform stochastic record linkage following the framework of Fellegi and Sunter (see reference).

fsWeights calculates matching weights on an object based on the specified m- and u-probabilities. Each of m and u can be a numeric vector or a single number in the range $[0, 1]$.

fsClassify performs classification based on the calculated weights. All record pairs with weights greater or equal threshold.upper are classified as links. Record pairs with weights smaller than threshold.upper and greater or equal threshold.lower are classified as possible links. All remaining records are classified as non-links.

The "RecLinkData" method is a shortcut for emClassify.

The "RLBigData" method checks if weights are present in the underlying database. If this is the case, classification is based on the existing weights. If not, weights are calculated on the fly during classification, but not stored. The latter behaviour might be preferable when a very large dataset is to be classified and disk space is limited. A progress bar is displayed only if weights are calculated on the fly and, by default, unless output is diverted by sink (e.g. in a Sweave script).

For a general introduction to weight based record linkage, see the vignette "Weight-based deduplication".

Value

fsWeights returns a copy of the object with the calculated weights added. Note that "RLBigData" objects have some reference-style semantics, see clone for more information.

For the "RecLinkData" method, fsClassify returns a S3 object of class "RecLinkResult" that represents a copy of newdata with element rpairs$prediction, which stores the classification result, as addendum.

For the "RLBigData" method, fsClassify returns a S4 object of class "RLResult".

Author(s)

Andreas Borg, Murat Sariyar

References

Ivan P. Fellegi, Alan B. Sunter: A Theory for Record Linkage, in: Journal of the American Statistical Association Vol. 64, No. 328 (Dec., 1969), pp. 1183–1210.

See Also

epiWeights

Examples

```
# generate record pairs
data(RLdata500)
rpairs <- compare.dedup(RLdata500, blockfld=list(1,3,5,6,7), identity=identity.RLdata500)

# calculate weights
rpairs <- fsWeights(rpairs)

# classify and show results
summary(fsClassify(rpairs,0))
```

---

strcmp *String Metrics*

Description

Functions for computation of the similarity between two strings.

Usage

```
jarowinkler(str1, str2, W_1=1/3, W_2=1/3, W_3=1/3, r=0.5)
levenshteinSim(str1, str2)
levenshteinDist(str1, str2)
```

Arguments

str1, str2	Two character vectors to compare.
W_1, W_2, W_3	Adjustable weights.
r	Maximum transposition radius. A fraction of the length of the shorter string.

Details

String metrics compute a similarity value in the range $[0, 1]$ for two strings, with 1 denoting the highest (usually equality) and 0 denoting the lowest degree of similarity. In the context of Record Linkage, string similarities can improve the discernibility between matches and non-matches.

jarowinkler is an implementation of the algorithm by Jaro and Winkler (see references). For the meaning of W_1, W_2, W_3 and r see the referenced article. For most applications, the default values are reasonable.

levenshteinDist returns the Levenshtein distance, which cannot be directly used as a valid string comparator.

levenshteinSim is a similarity function based on the Levenshtein distance, calculated by $1 - \frac{\mathrm{d}(str_1, str_2)}{\max(A,B))}$, where d is the Levenshtein distance function and A and B are the lengths of the strings.

Arguments `str1` and `str2` are expected to be of type `"character"`. Non-alphabetical characters can be processed. Valid format combinations for the arguments are:

- Two arrays with the same dimensions.
- Two vectors. The shorter one is recycled as necessary.

Value

A numeric vector with similarity values in the interval $[0, 1]$. For `levenshteinDist`, the edit distance as an integer vector.

Note

String comparison is case-sensitive, which means that for example `"R"` and `"r"` have a similarity of 0. If this behaviour is undesired, strings should be normalized before processing.

Author(s)

Andreas Borg, Murat Sariyar

References

Winkler, W.E.: String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. In: Proceedings of the Section on Survey Research Methods, American Statistical Association (1990), S. 354–369.

Examples

```
# compare two strings:
jarowinkler("Andreas","Anreas")
# compare one string with several others:
levenshteinSim("Andreas",c("Anreas","Andeas"))
# compare two vectors of strings:
jarowinkler(c("Andreas","Borg"),c("Andreas","Bork"))
```

---

subset *Subset operator for record linkage objects*

Description

Extracts a subset of a `"RecLinkData"` or `"RecLinkResult"` object.

Usage

```
  ## S3 method for class 'RecLinkData'
x[i]
  ## S3 method for class 'RecLinkResult'
x[i]
  ## S3 method for class 'RLBigData'
x[i]
  ## S3 method for class 'RLResult'
x[i]
```

Arguments

x	The object which to index.
i	Indices of pairs to include in the subset.

Value

A copy of x with only the pairs with indices specified by x.

Author(s)

Andreas Borg, Murat Sariyar

Examples

```
## Samples a subset of pairs

data(RLdata500)
rpairs <- compare.dedup(RLdata500, identity = identity.RLdata500,
  blockfld = list(1,3,5,6,7))
nPairs <- nrow(rpairs$pairs)
s <- sample(nPairs, nPairs / 2)
samp <- rpairs[s]
```

---

summary *Print Summary of Record Linkage Data*

Description

Prints information on RecLinkData and "RecLinkResult" objects.

Usage

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

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

Arguments

object	The object for which to print a summary.
...	Additional arguments from the generic, silently ignored.

Details

The printed information for RecLinkData objects includes:

- The number of records.
- The number of record pairs.
- The number of true matches, true non-matches and pairs with unknown status.
- If weights have been calculated for this object, a textual histogram of the weight distribution.

Information on "RecLinkResult" objects includes all of the above and the following:

- The number of detected links, non-links and possible links.
- The following error measures, if the true matching status of all record pairs is known: Alpha error (ratio of false links to matches), beta error (ratio of false non-links to non-matches) and accuracy (ratio of correctly classified pairs to the total number of pairs).
- A cross-classified table counting true matching status against classification. The true matching status is represented as logical values, possibly including NA for unknown status. Classification results are represented by:

  "L" for links,

  "P" for possible links

  "N" for non-links

Value

Returns an invisible NULL and is used for its side effect.

Author(s)

Andreas Borg

See Also

RecLinkData,"RecLinkResult"

summary.RLBigData *summary methods for* "RLBigData" *objects.*

Description

Shows summarized information on a "RLBigDataDedup" or "RLBigDataDedup" object.

Usage

```
  ## S3 method for class 'RLBigDataDedup'
summary(object, ...)
  ## S3 method for class 'RLBigDataLinkage'
summary(object, ...)
  ## S3 method for class 'summaryRLBigDataDedup'
print(x, ...)
  ## S3 method for class 'summaryRLBigDataLinkage'
print(x, ...)
```

Arguments

object	The object for which to show a summary.
x	Return value of the summary function.
...	Additional arguments from the generic function are ignored.

Details

The summary methods return a list of the format shown below. The print method displays this information on the console in a user-friendly format.

Blocking fields are displayed in a style like '[attr1], [attr2, attr3]', where 'attr1' etc. are column names and attributes within brackets represent one blocking iteration. See compare.dedup or RLBigDataDedup for an explanation of blocking criteria.

Value

For summary, a list with components

nData	Only for the "RLBigDataDedup" method: Number of records in the dataset.
nData1	Only for the "RLBigDataLinkage" method: Number of records in dataset 1.
nData2	Only for the "RLBigDataLinkage" method: Number of records in dataset 2.
attributes	Column names of dataset(s).
blockFld	Blocking definition as a list of character vectors, representing column names.
nPairs	Number of record pairs
nMatches	Number of matches in the set of record pairs.
nNonMatches	Number of non-matches in the set of record pairs.
nUnkonwn	Number of record pairs with unknown matching status.
weightHist	Only if weights have been calculated for object: a summary of the weights in histogram style.

Author(s)

Andreas Borg, Murat Sariyar

See Also

summary "RLBigData" RLBigDataDedup, RLBigDataLinkage

Examples

```
data(RLdata500)
rpairs <- RLBigDataDedup(RLdata500, identity = identity.RLdata500,
   blockfld=list(1,3,5:7))
rpairs <- epiWeights(rpairs)
summary(rpairs)
```

---

summary.RLResult *Summary method for* "RLResult" *objects.*

Description

Get summarized information on a "RLResult" object.

Usage

```
  ## S4 method for signature 'RLResult'
summary(object)
  ## S3 method for class 'summaryRLResult'
print(x, ...)
```

Arguments

object	The object for which to show a summary.
x	Return value of the summary function.
...	Additional arguments from the generic function are ignored.

Details

The summary methods return a list of the format shown below. The print method displays this information on the console in a user-friendly format.

Value

For summary, a list with components

nPairs	Number of record pairs.
nLinks	Number of detected links.
nPossibleLinks	Number of detected possible links.

Author(s)

Andreas Borg, Murat Sariyar

See Also

summary "RLResult"

Examples

```
data(RLdata500)
rpairs <- RLBigDataDedup(RLdata500, blockfld=list(1,3,5:7),
  identity = identity.RLdata500)
rpairs <- epiWeights(rpairs)
result <- epiClassify(rpairs, 0.7)
summary(result)
```

---

trainSupv *Train a Classifier*

Description

Trains a classifier for supervised classification of record pairs.

Usage

```
trainSupv(rpairs, method, use.pred = FALSE, omit.possible = TRUE,
  convert.na = TRUE, include.data = FALSE, ...)
```

Arguments

rpairs	Object of class RecLinkData. Training data.
method	A character vector. The classification method to use.
use.pred	Logical. Whether to use results of an unsupervised classification instead of true matching status.
omit.possible	Logical. Whether to remove pairs labeled as possible links or with unknown status.
convert.na	Logical. Whether to convert NAs to 0 in the comparison patterns.
include.data	Logical. Whether to include training data in the result object.
...	Further arguments to the training method.

Details

The given dataset is used as training data for a supervised classification. Either the true matching status has to be known for a sufficient number of data pairs or the data must have been classified previously, e.g. by using emClassify or classifyUnsup. In the latter case, argument use.pred has to be set to TRUE.

A classifying method has to be provided as a character string (factors are converted to character) through argument method. The supported classifiers are:

"svm" Support vector machine, see svm.

"rpart" Recursive partitioning tree, see rpart.

"ada" Stochastic boosting model, see ada.

"bagging" Bagging with classification trees, see bagging.

"nnet" Single-hidden-layer neural network, see nnet.

"bumping" A bootstrap based method using classification trees, see details.

Arguments in ... are passed to the corresponding function.

Most classifiers cannot handle NAs in the data, so by default these are converted to 0 before training.

By omit.possible = TRUE, possible links or pairs with unknown status are excluded from the training set. Setting this argument to FALSE allows three-class-classification (links, non-links and possible links), but the results tend to be poor.

Leaving include.data=FALSE saves memory, setting it to TRUE can be useful for saving the classificator while keeping track of the underlying training data.

BUMPING, (acronym for "Bootstrap umbrella of model parameters"), is an ensemble method described by *Tibshirani and Knight, 1999*. Such as in bagging, multiple classifiers are trained on bootstrap samples of the training set. The key difference is that not the aggregated decision of all classifiers (e.g. by majority vote) is used to classify new data, but only the single model that performs best on the whole training set. In combination with classification trees as underlying classifiers this approach allows good interpretability of the trained model while being more stable against outliers than traditionally induced decision trees. The number of bootstrap samples to use can be controlled by supplying the argument n.bootstrap, which defaults to 25.

Value

An object of class RecLinkClassif with the following components:

train	If include.data is TRUE, a copy of rpairs, otherwise an empty data frame with the same column names.
model	The model returned by the underlying training function.
method	A copy of the argument method.

Author(s)

Andreas Borg, Murat Sariyar

References

Tibshirani R, Knight K: Model search by bootstrap "bumping". Journal of Computational and Graphical Statistics 8(1999):671–686.

See Also

`classifySupv` for classifying with the trained model, `classifyUnsup` for unsupervised classification

Examples

```
# Train a rpart decision tree with additional parameter minsplit
data(RLdata500)
pairs=compare.dedup(RLdata500, identity=identity.RLdata500,
                    blockfld=list(1,3,5,6,7))
model=trainSupv(pairs, method="rpart", minsplit=5)
summary(model)
```

`unorderedPairs` *Create Unordered Pairs*

Description

Creates all unordered pairs of some objects or of the first `x` natural numbers.

Usage

```
unorderedPairs(x)
```

Arguments

`x`	Either an arbitrary vector of literals or a natural number

Details

If `x` has length one, all unordered pairs of the first `x` natural numbers are created. If `x` has more than one element, all unordered pairs of the elements of `x` are created.

Value

A matrix with two rows, each column holding one pair.

Author(s)

Andreas Borg

Examples

```
# create unordered pairs of {1,2,3}: {1,2},{1,3} and {2,3}
unorderedPairs(3)
# create unordered pairs of {"a","b","c"}: {"a","b"}, {"a","c"},{"b","c"}
unorderedPairs(c("a","b","c"))
```

`%append%-methods` *Concatenate comparison patterns or classification results*

Description

Combines two object of class "RecLinkData" or "RecLinkResult" by concatenating comparison patterns and, if available, weights and classification results.

Usage

```
x %append% y

## S4 method for signature 'RecLinkData,RecLinkData'
x %append% y

## S4 method for signature 'RecLinkResult,RecLinkResult'
x %append% y
```

Arguments

x, y	The objects to combine.

Value

An object with class corresponding to the input objects which represents the concatenation of `x` and `y`. Its component `pairs` is `rbind(x$pairs, y$pairs)`. If both `x` and `y` have weights stored in component `Wdata`, the result gets `c(x$Wdata, y$Wdata)` as component `Wdata`. For the "RecLinkResult" method, the result also includes the concatenation of the predicted classes in `x` and `y` as component `prediction`.

Note

The methods perform only a minimum of integrity checks, so the user has to make sure that the underlying data, the formats of comparison patterns (e.g. excluded columns) and the type of weights (method and parameters of weight calculation) match.

Author(s)

Andreas Borg, Murat Sariyar

Examples

```
data(RLdata500)
rpairs1=compare.dedup(RLdata500, blockfld=1, identity = identity.RLdata500)
rpairs2=compare.dedup(RLdata500, blockfld=3, identity = identity.RLdata500)

summary(rpairs1)
summary(rpairs2)
summary(rpairs1 %append% rpairs2)
```

Index