Package 'PPRL'

December 2, 2025

Type Package

Title Privacy Preserving Record Linkage

Version 0.3.9

Date 2025-12-02

Author Rainer Schnell [aut, cph],
Dorothea Rukasz [aut],
Christian Borgs [ctb],
Julian Reinhold [ctb],
Severin Weiand [ctb, cre],
Stefan Brumme [ctb] (HMAC, SHA256),
William B. Brogden [ctb] (Metaphone),
Tim O'Brien [ctb] (Metaphone),
Stephen Lacy [ctb] (Double Metaphone),
Apache Software Foundation [cph]

Maintainer Severin Weiand <mitarbeiter.schnell@uni-due.de>

Description A toolbox for deterministic, probabilistic and privacy-preserving record linkage techniques. Combines the functionality of the 'Merge Tool-
Box' (<https://www.record-linkage.de>) with current privacy-preserving techniques.

License GPL-3

Depends R (>= 3.1.0)

Imports Rcpp (>= 1.0.9), settings

LinkingTo Rcpp

RoxygenNote 7.3.3

NeedsCompilation yes

Encoding UTF-8

Repository CRAN

Date/Publication 2025-12-02 17:10:02 UTC

Contents

`PPRL-package` *PPRL: Privacy Preserving Record Linkage*

Description

A toolbox for deterministic, probabilistic and privacy-preserving record linkage techniques. Combines the functionality of the 'Merge ToolBox' (`https://www.record-linkage.de`) with current privacy-preserving techniques.

Author(s)

Maintainer: Severin Weiand <`mitarbeiter.schnell@uni-due.de`> [contributor]

Authors:

- Rainer Schnell [copyright holder]
- Dorothea Rukasz

Other contributors:

- Christian Borgs [contributor]
- Julian Reinhold [contributor]
- Stefan Brumme (HMAC, SHA256) [contributor]
- William B. Brogden (Metaphone) [contributor]

- Tim O'Brien (Metaphone) [contributor]
- Stephen Lacy (Double Metaphone) [contributor]
- Apache Software Foundation [copyright holder]

`CompareAS16` *Comparing bit vectors generated by CreateAS16*

Description

Comparing all elements of two vectors of records with each other using Armknechts & Schnells methods "create" and "compare".

Usage

```
CompareAS16(IDA, dataA, IDB, dataB, password, t = 0.85)
```

Arguments

`IDA`	A character vector or integer vector containing the IDs of the first data.frame.
`dataA`	A character vector containing the bit vectors that are to be created by Armknechts method "create".
`IDB`	A character vector or integer vector containing the IDs of the second data.frame.
`dataB`	A character vector containing the bit vectors that are to be created by Armknechts method "create".
`password`	A string containing the password used in the method "create".
`t`	A float containing the lower Tanimoto similarity threshold.

Details

Two bit vectors generated by `CreateAS16` are compared as described in the original publication.

Value

The function returns a data.frame with four columns containing all ID-pairs of all bit vectors, the estimated Tanimoto similarity and the classification (links/non-links).

Source

Armknecht, F., Schnell, R. (unpublished): Privacy Preserving Record Linkage Based on Bloom Filters and Codes. Working Paper.

See Also

`CreateAS16`

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Create Bloom Filter
testData <- CreateBF(ID = testData$V1, testData$V7, k = 20, padding = 1,
  q = 2, l = 1000, password = "(H]$6Uh*-Z204q")

# Optional: Create the new Bloom filter, the output of this function is just
# to see the created bit vectors, it is not the input of CompareAS16.
testAS <- CreateAS16(testData$ID, testData$CLKs, password = "khäuds")

# Compare bit vectcors, input is not the out put of CreateAS16,
# but the original Bloom Filters. CreateAS16 is executated in CompareAS16.
res <- CompareAS16(testData$ID, testData$CLKs, testData$ID, testData$CLKs,
  password = "khäuds", t = 0.85)
```

`Create581` *Create Encrypted Statistical Linkage Keys*

Description

Creates ESLs (also known as 581-Keys), which are the hashed combination of the full date of birth and sex and subsets of first and last names.

Usage

```
Create581(ID, data, code, password)
```

Arguments

`ID`	a character vector or integer vector containing the IDs of the data.frame.
`data`	a data.frame containig the data to be encoded.
`code`	a list indicating how data is to be encoded for each column. The list must have the same length as the number of columns of the data.frame to be encrypted.
`password`	a string used as a password for the HMAC.

Details

The original implementation uses the second and third position of the first name, the second, third and fifth position of the last name and full date of birth and sex as an input for an HMAC. This would be akin to using `code = list(c(2, 3), c(2, 3, 5), 0, 0)`. In this implementation, the positions of the subsets can be customized.

Value

A data.frame containing IDs and the corresponding Encrypted Statistical Linkage Keys.

Source

Karmel, R., Anderson, P., Gibson, D., Peut, A., Duckett, S., Wells, Y. (2010): Empirical aspects of record linkage across multiple data sets using statistical linkage keys: the experience of the PIAC cohort study. BMC Health Services Research 41(10).

See Also

CreateALC, StandardizeString

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Encrypt data
res <- Create581(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  code = list(0, 0, c(2, 3), c(2, 3, 5)),
  password = "(H]$6Uh*-Z204q")

# Code: 0 means the whole string is used,
# c(2, 3) means the second and third letter of the string is used
```

CreateALC *Anonymous Linkage Codes (ALCs)*

Description

Creates ALCs from clear-text data by creating soundex phonetics for first and last names and concatenating all other identifiers. The resulting code is encrypted using SHA-2. The user can decide on which columns the soundex phonetic is applied.

Usage

```
CreateALC(ID, data, soundex, password)
```

Arguments

ID	A character vector or integer vector containing the IDs of the data.frame.
data	a data.frame containing the data to be encoded.
soundex	a binary vector with one element for each input column, indicating whether soundex is to be used. 1 = soundex is used, 0 = soundex is not used. The soundex vector must have the same length as the number of columns the data.frame.
password	a string used as a password for the HMAC.

Value

A data.frame containing IDs and the corresponding Anonymous Linkage Codes.

Source

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

See Also

Create581, StandardizeString

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Encrypt data, use Soundex for names
res <- CreateALC(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  soundex = c(0, 0, 1, 1),
  password = "$6Uh*-Z204q")
```

CreateAS16 *Creating Records with Armknechts method create*

Description

This method generates a new bit vector out of an existing Bloom Filter. Building and comparisons are both possible with CompareAS16.

Usage

```
CreateAS16(ID, data, password)
```

Arguments

ID	A character vector or integer vector containing the IDs of the second data.frame.
data	A character vector containing the original bit vectors created by any Bloom Filter-based method.
password	A string containing the password to be used for both "create" and "compare".

Value

A character vector containing bit vectors created as described in the original publication.

Source

Armknecht, F., Schnell, R. (unpublished): Privacy Preserving Record Linkage Based on Bloom Filters and Codes. Working Paper.

See Also

CompareAS16, CreateBF, CreateCLK

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Create Bloom Filter
testData <- CreateBF(ID = testData$V1, testData$V7, k = 20, padding = 1,
  q = 2, l = 1000, password = "(H]$6Uh*-Z204q")

# Create the new Bloom Filter
testAS <- CreateAS16(testData$ID, testData$CLKs, password = "khäuds")
```

CreateBalancedBF *Balanced Bloom Filter Encoding*

Description

Creates CLKs with constant Hamming weights by adding a negated copy of the binary input vector which is then permutated.

Usage

```
CreateBalancedBF(ID, data, password)
```

Arguments

ID	A character vector or integer vector containing the IDs of the data.frame.
data	Bit vectors as created by any Bloom filter-based method.
password	a string used as a password for the random permutation.

Value

A data.frame containing IDs and the corresponding Balanced Bloom Filter.

References

Berger, J. M. (1961): A Note on Error Detection Codes for Asymmetric Channels. In: Information and Control 4: 68–73.

Knuth, Donald E. (1986): Efficient Balanced Codes. In: IEEE Transactions on Information Theory IT-32 (1): 51–53.

Schnell, R., Borgs, C. (2016): Randomized Response and Balanced Bloom Filters for Privacy Preserving Record Linkage. IEEE International Conference on Data Mining (ICDM 2016), Barcelona.

See Also

CreateBF, CreateBitFlippingBF, CreateCLK, CreateDoubleBalancedBF, CreateEnsembleCLK, CreateMarkovCLK, CreateRecordLevelBF, StandardizeString

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Create bit vectors e.g. with CreateBF
testData <- CreateBF(ID = testData$V1,
  testData$V7, k = 20, padding = 1, q = 2,
  l = 1000, password = "(H]$6Uh*-Z204q")

# Create Balanced Bloom Filters
BB <- CreateBalancedBF(ID = testData$ID, data = testData$CLKs,
  password = "hdayfkgh")
```

CreateBF *Bloom Filter Encoding*

Description

Creates Bloom filters for each row of the input data by splitting the input into q-grams which are hashed into a bit vector.

Usage

```
CreateBF(ID, data, password, k = 20, padding = 1, qgram = 2, lenBloom = 1000)
```

Arguments

ID	a character vector or integer vector containing the IDs of the data.frame.
data	a character vector containing the data to be encoded. Make sure the input vectors are not factors.
password	a string used as a password for the random hashing of the q-grams.
k	number of bit positions set to one for each bigram.
padding	integer (0 or 1) indicating if string padding is to be used.
qgram	integer (1 or 2) indicating whether to split the input strings into bigrams (q = 2) or unigrams (q = 1).
lenBloom	desired length of the Bloom filter in bits.

Value

A data.frame containing IDs and the corresponding bit vector.

Source

Schnell, R., Bachteler, T., Reiher, J. (2009): Privacy-preserving record linkage using Bloom filters. BMC Medical Informatics and Decision Making 9: 41.

See Also

CreateCLK, StandardizeString

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Encode data
BF <- CreateBF(ID = testData$V1, data = testData$V7,
  k = 20, padding = 1, q = 2, l = 1000,
  password = "(H]$6Uh*-Z204q")
```

`CreateBitFlippingBF` *Permanent Randomized Response Bloom Filters*

Description

Applies Permanent Randomized Response to flip bits of the bit vectors given.

Usage

```
CreateBitFlippingBF(data, password, f)
```

Arguments

`data`	a data.frame containing the IDs in the first column and bit vectors created by any Bloom filter-based method in the second column.
`password`	a string to seed the random bit flipping.
`f`	a numeric between 0 and 1 giving the probability of flipping a bit.

Details

The randomized response technique is used on each bit position B[i] of a Bloom filter B. B[i] is set to one or zero with a probability of $1/2 * f$ for each outcome. The bit position remains unchanged with a probability of $1 - f$, where $0 <= f <= 1$.

Value

A data.frame containing IDs and the corresponding bit vector.

Source

Schnell, R., Borgs, C. (2016): Randomized Response and Balanced Bloom Filters for Privacy Preserving Record Linkage. IEEE International Conference on Data Mining (ICDM 2016), Barcelona.

See Also

`CreateBF`, `CreateCLK`, `StandardizeString`

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <-read.csv(testFile, head = FALSE,
  sep = "\t", colClasses = "character")

## Encode data into Bloom Filters
BF <- CreateBF(ID = testData$V1, data = testData$V7,
  k = 20, padding = 1, q = 2, l = 1000,
  password = "(H]$6Uh*-Z204q" )
```

```
# Create Permanent Randomized Response Bloom Filter
RR <- CreateBitFlippingBF(BF, password = "l+kfdj1J", f = 0.1)
```

CreateCLK *Cryptographic Long-term Keys (CLKs)*

Description

Each column of the input data.frame is hashed into a single additive Bloom filter.

Usage

```
CreateCLK(ID, data, password, k = 20,
  padding =  as.integer(c(0)),
  qgram =  as.integer(c(2)), lenBloom = 1000)
```

Arguments

ID	A character vector or integer vector containing the IDs of the data.frame.
data	a data.frame containing the data to be encoded. Make sure the input vectors are not factors.
password	a character vector with a password for each column of the input data.frame for the random hashing of the q-grams.
k	number of bit positions set to one for each bigram.
padding	integer vector (0 or 1) indicating if string padding is to be used on the columns of the input. The padding vector must have the same size as the number of columns of the input data.
qgram	integer vector (1 or 2) indicating whether to split the input strings into bigrams (q = 2) or unigrams (q = 1). The qgram vector must have the same size as the number of columns of the input data.
lenBloom	desired length of the final Bloom filter in bits.

Value

A data.frame containing IDs and the corresponding bit vector.

Source

Schnell, R. (2014): An efficient Privacy-Preserving Record Linkage Technique for Administrative Data and Censuses. Journal of the International Association for Official Statistics (IAOS) 30: 263-270.

See Also

CreateBF, StandardizeString

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <-read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

## Encode data
CLK <- CreateCLK(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  k = 20, padding = c(0, 0, 1, 1),
  q = c(1, 1, 2, 2), l = 1000,
  password = c("HUh4q", "lkjg", "klh", "Klk5"))
```

CreateDoubleBalancedBF

Double Balanced Bloomfilter Encoding

Description

Double balanced Bloom Filter are created by first creating balanced Bloom Filters, see CreateBalancedBF, negating the whole data set and shuffling each Bloom Filter.

Usage

```
CreateDoubleBalancedBF(ID, data, password)
```

Arguments

ID	A character vector containing the ID. The ID vector must have the same size as the number of rows of data.
data	CLKs as created by CreateCLK or CreateBF.
password	A string to encode the routines.

Value

A data.frame containing IDs and the corresponding double balanced bit vector.

References

Schnell, R. (2017): Recent Developments in Bloom Filter-based Methods for Privacy-preserving Record Linkage. Curtin Institute for Computation, Curtin University, Perth, 12.9.2017.

See Also

CreateBalancedBF, CreateBF, CreateCLK, StandardizeString

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <-read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Create bit vectors e.g. with CreateBF
testData <- CreateBF(ID = testData$V1,
  testData$V7, k = 20, padding = 1, q = 2,
  l = 1000, password = "(H]$6Uh*-Z204q")

# Create Double Balanced Bloom Filters
DBB <- CreateDoubleBalancedBF(ID = testData$ID, data = testData$CLKs,
  password = "hdayfkgh")
```

`CreateEnsembleCLK` *Combine multiple independent CLKs using a simple majority rule*

Description

Creates multiple CLKs which are combined using a simple majority rule.

Usage

```
CreateEnsembleCLK(ID, data, password, NumberOfCLK = 1 , k = 20,
  padding = as.integer(c(0)), qgram = as.integer(c(2)),
  lenBloom = 1000)
```

Arguments

`ID`	A character vector or integer vector containing the IDs of the data.frame.
`data`	a data.frame containing the data to be encoded. Make sure the input vectors are not factors.
`password`	a character vector with a password for each column of the input data.frame for the random hashing of the q-grams.
`NumberOfCLK`	number of independent CLKs to be built.
`k`	number of bit positions set to one for each bigram.
`padding`	integer vector (0 or 1) indicating if string padding is to be used on the columns of the input. The padding vector must have the same size as the number of columns of the input data.
`qgram`	integer vector (1 or 2) indicating whether to split the input strings into bigrams (q = 2) or unigrams (q = 1). The qgram vector must have the same size as the number of columns of the input data.
`lenBloom`	desired length of the final Bloom filter in bits.

Details

Creates a set number of independent CLKs for each record of the input data.frame and combines them using a simple majority rule. The bit positions $B[i]$ in the final CLK of the length B are set to $B[i] = 1$ if more than half of the independent CLKs bit positions $B[i]$ have a value of one. Otherwise the final bit position is zero.

Value

A data.frame containing IDs and the corresponding ensemble bit vector.

References

Kuncheva, L. (2014): Combining Pattern Classifiers: Methods and Algorithms. Wiley.

See Also

CreateBF, CreateCLK, StandardizeString

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <-read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")
## Not run:
# Create Ensemble CLK
EnsembleCLK <- CreateEnsembleCLK(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  k = 20, padding = c(0, 0, 1, 1),
  q = c(1, 2, 2, 2), l = 1000,
  password = c("HUh4q", "lkjg", "klh", "Klk5"),
  NumberOfCLK = 5)

## End(Not run)
```

CreateMarkovCLK *Create CLKs with Markov Chain-based transition matrix*

Description

Builds CLKs encoding additional bigrams based on the transition probabilities as estimated by a Markov Chain.

Usage

```
CreateMarkovCLK(ID, data, password, markovTable, k1 = 20, k2 = 4,
  padding = as.integer(c(0)), qgram = as.integer(c(2)),
  lenBloom = 1000,  includeOriginalBigram = TRUE, v = FALSE)
```

Arguments

ID	a character vector or integer vector containing the IDs of the data.frame.
data	a data.frame containing the data to be encoded. Make sure the input vectors are not factors.
password	a character vector with a password for each column of the input data.frame for the random hashing of the q-grams.
markovTable	a numeric matrix containing the transition probabilities for all bigrams possible.
k1	number of bit positions set to one for each bigram.
k2	number of additional bigrams drawn for each original bigram.
padding	integer vector (0 or 1) indicating if string padding is to be used on the columns of the input. The padding vector must have the same size as the number of columns of the input data.
qgram	integer vector (1 or 2) indicating whether to split the input strings into bigrams (q = 2) or unigrams (q = 1). The qgram vector must have the same size as the number of columns of the input data.
lenBloom	desired length of the final Bloom filter in bits.
includeOriginalBigram	by default, the original bigram is encoded together with the additional bigrams. Set this to FALSE to include only the additional bigrams to further increase the security.
v	verbose.

Details

A transition matrix for all possible bigrams is built using a function to fit a markov chain distribution using a Laplacian smoother. A transition matrix built for bigrams using the NC Voter Data is included in the package. For each original bigram in the data, k2 new bigrams are drawn according to their follow-up probability as given by the transition matrix. The final bigram set is then encoded following CreateCLK.

Value

A data.frame containing IDs and the corresponding bit vector.

References

Schnell R., Borgs C. (2017): Using Markov Chains for Hardening Bloom Filter Encryptions against Cryptographic Attacks in Privacy Preserving Record Linkage. German Record Linkage Center Working Paper.

Schnell, R. (2017): Recent Developments in Bloom Filter-based Methods for Privacy-preserving Record Linkage. Curtin Institute for Computation, Curtin University, Perth, 12.9.2017.

See Also

CreateCLK, StandardizeString

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <-read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

## Not run:
# Load example Markov chain matrix (created from NC Voter Data)
markovFile <-file.path(path.package("PPRL"), "extdata/TestMatrize.csv")
markovData <-read.csv(markovFile,  sep = " ",
  header = TRUE, check.names = FALSE)
markovData <- as.matrix(markovData)

# Create Markov CLK using
CLKMarkov <- CreateMarkovCLK(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  markovTable = markovData,
  k1 = 20, k2 = 4, l = 1000,
  padding = c(0, 0, 1, 1),
  q = c(1, 2, 2, 2),
  password = c("(H]$6Uh*-Z204q", "lkjg", "klh", "KJHklk5"),
  includeOriginalBigram = TRUE)

## End(Not run)
```

`CreateRecordLevelBF` *Record Level Bloom Filter (RLBF) Encoding*

Description

Creates Record Level Bloom filters, combining single Bloom filters into a singular bit vector.

Usage

```
CreateRecordLevelBF(ID, data, password, lenRLBF = 1000, k = 20,
                        padding = as.integer(c(0)),
                        qgram = as.integer(c(2)),
                        lenBloom = as.integer(c(500)),
                        method = "StaticUniform",
                        weigths = as.numeric(c(1)))
```

Arguments

`ID`	a character vector or integer vector containing the IDs of the data.frame.
`data`	a character vector containing the data to be encoded. Make sure the input vectors are not factors.
`password`	a string used as a password for the random hashing of the q-grams and the shuffling.

lenRLBF	length of the final Bloom filter.
lenBloom	an integer vector containing the length of the first level Bloom filters which are to be combined. For the methods "StaticUniform" and "StaticWeighted", a single integer is required, since all original Bloom filters will have the same length.
k	number of bit positions set to one for each q-gram.
padding	integer (0 or 1) indicating if string padding is to be used.
qgram	integer (1 or 2) indicating whether to split the input strings into bigrams (q = 2) or unigrams (q = 1).
method	any of either "StaticUniform", "StaticWeighted", "DynamicUniform" or "DynamicWeighted" (see details).
weigths	weigths are used for the "StaticWeighted" and "DynamicWeighted" methods. The weights vector must have the same length as number of columns in the input data. The sum of the weights must be 1.

Details

Single Bloom filters are first built for every variable in the input data.frame. Combining the Bloom filters is done by sampling a set fraction of the original Bloom filters and concatenating the samples. The result is then shuffled. The sampling can be done using four different weighting methods:

1. StaticUniform
2. StaticWeighted
3. DynamicUniform
4. DynamicWeighted

Details are described in the original publication.

Value

A data.frame containing IDs and the corresponding bit vector.

Source

Durham, E. A. (2012). A framework for accurate, efficient private record linkage. Dissertation. Vanderbilt University.

See Also

CreateBF, CreateCLK,

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
## Not run:
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")
```

```
# StaticUniform
RLBF <- CreateRecordLevelBF(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  lenRLBF = 1000, k = 20,
  padding = c(0, 0, 1, 1), qgram = c(1, 1, 2, 2),
  lenBloom = 500,
  password = c("(H]$6Uh*-Z204q", "lkjg", "klh", "KJHkälk5"),
  method = "StaticUniform")

# StaticWeighted
RLBF <- CreateRecordLevelBF(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  lenRLBF = 1000, k = 20,
  padding = c(0, 0, 1, 1), qgram = c(1, 1, 2, 2),
  lenBloom = 500,
  password = c("(H]$6Uh*-Z204q", "lkjg", "klh", "KJHkälk5"),
  method = "StaticWeighted", weigths = c(0.1, 0.1, 0.5, 0.3))

# DynamicUniform
RLBF <- CreateRecordLevelBF(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  lenRLBF = 1000, k = 20,
  padding = c(0, 0, 1, 1), qgram = c(1, 1, 2, 2),
  lenBloom = c(300, 400, 550, 500),
  password = c("(H]$6Uh*-Z204q", "lkjg", "klh", "KJHkälk5"),
  method = "DynamicUniform")

# DynamicWeighted
RLBF <- CreateRecordLevelBF(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  lenRLBF = 1000, k = 20,
  padding = c(0, 0, 1, 1), qgram = c(1, 1, 2, 2),
  lenBloom = c(300, 400, 550, 500),
  password = c("(H]$6Uh*-Z204q", "lkjg", "klh", "KJHkälk5"),
  method = "DynamicWeighted", weigths = c(0.1, 0.1, 0.5, 0.3))

## End(Not run)
```

`DeterministicLinkage` *Deterministic Record Linkage*

Description

Deterministic Record Linkage of two data sets giving results enabling rule-based methods.

Usage

```
DeterministicLinkage(IDA, dataA, IDB, dataB, blocking = NULL, similarity)
```

Arguments

IDA	A character vector or integer vector containing the IDs of the first data.frame.
dataA	A data.frame containing the data to be linked and all linking variables as specified in SelectBlockingFunction and SelectSimilarityFunction.
IDB	A character vector or integer vector containing the IDs of the second data.frame.
dataB	A data.frame containing the data to be linked and all linking variables as specified in SelectBlockingFunction and SelectSimilarityFunction.
blocking	Optional blocking variables. See SelectBlockingFunction.
similarity	Variables used for linking and their respective linkage methods as specified in SelectSimilarityFunction.

Details

To call the Deterministic Linkage function it is necessary to set up linking variables and methods. Using blocking variables is optional. Further options are available in SelectBlockingFunction and SelectSimilarityFunction.

Value

A data.frame containing ID-pairs and the link status for each linking variable. This way, rules can be put into place allowing the user to classify links and non-links.

Source

Christen, P. (2012): Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer.

Schnell, R., Bachteler, T., Reiher, J. (2004): A toolbox for record linkage. Austrian Journal of Statistics 33(1-2): 125-133.

See Also

ProbabilisticLinkage, SelectBlockingFunction, SelectSimilarityFunction, StandardizeString

Examples

```
# load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# define year of birth (V3) as blocking variable
bl <- SelectBlockingFunction("V3", "V3", method = "exact")

# Select first name and last name as linking variables,
# to be linked using the soundex phonetic (first name)
# and exact matching (last name)
l1 <- SelectSimilarityFunction("V7", "V7", method = "Soundex")
l2 <- SelectSimilarityFunction("V8", "V8", method = "exact")
```

```
# Link the data as specified in bl and l1/l2
# (in this small example data is linked to itself)
res <- DeterministicLinkage(testData$V1, testData,
  testData$V1, testData, similarity = c(l1, l2), blocking = bl)
```

`ElegantPairingInt` *Elegant Pairing*

Description

Unordered Pairing Function creating a new unique integer from two input integers.

Usage

```
ElegantPairingInt(int1, int2)
```

Arguments

`int1`	first integer to be paired.
`int2`	second integer to be paired.

Details

With two of non-negative integers x and y as an input, the pairing is computed as:

$$elegantPairing(x, y) = (x * y) + floor((|x - y| - 1)^2)/4)$$

The function is commutative. x and y have to be non-negative integers.

Value

The function outputs a single non-negative integer that is uniquely associated with that unordered pair.

Source

Szudzik, M. (2006): An Elegant Pairing Function. Wolfram Science Conference NKS 2006.

See Also

`ElegantPairingVec`

Examples

```
ElegantPairingInt(2, 3)
```

ElegantPairingVec *Simple Pairing Function*

Description

Unordered Pairing Function creating a new unique integer from two input integers in a data.frame.

Usage

```
ElegantPairingVec (ID, data)
```

Arguments

ID	A character vector or integer vector containing the IDs of the data.frame.
data	a data.frame consisting of two columns containing the integers on which the pairing function is to be applied.

Details

With two of non-negative integers x and y as an input, the pairing is computed as:

$$elegantPairing(x, y) = (x * y) + floor((|x - y| - 1)^2)/4)$$

The function is commutative. x and y have to be non-negative integers. The function outputs a single non-negative integer that is uniquely associated with that unordered pair.

Value

A data.frame containing IDs and the computed integer.

Source

Szudzik, M. (2006): An Elegant Pairing Function. Wolfram Science Conference NKS 2006.

See Also

ElegantPairingInt

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Create numeric data frame of day and month of birth
dataInt <- data.frame(as.integer(testData$V4), as.integer(testData$V5))
```

```
# Use unordered pairing on day and month
res <- ElegantPairingVec(testData$V1, dataInt)
```

ProbabilisticLinkage *Probabilistic Record Linkage*

Description

Probabilistic Record Linkage of two data sets using distance-based or probabilistic methods.

Usage

```
ProbabilisticLinkage(IDA, dataA, IDB, dataB,  blocking = NULL, similarity)
```

Arguments

IDA	A character vector or integer vector containing the IDs of the first data.frame.
dataA	A data.frame containing the data to be linked and all linking variables as specified in SelectBlockingFunction and SelectSimilarityFunction.
IDB	A character vector or integer vector containing the IDs of the second data.frame.
dataB	A data.frame containing the data to be linked and all linking variables as specified in SelectBlockingFunction and SelectSimilarityFunction.
blocking	Optional blocking variables. See SelectBlockingFunction.
similarity	Variables used for linking and their respective linkage methods as specified in SelectSimilarityFunction.

Details

To call the Probabilistic Linkage function it is necessary to set up linking variables and methods. Using blocking variables is optional. Further options are available in SelectBlockingFunction and SelectSimilarityFunction. Using this method, the Fellegi-Sunter model is used, with the EM algorithm estimating the weights (Winkler 1988).

Value

A data.frame containing pairs of IDs, their corresponding similarity value and the match status as determined by the linkage procedure.

Source

Christen, P. (2012): Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer.

Schnell, R., Bachteler, T., Reiher, J. (2004): A toolbox for record linkage. Austrian Journal of Statistics 33(1-2): 125-133.

Winkler, W. E. (1988): Using the EM algorithm for weight computation in the Fellegi-Sunter model of record linkage. Proceedings of the Section on Survey Research Methods Vol. 667, American Statistical Association: 671.

See Also

CreateBF, CreateCLK, DeterministicLinkage, SelectBlockingFunction, SelectSimilarityFunction, StandardizeString

Examples

```
# load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# define year of birth (V3) as blocking variable
bl <- SelectBlockingFunction("V3", "V3", method = "exact")

# Select first name and last name as linking variables,
# to be linked using the Jaro-Winkler similarity measure (first name)
# and levenshtein distance (last name)
l1 <- SelectSimilarityFunction("V7", "V7", method = "jw")
l2 <- SelectSimilarityFunction("V8", "V8", method = "lv")

# Link the data as specified in bl and l1/l2
# (in this small example data is linked to itself)
res <- ProbabilisticLinkage(testData$V1, testData,
  testData$V1, testData, similarity = c(l1, l2), blocking = bl)
```

`SelectBlockingFunction`

Select blocking method prior to linkage

Description

Before calling ProbabilisticLinkage or DeterministicLinkage, a blocking method can be selected. For each blocking variable desired, the function call has to be repeated.

Usage

```
SelectBlockingFunction(variable1, variable2, method)
```

Arguments

variable1	Column name of blocking variable 1.
variable2	Column name of blocking variable 2.
method	Desired blocking method. Possible values are 'exact' and 'exactCL'.

Details

The following methods are available for blocking:

Simple exact blocking. All records with the same values for the blocking variable create a block. Searching for links is only done within these blocks.

`'exactCL'` The same as `'exact'`. Only works with strings; all caracters are capitalised.

References

Christen, P. (2012): Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer.

See Also

DeterministicLinkage, ProbabilisticLinkage, SelectSimilarityFunction, StandardizeString

Examples

```
# load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# define year of birth (V3) as blocking variable
bl <- SelectBlockingFunction("V3", "V3", method = "exact")

# Select first name and last name as linking variables,
# to be linked using the Jaro-Winkler similarity measure (first name)
# and levenshtein distance (last name)
l1 <- SelectSimilarityFunction("V7", "V7", method = "jw")
l2 <- SelectSimilarityFunction("V8", "V8", method = "lv")

# Link the data as specified in bl and l1/l2
# (in this small example data is linked to itself)
res <- ProbabilisticLinkage(testData$V1, testData,
  testData$V1, testData, similarity = c(l1, l2), blocking = bl)
```

SelectSimilarityFunction

Select Similarity Function for Linkage

Description

To call DeterministicLinkage or ProbabilisticLinkage it is mandatory to select a similarity function for each variable. Each element of the setup contains the two variable names and the method. For some methods further informations can be entered.

Usage

```
SelectSimilarityFunction(variable1, variable2,
  method = "jw",
  ind_c0 = FALSE, ind_c1 = FALSE,
  m = 0.9, u = 0.1, p = 0.05, epsilon = 0.0004,
  lower = 0.0, upper = 0.0,
  threshold = 0.85, jaroWeightFactor = 1.0, lenNgram = 2)
```

Arguments

variable1	name of linking variable 1 in the data.frame. The column must be of type character, numeric or integer, containing the data to be merged. The data vector must have the same length as the ID vector.
variable2	name of linking variable 2 in the data.frame. The column must be of type character, numeric or integer, containing the data to be merged. The data vector must have the same length as the ID vector.
method	linking method. Possible values are: • 'exact' = Exact matching • 'exactCL' = Exact matching using capital letters • 'LCS' = Longest Common Subsequence • 'lv' = Levenshtein distance • 'dl' = Damerau Levenshtein distance • 'jaro' = Jaro similarity • 'jw' = Jaro-Winkler similarity • 'jw2' = Modified Jaro-Winkler similarity • 'ngram' = n-gram similarity • 'Gcp' = German census phonetic (Baystat) • 'Reth' = Reth-Schek (IBM) phonetic • 'Soundex' = Soundex phonetic • 'Metaphone' = Metaphone phonetic • 'DoubleMetaphone' = Double Metaphone phonetic
ind_c0	Only used for jw2. Increase the probability of a match when the number of matched characters is large. This option allows for a little more tolerance when the strings are large. It is not an appropriate test when comparing fixed length fields such as phone and social security numbers. A nonzero value indicates the option is deactivated.
ind_c1	Only used for jw2. All lower case characters are converted to upper case prior to the comparison. Disabling this feature means that the lower case string "code" will not be recognized as the same as the upper case string "CODE". Also, the adjustment for similar characters section only applies to uppercase characters. A nonzero value indicates the option is deactivated.
m	Initial m value for the EM algorithm. Only used when linking using ProbabilisticLinkage. $0.0 < m < 1.0$.

u — Initial u value for the EM algorithm. Only used when linking using ProbabilisticLinkage. 0.0 < u < 1.0.

p — Initial p value for the EM algorithm. Only used when linking using ProbabilisticLinkage. 0.0 < u < 1.0.

epsilon — epsilon is a stop criterum for the EM algorithm. The EM algorithm can be terminated when relative change of likelihood logarithm is less than epsilon. Only used when linking using ProbabilisticLinkage.

lower — Matches lower than 'lower' are classified as non-match. Everything between 'lower' and 'upper' is classified as possible match. Only used when linking using ProbabilisticLinkage.

upper — Matches higher than 'upper' are classified as match. Everything between 'lower' and 'upper' is classified as possible match. Only used when linking using ProbabilisticLinkage.

threshold — If using string similarities: Outputs only matches above the similarity threshold value. If using string distances: Outputs only matches below the set threshold distance.

jaroWeightFactor — By the Jaro weight adjustment the matching weight is adjusted according to the degree of similarity between the variable values. The weight factor which determines the Jaro adjusted matching weight. Only used when linking using ProbabilisticLinkage.

lenNgram — Length of ngrams. Only used for the method ngram. Length of ngrams must be between 1 and 4.

Value

Calling the function will not return anything.

References

Christen, P. (2012): Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer.

Schnell, R., Bachteler, T., Reiher, J. (2004): A toolbox for record linkage. Austrian Journal of Statistics 33(1-2): 125-133.

Winkler, W. E. (1988): Using the EM algorithm for weight computation in the Fellegi-Sunter model of record linkage. Proceedings of the Section on Survey Research Methods Vol. 667, American Statistical Association: 671.

See Also

DeterministicLinkage, ProbabilisticLinkage, SelectBlockingFunction, StandardizeString

Examples

```
# load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
```

```
  colClasses = "character")

# define year of birth (V3) as blocking variable
bl <- SelectBlockingFunction("V3", "V3", method = "exact")

# Select first name and last name as linking variables,
# to be linked using the jaro-winkler (first name)
# and exact matching (last name)
l1 <- SelectSimilarityFunction("V7","V7", method = "jw",
  ind_c0 = FALSE, ind_c1 = FALSE , m = 0.9, u = 0.1,
  lower = 0.0, upper = 0.0)
l2 <- SelectSimilarityFunction("V8","V8", method = "exact")

# Link the data as specified in bl and l1/l2
# (in this small example data is linked to itself)
res <- ProbabilisticLinkage(testData$V1, testData,
  testData$V1, testData, similarity = c(l1, l2), blocking = bl)
```

`StandardizeString` *Standardize String*

Description

Preprocessing (cleaning) of strings prior to linkage.

Usage

```
StandardizeString(strings)
```

Arguments

`strings`	A character vector of strings to be standardized.

Details

Strings are capitalized, letters are substituted as described below. Leading and trailing blanks are removed. Other non-ASCII characters are deleted.

- Replace "Æ" with "AE"
- Replace "æ" with "AE"
- Replace "Ä" with "AE"
- Replace "ä" with "AE"
- Replace "Å" with "A"
- Replace "å" with "A"
- Replace "Â" with "A"
- Replace "â" with "A"

- Replace "À" with "A"
- Replace "à" with "A"
- Replace "Á" with "A"
- Replace "á" with "A"
- Replace "Ç" with "C"
- Replace "Ç" with "C"
- Replace "Ê" with "E"
- Replace "ê" with "E"
- Replace "È" with "E"
- Replace "è" with "E"
- Replace "É" with "E"
- Replace "é" with "E"
- Replace "Ï" with "I"
- Replace "ï" with "I"
- Replace "Î" with "I"
- Replace "î" with "I"
- Replace "Ì" with "I"
- Replace "ì" with "I"
- Replace "Í" with "I"
- Replace "í" with "I"
- Replace "Ö" with "OE"
- Replace "ö" with "OE"
- Replace "Ø" with "O"
- Replace "ø" with "O"
- Replace "Ô" with "O"
- Replace "ô" with "O"
- Replace "Ò" with "O"
- Replace "ò" with "O"
- Replace "Ó" with "O"
- Replace "ó" with "O"
- Replace "ß" with "SS"
- Replace "Ş" with "S"
- Replace "ş" with "S"
- Replace "ü" with "UE"
- Replace "Ü" with "UE"
- Replace "Ů" with "U"
- Replace "Û" with "U"
- Replace "û" with "U"
- Replace "Ù" with "U"
- Replace "ù" with "U"

Value

Returns a character vector with standardized strings.

Examples

```
strings = c("Päter", " Jürgen", " Roß")
StandardizeString(strings)
```

`WolframRule30` *Apply Wolframs rule 30 on bit vectors*

Description

Apply Wolframs Cellular Automaton rule 30 on the input bit vectors.

Usage

```
WolframRule30(ID, data, lenBloom, t)
```

Arguments

`ID`	IDs as character vector.
`data`	character vector containing bit vectors.
`lenBloom`	length of Bloom filters.
`t`	indicates how often rule 30 is to be used.

Value

Returns a character vector with new bit vectors after rule 30 has been applied t times.

References

https://en.wikipedia.org/wiki/Rule_30

Schnell, R. (2017): Recent Developments in Bloom Filter-based Methods for Privacy-preserving Record Linkage. Curtin Institute for Computation, Curtin University, Perth, 12.9.2017.

Wolfram, S. (1983): Statistical mechanics of cellular automata. Rev. Mod. Phys. 55 (3): 601–644.

See Also

`WolframRule90`

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Create bit vector e.g. by CreateCLK or CreateBF
CLK <- CreateCLK(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  k = 20, padding = c(0, 0, 1, 1),
  q = c(1, 1, 2, 2), l = 1000,
  password = c("HUh4q", "lkjg", "klh", "Klk5"))

# Apply rule 30 once
res <- WolframRule30(CLK$ID, CLK$CLK, lenBloom = 1000, t = 1)
```

`WolframRule90` *Apply Wolframs rule 90 on bit vectors*

Description

Apply Wolframs Cellular Automaton rule 90 on the input bit vectors.

Usage

```
WolframRule90(ID, data,  lenBloom, t)
```

Arguments

`ID`	IDs as character vector.
`data`	character vector containing bit vectors.
`lenBloom`	length of Bloom filters.
`t`	indicates how often rule 90 is to be used.

Value

Returns a character vector with new bit vectors after rule 90 has been applied t times.

References

https://en.wikipedia.org/wiki/Rule_90

Martin, O., Odlyzko, A. M., Wolfram, S. (1984): Algebraic properties of cellular automata. Communications in Mathematical Physics, 93 (2): 219-258.

Schnell, R. (2017): Recent Developments in Bloom Filter-based Methods for Privacy-preserving Record Linkage. Curtin Institute for Computation, Curtin University, Perth, 12.9.2017.

Wolfram, S. (1983): Statistical mechanics of cellular automata. Rev. Mod. Phys. 55 (3): 601–644.

See Also

WolframRule30

Examples

```
# Load test data
testFile <- file.path(path.package("PPRL"), "extdata/testdata.csv")
testData <- read.csv(testFile, head = FALSE, sep = "\t",
  colClasses = "character")

# Create bit vector e.g. by CreateCLK or CreateBF
CLK <- CreateCLK(ID = testData$V1,
  data = testData[, c(2, 3, 7, 8)],
  k = 20, padding = c(0, 0, 1, 1),
  q = c(1, 1, 2, 2), l = 1000,
  password = c("HUh4q", "lkjg", "klh", "Klk5"))

# Apply rule 90 once
res <- WolframRule90(CLK$ID, CLK$CLK, lenBloom = 1000, t = 1)
```

Index