Package 'PGRdup'

July 27, 2020

Title Discover Probable Duplicates in Plant Genetic Resources Collections

Version 0.2.3.6

Description Provides functions to aid the identification of probable/possible duplicates in Plant Genetic Resources (PGR) collections using 'passport databases' comprising of information records of each constituent sample. These include methods for cleaning the data, creation of a searchable Key Word in Context (KWIC) index of keywords associated with sample records and the identification of nearly identical records with similar information by fuzzy, phonetic and semantic matching of keywords.

Depends R (>= 3.0.2)

Imports data.table (>= 1.9.3), igraph, stringdist (>= 0.9.4), stringi, ggplot2, grid, gridExtra, methods, utils, stats

Suggests diagram, wordcloud, microbenchmark, XML, knitr, rmarkdown, pander

License GPL-2 | GPL-3

Encoding latin1

LazyData true

VignetteBuilder knitr

RoxygenNote 7.1.1

URL https://cran.r-project.org/package=PGRdup, https://github.com/aravind-j/PGRdup, https://doi.org/10.5281/zenodo.841963, https://aravind-j.github.io/PGRdup/, https://www.rdocumentation.org/packages/PGRdup

BugReports https://github.com/aravind-j/PGRdup/issues

NeedsCompilation yes

Author J. Aravind [aut, cre] (<https://orcid.org/0000-0002-4791-442X>),
J. Radhamani [aut],
Kalyani Srinivasan [aut],
B. Ananda Subhash [aut],
Rishi Kumar Tyagi [aut],
ICAR-NBGPR [cph] (www.nbpgr.ernet.in),
Maurice Aubrey [ctb] (Double Metaphone),
Kevin Atkinson [ctb] (Double Metaphone),
Lawrence Philips [ctb] (Double Metaphone)

Maintainer J. Aravind <j.aravind@icar.gov.in>

Repository CRAN

Date/Publication 2020-07-27 06:40:11 UTC

R topics documented:

PGRdup-package *The PGRdup Package*

Description

Functions to facilitate genebank managers in the identification of probable duplicate accessions from plant genetic resources (PGR) passport databases.

Author(s)

J Aravind <aravindj@nbpgr.ernet.in>
J Radhamani <radhamani@nbpgr.ernet.in>
Kalyani Srinivasan <kalyani@nbpgr.ernet.in>
B Ananda Subhash <anandasubhash@gmail.com>
RK Tyagi <rktyagi@nbpgr.ernet.in>

AddProbDup *Add probable duplicate sets fields to the PGR passport database*

Description

AddProbDup adds the fuzzy, phonetic and semantic probable duplicates sets data fields from an object of class ProbDup to the original PGR passport database.

Usage

```
AddProbDup(pdup, db, addto = c("I", "II"), max.count = 30)
```

Arguments

pdup	An object of class ProbDup.
db	A data frame of the PGR passport database.
addto	Either "I" or "II" indicating the database to which the data.fields are to be added (see **Details**).
max.count	The maximum count of probable duplicate sets whose information is to be retrieved.

Details

This function helps to add information associated with identified fuzzy, phonetic and semantic probable duplicate sets using the ProbDup function to the original PGR passport database. Associated data fields such as SET_NO, ID and IDKW are added based on the PRIM_ID field(column).

In case more than one KWIC index was used to generate the object of class ProbDup, the argument addto can be used to specify to which database the data fields are to be added. The default "I" indicates the database from which the first KWIC index was created and "II" indicates the database from which the second index was created.

Value

A data frame of the PGR passport database with the probable duplicate sets fields added.

Note

When any primary ID/key records in the fuzzy, phonetic or semantic duplicate sets are found to be missing from the original database db, then they are ignored and only the matching records are considered for adding the information with a warning.

This may be due to data standardization of the primary ID/key field using the function DataClean before creation of the KWIC index and subsequent identification of probable duplicate sets. In such a case, it is recommended to use an identical data standardization operation on the database db before running this function.

See Also

DataClean, KWIC, ProbDup

Examples

```
## Not run:

#' # Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
          "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
          "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
          "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
          "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
          "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
```

```
                phonetic = TRUE, encoding = "primary",
                semantic = TRUE, syn = syn)

# Add the duplicates sets to the original database
GNwithdup <-  AddProbDup(pdup = GNdup, db = GN1000, addto = "I")


## End(Not run)
```

DataClean *Clean PGR passport data*

Description

DataClean cleans the data in a character vector according to the conditions in the arguments.

Usage

```
DataClean(
  x,
  fix.comma = TRUE,
  fix.semcol = TRUE,
  fix.col = TRUE,
  fix.bracket = TRUE,
  fix.punct = TRUE,
  fix.space = TRUE,
  fix.sep = TRUE,
  fix.leadzero = TRUE
)
```

Arguments

x	A character vector. If not, coerced to character by as.character.
fix.comma	logical. If TRUE, all the commas are replaced by space (see **Details**).
fix.semcol	logical. If TRUE, all the semicolons are replaced by space (see **Details**).
fix.col	logical. If TRUE, all the colons are replaced by space (see **Details**).
fix.bracket	logical. If TRUE, all the brackets are replaced by space (see **Details**).
fix.punct	logical. If TRUE, all punctuation characters are removed (see **Details**).
fix.space	logical. If TRUE, all space characters are replaced by space and multiple spaces are converted to single space (see **Details**).
fix.sep	logical. If TRUE, space between alphabetic characters followed by digits is removed (see **Details**).
fix.leadzero	logical. If TRUE, leading zeros are removed (see **Details**).

Details

This function aids in standardization and preparation of the PGR passport data for creation of a KWIC index with KWIC function and the identification of probable duplicate accessions by the ProbDup function. It cleans the character strings in passport data fields(columns) specified as the input character vector x according to the conditions in the arguments in the same order. If the input vector x is not of type character, it is coerced to a character vector.

This function is designed particularly for use with fields corresponding to accession names such as accession ids, collection numbers, accession names etc. It is essentially a wrapper around the gsub base function with regex arguments. It also converts all strings to upper case and removes leading and trailing spaces.

Commas, semicolons and colons which are sometimes used to separate multiple strings or names within the same field can be replaced with a single space using the logical arguments fix.comma, fix.semcol and fix.col respectively.

Similarly the logical argument fix.bracket can be used to replace all brackets including parenthesis, square brackets and curly brackets with space.

The logical argument fix.punct can be used to remove all punctuation from the data.

fix.space can be used to convert all space characters such as tab, newline, vertical tab, form feed and carriage return to spaces and finally convert multiple spaces to single space.

fix.sep can be used to merge together accession identifiers composed of alphabetic characters separated from as series of digits by a space character. For example IR 64, PUSA 256 etc.

fix.leadzero can be used to remove leading zeros from accession name fields to facilitate matching to identify probable duplicates. e.g. IR0064 -> IR64

Value

A character vector with the cleaned data converted to upper case. NAs if any are converted to blank strings.

See Also

gsub, regex, MergeKW, KWIC, ProbDup

Examples

```
names <- c("S7-12-6", "ICG-3505", "U 4-47-18;EC 21127", "AH 6481", "RS   1",
           "AK 12-24", "2-5 (NRCG-4053)", "T78, Mwitunde", "ICG 3410",
           "#648-4 (Gwalior)", "TG4;U/4/47/13", "EC0021003")
DataClean(names)
```

DisProbDup *Get disjoint probable duplicate sets*

Description

DisProbDup finds and joins intersecting sets in an object of class ProbDup to get disjoint probable duplicate sets.

Usage

```
DisProbDup(pdup, combine = c("F", "P", "S"))
```

Arguments

pdup	An object of class ProbDup.
combine	A character vector indicating the type of sets to be considered together for retrieving disjoint sets. If NULL, then disjoint sets within each type are retrieved (see **Details**).

Details

This function considers the accession primary keys/IDs for finding intersecting sets and subsequently joins them to retrieve disjoint sets. These operations are implemented utilizing the igraph package functions.

Disjoint sets are retrieved either individually for each type of probable duplicate sets or considering all type of sets simultaneously. In case of the latter, the disjoint of all the type of sets alone are returned in the output as an additional data frame DisjointDuplicates in an object of class ProbDup

Value

Returns an object of class ProbDup with either the disjoint sets within each type - FuzzyDuplicates, PhoneticDuplicates and SemanticDuplicates when combine = NULL or the combined disjoint duplicate sets as an additional element DisjointDupicates according to the choice specified in the argument combine.

See Also

ProbDup

Examples

```
## Not run:

# Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
```

```
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
         "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
         "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
         "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
         "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
         "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
                 phonetic = TRUE, encoding = "primary",
                 semantic = TRUE, syn = syn)
lapply(GNdup, dim)

# Get disjoint probable duplicate sets of each kind
disGNdup1 <- DisProbDup(GNdup, combine = NULL)
lapply(disGNdup1, nrow)

# Get disjoint probable duplicate sets combining all the kinds of sets
disGNdup2 <- DisProbDup(GNdup, combine = c("F", "P", "S"))
lapply(disGNdup2, nrow)


## End(Not run)
```

DoubleMetaphone *'Double Metaphone' phonetic algorithm*

Description

DoubleMetaphone converts strings to double metaphone phonetic codes.

Usage

```
DoubleMetaphone(str)
```

Arguments

str	A character vector whose strings are to be encoded by double metaphone algorithm.

Details

An implementation of the Double Metaphone phonetic algorithm in R. If non-ASCII characters encountered in the input character vector str, a warning is issued and they are transliterated so that the accented characters are converted to their ASCII unaccented versions.

Value

Returns a list with two character vectors of the same length as the input vector. The first character vector contains the primary double metaphone encodings, while the second character vector contains the alternate encodings.

Acknowledgement

The C code for the double metaphone algorithm was adapted from Maurice Aubrey's perl module hosted at the **gitpan/Text-DoubleMetaphone** public github library along with the corresponding license information.

Note

In case of non-ASCII characters in strings, a warning is issued and accented characters are converted to their ASCII unaccented versions.

References

Philips, Lawrence. 2000. "The Double Metaphone Search Algorithm." *C/C++ Users Journal* 18 (6): 38-43. http://dl.acm.org/citation.cfm?id=349124.349132.

See Also

phonetic, phonetics

Examples

```
# Return the primary and secondary Double Metaphone encodings for a character vector.
str1 <- c("Jyothi", "Jyoti")
str2 <- c("POLLACHI", "BOLLACHI")
DoubleMetaphone(str1)
DoubleMetaphone(str2)
## Not run:
# Issue a warning in case of non-ASCII characters.
str3 <- c("J\xf5geva", "Jogeva")
```

```
DoubleMetaphone(str3)
## End(Not run)
```

GN1000 *Sample groundnut PGR passport data*

Description

Sample PGR passport data of 1000 groundnut accessions held in the Indian National Genebank at National Bureau of Plant Genetic Resources (NBPGR), New Delhi.

Usage

```
data(GN1000)
```

Format

A data frame having 1000 records with the following 10 columns(fields):

- CommonName : Common name
- BotanicalName : Botanical name
- NationalID : NBPGR National identifier
- CollNo : Collector number
- DonorID : Donor ID
- OtherID1 : Other ID field 1
- OtherID2 : Other ID field 2
- BioStatus : Biological status
- SourceCountry : Country of origin
- TransferYear : Year of transfer

See Also

http://www.nbpgr.ernet.in:8080/PGRPortal/

KWCounts *Generate keyword counts*

Description

KWCounts generates keyword counts from PGR passport database fields(columns).

Usage

```
KWCounts(x, fields, excep)
```

Arguments

x	A data frame.
fields	A character vector with the names of fields(columns) of the data frame from which KWIC index is to be generated. The first field is considered as the primary key or identifier (see **Details**).
excep	A vector of the keywords not to be considered for the counts (see **Details**).

Details

This function computes the keyword counts from PGR passport database fields(columns) specified in the fields argument. The first field is considered as the primary key or identifier and is not used for counting the keywords. Any strings given in the excep argument are ignored for generating the counts.

The keyword counts can give a rough indication of the completeness of the data in the database fields being used for identification of probable duplicates.

Value

A data frame with the keyword counts for each record.

Note

For large number of exceptions and/or large data.frame computation of keyword counts may take some time.

See Also

stri_count

Examples

```
# Load PGR passport database
GN <- GN1000

# Specify database fields to use as a vector
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
          "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
          "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
          "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
          "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
          "U", "VALENCIA", "VIRGINIA", "WHITE")

# Compute the keyword counts
GNKWCouts <- KWCounts(GN, GNfields, exep)

# Plot the keyword counts
bp <- barplot(table(GNKWCouts$COUNT),
              xlab = "Word count", ylab = "Frequency", col = "#CD5555")
text(bp, 0, table(GNKWCouts$COUNT),cex=1,pos=3)
```

KWIC *Create a KWIC index*

Description

KWIC creates a Keyword in Context index from PGR passport database fields.

Usage

```
KWIC(x, fields, min.freq = 10)
```

Arguments

x	A data frame from which KWIC index is to be generated.
fields	A character vector with the names of fields(columns) of the data frame from which KWIC index is to be generated. The first field is considered as the primary key or identifier (see **Details**).
min.freq	Frequency of keywords are not computed if below min.freq. Default is 10.

Details

The function generates a Keyword in Context index from a data frame of a PGR passport database based on the fields(columns) stated in the arguments, using data.table package.

The first element of vector fields is considered as the primary key or identifier which uniquely identifies all rows in the data frame.

Cleaning of the data the input fields(columns) using the DataClean function with appropriate arguments is suggested before running this function.

Value

A list of class KWIC containing the following components:

KWIC	The KWIC index in the form of a data frame.
KeywordFreq	A data frame of the keywords detected with frequency greater than min.freq.
Fields	A character vector with the names of the PGR database fields from which the keywords were extracted.

References

Knüpffer, H. 1988. "The European Barley Database of the ECP/GR: An Introduction." *Die Kulturpflanze* 36 (1): 135-62. doi:https://doi.org/10.1007/BF01986957.

Knüpffer, H., L. Frese, and M. W. M. Jongen. 1997. "Using Central Crop Databases: Searching for Duplicates and Gaps." In *Central Crop Databases: Tools for Plant Genetic Resources Management. Report of a Workshop, Budapest, Hungary, 13-16 October 1996*, edited by E. Lipman, M. W. M. Jongen, T. J. L. van Hintum, T. Gass, and L. Maggioni, 67-77. Rome, Italy and Wageningen, The Netherlands: International Plant Genetic Resources Institute and Centre for Genetic Resources.

See Also

data.table, print.KWIC

Examples

```
# Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))

## Not run:

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)
GNKWIC

# Retrieve the KWIC index from the KWIC object
KWIC <- GNKWIC[[1]]

# Retrieve the keyword frequencies from the KWIC object
KeywordFreq <- GNKWIC[[2]]

# Show error in case of duplicates and NULL values
# in the primary key/ID field "NationalID"
```

```
GN[1001:1005,] <- GN[1:5,]
GN[1001,3] <- ""
GNKWIC <- KWIC(GN, GNfields)

## End(Not run)
```

MergeKW *Merge keyword strings*

Description

These functions merge keyword strings separated by delimiters such as space, period or dash in a character vector into single keyword strings.

Usage

```
MergeKW(x, y, delim = c("space", "dash", "period"))

MergePrefix(x, y, delim = c("space", "dash", "period"))

MergeSuffix(x, y, delim = c("space", "dash", "period"))
```

Arguments

x	A character vector. If not, coerced to character by `as.character`.
y	A list of character vectors with pairs of strings that are to be merged (for `MergeKW`) or a character vector of strings which are to be merged to succeeding string (for `MergePrefix`) or the preceding string (for `MergeSuffix`). If not of type character, coerced by `as.character`.
delim	Delimiting characters to be removed between keywords.

Details

These functions aid in standardization of relevant data fields(columns) in PGR passport data for creation of a KWIC index with `KWIC` function and subsequent identification of probable duplicate accessions by the `ProbDup` function.

It is recommended to run this function before using the `DataClean` function on the relevant data fields(columns) of PGR passport databases.

`MergeKW` merges together pairs of strings specified as a list in argument `y` wherever they exist in a character vector. The second string in the pair is merged even when it is followed by a number.

`MergePrefix` merges prefix strings specified as a character vector in argument `y` to the succeeding root word, wherever they exist in a character vector.

`MergeSuffix` merges suffix strings specified as a character vector in argument `y` to the preceding root word, wherever they exist in a character vector. The suffix strings which are followed by numbers are also merged.

Value

A character vector of the same length as x with the required keyword strings merged.

See Also

DataClean, KWIC, ProbDup

Examples

```
names <- c("Punjab Bold", "Gujarat- Dwarf", "Nagpur.local", "SAM COL 144",
           "SAM COL--280", "NIZAMABAD-LOCAL", "Dark Green Mutant",
           "Dixie-Giant", "Georgia- Bunch", "Uganda-erect", "Small Japan",
           "Castle  Cary", "Punjab erect", "Improved small japan",
           "Dark Purple")

# Merge pairs of strings
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
           c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
           c("Mota", "Company"))
names <- MergeKW(names, y1, delim = c("space", "dash", "period"))

# Merge prefix strings
y2 <- c("Light", "Small", "Improved", "Punjab", "SAM")
names <- MergePrefix(names, y2, delim = c("space", "dash", "period"))

# Merge suffix strings
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
names <- MergeSuffix(names, y3, delim = c("space", "dash", "period"))
```

MergeProbDup *Merge two objects of class* ProbDup

Description

MergeProbDup merges two objects of class ProbDup into a single one.

Usage

```
MergeProbDup(pdup1, pdup2)
```

Arguments

pdup1	An object of class ProbDup.
pdup2	An object of class ProbDup.

Value

An object of class ProbDup with the merged list of fuzzy, phonetic and semantic probable duplicate sets.

See Also

ProbDup, SplitProbDup

Examples

```
## Not run:
#' # Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
         "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
         "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
         "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
         "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
         "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
                 phonetic = TRUE, encoding = "primary",
                 semantic = TRUE, syn = syn)

# Split the probable duplicate sets
GNdupSplit <- SplitProbDup(GNdup, splitat = c(10, 10, 10))

# Merge the split sets
GNdupMerged <- MergeProbDup(GNdupSplit[[1]], GNdupSplit[[3]])


## End(Not run)
```

ParseProbDup *Parse an object of class* ProbDup *to a data frame.*

Description

ParseProbDup converts an object of class ProbDup to a data frame for export.

Usage

```
ParseProbDup(pdup, max.count = 30, insert.blanks = TRUE)
```

Arguments

pdup	An object of class ProbDup.
max.count	The maximum count of probable duplicate sets which are to be parsed to a data frame.
insert.blanks	logical. If TRUE, inserts a row of NAs after each set.

Value

A data frame of the long/narrow form of the probable duplicate sets data with the following core columns:

SET_NO	The set number.
TYPE	The type of probable duplicate set. 'F' for fuzzy, 'P' for phonetic and 'S' for semantic matching sets.
K	The KWIC index or database of origin of the record. The method is specified within the square brackets in the colu
PRIM_ID	The primary ID of the accession record from which the set could be identified.
IDKW	The 'matching' keywords along with the IDs.
COUNT	The number of elements in a set.

For the retrieved columns(fields) the prefix K* indicates the KWIC index of origin.

See Also

ProbDup,

Examples

```
## Not run:

#' # Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")
```

```
# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
          "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
          "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
          "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
          "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
          "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
                 phonetic = TRUE, encoding = "primary",
                 semantic = TRUE, syn = syn)

# Convert to data frame of sets
GNdupParsed <- ParseProbDup(GNdup)


## End(Not run)
```

`print.KWIC` *Prints summary of* `KWIC` *object.*

Description

`print.KWIC` prints to console the summary of an object of class `KWIC` including the database fields(columns) used, the total number of keywords and the number of distinct keywords in the index.

Usage

```
## S3 method for class 'KWIC'
print(x, ...)
```

Arguments

x	An object of class KWIC.
...	Unused

See Also

KWIC

---

print.ProbDup *Prints summary of* ProbDup *object.*

Description

print.ProbDup prints to console the summary of an object of class ProbDup including the method used ("a", "b" or "c"), the database fields(columns) considered, the number of probable duplicate sets of each kind along with the corresponding number of records.

Usage

```
## S3 method for class 'ProbDup'
print(x, ...)
```

Arguments

x	An object of class ProbDup.
...	Unused

See Also

ProbDup

ProbDup *Identify probable duplicates of accessions*

Description

ProbDup identifies probable duplicates of germplasm accessions in KWIC indexes created from PGR passport databases using fuzzy, phonetic and semantic matching strategies.

Usage

```
ProbDup(
  kwic1,
  kwic2 = NULL,
  method = c("a", "b", "c"),
  excep = NULL,
  chunksize = 1000,
  useBytes = TRUE,
  fuzzy = TRUE,
  max.dist = 3,
  force.exact = TRUE,
  max.alpha = 4,
  max.digit = Inf,
  phonetic = TRUE,
  encoding = c("primary", "alternate"),
  phon.min.alpha = 5,
  min.enc = 3,
  semantic = FALSE,
  syn = NULL
)
```

Arguments

kwic1	An object of class KWIC.
kwic2	An object of class KWIC. Required for method "b" and "c" only (see **Details**).
method	The method to be followed for identification of probable duplicates. Either "a", "b" or "c". (see **Details**).
excep	A vector of the keywords in KWIC not to be used for probable duplicate search (see **Details**).
chunksize	A value indicating the size of KWIC index keyword block to be used for searching for matches at a time in case of large number of keywords(see **Note**).
useBytes	logical. If TRUE, performs byte-wise comparison instead of character-wise comparison (see **Note**).
fuzzy	logical. If TRUE identifies probable duplicates based on fuzzy matching.
max.dist	The maximum levenshtein distance between keyword strings allowed for a match. Default is 3 (see **Details**).
force.exact	logical. If TRUE, enforces exact matching instead of fuzzy matching for keyword strings which match the criteria specified in arguments max.alpha and max.digit (see **Details**).
max.alpha	Maximum number of alphabet characters present in a keyword string up to which exact matching is enforced rather than fuzzy matching. Default is 4 (see **Details**).
max.digit	Maximum number of numeric characters present in a keyword string up to which exact matching is enforced rather than fuzzy matching. Default is Inf (see **Details**).

phonetic	logical. If TRUE identifies probable duplicates based on phonetic matching.
encoding	Double metaphone encoding for phonetic matching. The default is "primary" (see **Details**).
phon.min.alpha	Minimum number of alphabet characters to be present in a keyword string for phonetic matching (see **Details**).
min.enc	Minimum number of characters to be be present in double metaphone encoding of a keyword string for phonetic matching (see **Details**).
semantic	logical. If TRUE identifies probable duplicates based on semantic matching.
syn	A list with character vectors of synsets (see **Details**).

Details

This function performs fuzzy, phonetic and semantic matching of keywords in KWIC indexes of PGR passport databases (created using KWIC function) to identify probable duplicates of germplasm accessions. The function can execute matching according to either of the following three methods as specified by the method argument.

Method a: Perform string matching of keywords in a single KWIC index to identify probable duplicates of accessions in a single PGR passport database.

Method b: Perform string matching of keywords in the first KWIC index (query) with that of the keywords in the second index (source) to identify probable duplicates of accessions of the first PGR passport database among the accessions in the second database.

Method c: Perform string matching of keywords in two different KWIC indexes jointly to identify probable duplicates of accessions from among two PGR passport databases.

Fuzzy matching or approximate string matching of keywords is carried out by computing the generalized levenshtein (edit) distance between them. This distance measure counts the number of deletions, insertions and substitutions necessary to turn one string to the another. A distance of up to max.dist are considered for a match.

Exact matching will be enforced when the argument force.exact is TRUE. It can be used to avoid fuzzy matching when the number of alphabet characters in keywords is lesser than a critical value (max.alpha). Similarly, the value of max.digit can also be set according to the requirements. The default value of Inf avoids fuzzy matching and enforces exact matching for all keywords having any numerical characters. If max.digit and max.alpha are both set to Inf, exact matching will be enforced for all the keywords.

When exact matching is enforced, for keywords having both alphabet and numeric characters and with the number of alphabet characters greater than max.digit, matching will be carried out separately for alphabet and numeric characters present.

Phonetic matching of keywords is carried out using the Double Metaphone phonetic algorithm (DoubleMetaphone) to identify keywords that have the similar pronunciation. Either the primary or alternate encodings can be used by specifying the encoding argument. The argument phon.min.alpha sets the limits for the number of alphabet characters to be present in a string for executing phonetic matching. Similarly min.enc sets the limits for the number of characters to be present in the encoding of a keyword for phonetic matching.

Semantic matching matches keywords based on a list of accession name synonyms supplied as list with character vectors of synonym sets (synsets) to the syn argument. Synonyms in this context

refers to interchangeable identifiers or names by which an accession is recognized. Multiple keywords specified as members of the same synset in syn are merged together. To facilitate accurate identification of synonyms from the KWIC index, identical data standardization operations using the MergeKW and DataClean functions for both the original database fields and the synset list are recommended.

The probable duplicate sets identified initially here may be intersecting with other sets. To get the disjoint sets after the union of all the intersecting sets use the DisProbDup function.

The function AddProbDup can be used to add the information associated with the identified sets in an object of class ProbDup as fields(columns) to the original PGR passport database.

All of the string matching operations here are executed through the stringdist-package functions.

Value

A list of class ProbDup containing the following data frames of probable duplicate sets identified along with the corresponding keywords and set counts:

1. FuzzyDuplicates
2. PhoneticDuplicates
3. SemanticDuplicates

Each data frame has the following columns:

SET_NO	The set number.
TYPE	The type of probable duplicate set. 'F' for fuzzy, 'P' for phonetic and 'S' for semantic matching sets.
ID	The primary IDs of records of accessions comprising a set.
ID:KW	The 'matching' keywords along with the IDs.
COUNT	The number of elements in a set.

The prefix [K*] indicates the KWIC index of origin of the KEYWORD or PRIM_ID.

Note

As the number of keywords in the KWIC indexes increases, the memory consumption by the function also increases. For string matching, this function relies upon creation of a n*m matrix of all possible keyword pairs for comparison, where n and m are the number of keywords in the query and source indexes respectively. This can lead to cannot allocate vector of size errors in case very large KWIC indexes where the comparison matrix is too large to reside in memory. In such a case, try to adjust the chunksize argument to get the appropriate size of the KWIC index keyword block to be used for searching for matches at a time. However a smaller chunksize may lead to longer computation time due to the memory-time trade-off.

The progress of matching is displayed in the console as number of blocks completed out of total (e.g. 6 / 30), the percentage of achievement (e.g. 30%) and a text-based progress bar.

In case of multi-byte characters in keywords, the matching speed is further dependent upon the useBytes argument as described in **Encoding issues** for the stringdist function, which is made use of here for string matching.

References

van der Loo, M. P. J. 2014. "The Stringdist Package for Approximate String Matching." *R Journal* 6 (1):111-22. https://journal.r-project.org/archive/2014/RJ-2014-011/index.html.

See Also

KWIC, DoubleMetaphone stringdistmatrix, adist, print.ProbDup

Examples

```
## Not run:

# Method "a"
#===========

# Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
          "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
          "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
          "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
          "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
          "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
                 phonetic = TRUE, encoding = "primary",
                 semantic = TRUE, syn = syn)
GNdup
```

```
# Method "b and c"
#=================

# Load PGR passport databases
GN1 <- GN1000[!grepl("^ICG", GN1000$DonorID), ]
GN1$DonorID <- NULL
GN2 <- GN1000[grepl("^ICG", GN1000$DonorID), ]
GN2 <- GN2[!grepl("S", GN2$DonorID), ]
GN2$NationalID <- NULL

# Specify as a vector the database fields to be used
GN1fields <- c("NationalID", "CollNo", "OtherID1", "OtherID2")
GN2fields <- c("DonorID", "CollNo", "OtherID1", "OtherID2")

# Clean the data
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) DataClean(x))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
       "Bunch", "Peanut")
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Remove duplicated DonorID records in GN2
GN2 <- GN2[!duplicated(GN2$DonorID), ]

# Generate KWIC index
GN1KWIC <- KWIC(GN1, GN1fields)
GN2KWIC <- KWIC(GN2, GN2fields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
         "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
         "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
         "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
         "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
         "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdupb <- ProbDup(kwic1 = GN1KWIC, kwic2 = GN2KWIC, method = "b",
                  excep = exep, fuzzy = TRUE, phonetic = TRUE,
                  encoding = "primary", semantic = TRUE, syn = syn)
```

```
GNdupb

GNdupc <- ProbDup(kwic1 = GN1KWIC, kwic2 = GN2KWIC, method = "c",
                  excep = exep, fuzzy = TRUE, phonetic = TRUE,
                  encoding = "primary", semantic = TRUE, syn = syn)
GNdupc


## End(Not run)
```

`read.genesys` *Convert 'Darwin Core - Germplasm' zip archive to a flat file*

Description

`read.genesys` reads PGR data in a Darwin Core - germplasm zip archive downloaded from genesys database and creates a flat file `data.frame` from it.

Usage

```
read.genesys(zip.genesys, scrub.names.space = TRUE, readme = TRUE)
```

Arguments

`zip.genesys`	A character vector giving the file path to the downloaded zip file from Genesys.
`scrub.names.space`	logical. If `TRUE`, all space characters are removed from name field in names extension (see **Details**).
`readme`	logical. If `TRUE`, the genesys zip file readme is printed to console.

Details

This function helps to import to R environment, the PGR data downloaded from genesys database https://www.genesys-pgr.org/ as a Darwin Core - germplasm (DwC-germplasm) zip archive. The different csv files in the archive are merged as a flat file into a single `data.frame`.

All the space characters can be removed from the fields corresponding to accession names such as acceNumb, collNumb, ACCENAME, COLLNUMB, DONORNUMB and OTHERNUMB using the argument `scrub.names.space` to facilitate creation of KWIC index with `KWIC` function and subsequent matching operations to identify probable duplicates with `ProbDup` function.

The argument `readme` can be used to print the readme file in the archive to console, if required.

Value

A data.frame with the flat file form of the genesys data.

See Also

`data.table`

Examples

```
## Not run:
# Import the DwC-Germplasm zip archive "genesys-accessions-filtered.zip"
PGRgenesys <- read.genesys("genesys-accessions-filtered.zip",
                           scrub.names.space = TRUE, readme = TRUE)

## End(Not run)
```

ReconstructProbDup *Reconstruct an object of class ProbDup*

Description

ReconstructProbDup reconstructs a data frame of probable duplicate sets created using the function ReviewProbDup and subjected to manual clerical review, back into an object of class ProbDup.

Usage

```
ReconstructProbDup(rev)
```

Arguments

rev	A data frame with the the core columns(fields) SET_NO, TYPE, K, PRIM_ID, DEL, SPLIT, COUNT and IDKW

Details

A data frame created using the function ReviewProbDup from an object of class ProbDup for manual clerical review of identified probable duplicate sets can be reconstituted back to the same object after the review using this function. The instructions for modifying the sets entered in the appropriate format in the columns DEL and SPLIT during clerical review are taken into account for reconstituting the probable duplicate sets.

Any records with Y in column DEL are deleted and records with identical integers in the column SPLIT other than the default 0 are reassembled into a new set.

Value

An object of class ProbDup with the modified fuzzy, phonetic and semantic probable duplicate sets according to the instructions specified under clerical review.

See Also

ProbDup, ReviewProbDup

Examples

```
## Not run:

# Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
          "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
          "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
          "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
          "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
          "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
                 phonetic = TRUE, encoding = "primary",
                 semantic = TRUE, syn = syn)

# Get disjoint probable duplicate sets of each kind
disGNdup <- DisProbDup(GNdup, combine = NULL)

# Get the data frame for reviewing the duplicate sets identified
RevGNdup <- ReviewProbDup(pdup = disGNdup, db1 = GN1000,
                          extra.db1 = c("SourceCountry", "TransferYear"),
                          max.count = 30, insert.blanks = TRUE)
# Examine and review the duplicate sets using edit function
RevGNdup <- edit(RevGNdup)

# Examine and make changes to a set
subset(RevGNdup, SET_NO==12 & TYPE=="P", select= c(IDKW, DEL, SPLIT))
```

```
RevGNdup[c(110, 112, 114, 118, 121, 122, 124), 6] <- "Y"
RevGNdup[c(111, 115, 128), 7] <- 1
RevGNdup[c(113, 117, 120), 7] <- 2
RevGNdup[c(116, 119), 7] <- 3
RevGNdup[c(123, 125), 7] <- 4
RevGNdup[c(126, 127), 7] <- 5
subset(RevGNdup, SET_NO==12 & TYPE=="P", select= c(IDKW, DEL, SPLIT))

# Reconstruct ProDup object
GNdup2 <- ReconstructProbDup(RevGNdup)
lapply(disGNdup, nrow)
lapply(GNdup2, nrow)


## End(Not run)
```

`ReviewProbDup` *Retrieve probable duplicate set information from PGR passport database for review*

Description

`ReviewProbDup` retrieves information associated with the probable duplicate sets from the original PGR passport database(s) from which they were identified in order to facilitate manual clerical review.

Usage

```
ReviewProbDup(
  pdup,
  db1,
  db2 = NULL,
  extra.db1 = NULL,
  extra.db2 = NULL,
  max.count = 30,
  insert.blanks = TRUE
)
```

Arguments

`pdup`	An object of class `ProbDup`.
`db1`	A data frame of the PGR passport database.
`db2`	A data frame of the PGR passport database. Required when `pdup` was created using more than one KWIC Index.
`extra.db1`	A character vector of extra `db1` column names to be retrieved.
`extra.db2`	A character vector of extra `db2` column names to be retrieved.
`max.count`	The maximum count of probable duplicate sets whose information is to be retrieved.
`insert.blanks`	logical. If `TRUE`, inserts a row of /codeNAs after each set.

Details

This function helps to retrieve PGR passport information associated with fuzzy, phonetic or semantic probable duplicate sets in an object of class ProbDup from the original databases(s) from which they were identified. The original information of accessions comprising a set, which have not been subjected to data standardization can be compared under manual clerical review for the validation of the set.

By default only the fields(columns) which were used initially for creation of the KWIC indexes using the KWIC function are retrieved. Additional fields(columns) if necessary can be specified using the extra.db1 and extra.db2 arguments.

The output data frame can be subjected to clerical review either after exporting into an external spreadsheet using write.csv function or by using the edit function.

The column DEL can be used to indicate whether a record has to be deleted from a set or not. Y indicates "Yes", and the default N indicates "No".

The column SPLIT similarly can be used to indicate whether a record in a set has to be branched into a new set. A set of identical integers in this column other than the default 0 can be used to indicate that they are to be removed and assembled into a new set.

Value

A data frame of the long/narrow form of the probable duplicate sets data along with associated fields from the original database(s). The core columns in the resulting data frame are as follows:

SET_NO	The set number.
TYPE	The type of probable duplicate set. 'F' for fuzzy, 'P' for phonetic and 'S' for semantic matching sets.
K[*]	The KWIC index or database of origin of the record. The method is specified within the square brackets in the colu
PRIM_ID	The primary ID of the accession record from which the set could be identified.
IDKW	The 'matching' keywords along with the IDs.
DEL	Column to indicate whether record has to be deleted or not.
SPLIT	Column to indicate whether record has to be branched and assembled into new set.
COUNT	The number of elements in a set.

For the retrieved columns(fields) the prefix K* indicates the KWIC index of origin.

Note

When any primary ID/key records in the fuzzy, phonetic or semantic duplicate sets are found to be missing from the original databases db1 and db2, then they are ignored and only the matching records are considered for retrieving the information with a warning.

This may be due to data standardization of the primary ID/key field using the function DataClean before creation of the KWIC index and subsequent identification of probable duplicate sets. In such a case, it is recommended to use an identical data standardization operation on the databases db1 and db2 before running this function.

With R <= v3.0.2, due to copying of named objects by list(), Invalid .internal.selfref detected and fixed... warning can appear, which may be safely ignored.

See Also

DataClean, KWIC, ProbDup

Examples

```
## Not run:

# Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
         "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
         "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
         "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
         "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
         "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
                 phonetic = TRUE, encoding = "primary",
                 semantic = TRUE, syn = syn)

# Get disjoint probable duplicate sets of each kind
disGNdup <- DisProbDup(GNdup, combine = NULL)

# Get the data frame for reviewing the duplicate sets identified
RevGNdup <- ReviewProbDup(pdup = disGNdup, db1 = GN1000,
                          extra.db1 = c("SourceCountry", "TransferYear"),
                          max.count = 30, insert.blanks = TRUE)
# Examine and review the duplicate sets using edit function
```

```
RevGNdup <- edit(RevGNdup)

# OR examine and review the duplicate sets after exporting them as a csv file
write.csv(file="Duplicate sets for review.csv", x=RevGNdup)


## End(Not run)
```

SplitProbDup *Split an object of class* ProbDup

Description

SplitProbDup splits an object of class ProbDup into two on the basis of set counts.

Usage

```
SplitProbDup(pdup, splitat = c(30, 30, 30))
```

Arguments

pdup	An object of class ProbDup.
splitat	A vector of 3 integers indicating the set count at which Fuzzy, Phonetic and Semantic duplicate sets in pdup are to be split.

Value

A list with the the divided objects of class ProbDup (pdup1 and pdup2) along with the corresponding lists of accessions present in each (list1 and list2).

See Also

ProbDup, MergeProbDup

Examples

```
## Not run:
# Load PGR passport database
GN <- GN1000

# Specify as a vector the database fields to be used
GNfields <- c("NationalID", "CollNo", "DonorID", "OtherID1", "OtherID2")

# Clean the data
GN[GNfields] <- lapply(GN[GNfields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
c("Mota", "Company"))
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
```

```
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN[GNfields] <- lapply(GN[GNfields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Generate KWIC index
GNKWIC <- KWIC(GN, GNfields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
          "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
          "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
          "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
          "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
          "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

# Fetch probable duplicate sets
GNdup <- ProbDup(kwic1 = GNKWIC, method = "a", excep = exep, fuzzy = TRUE,
                 phonetic = TRUE, encoding = "primary",
                 semantic = TRUE, syn = syn)

# Split the probable duplicate sets
GNdupSplit <- SplitProbDup(GNdup, splitat = c(10, 10, 10))


## End(Not run)
```

`ValidatePrimKey` *Validate if a data frame column confirms to primary key/ID constraints*

Description

`ValidatePrimKey` checks if a column in a data frame confirms to the primary key/ID constraints of absence of duplicates and NULL values. Aberrant records if encountered are returned in the output list.

Usage

```
ValidatePrimKey(x, prim.key)
```

Arguments

`x`	A data frame.
`prim.key`	A character vector indicating the name of the data frame column to be validated for primary key/ID constraints (see **Details**).

Details

The function checks whether a field(column) in a data frame of PGR passport database confirms to the primary key/ID constraints of absence of duplicates and NULL values. If records with nonconforming values in the column are encountered, they are returned in the output list for rectification.

If multiple fields(columns) are given as a character vector in `prim.key` field, only the first element will be considered as the primary key/ID field(column).

Cleaning of the data in the input field(column) using the `DataClean` function with appropriate arguments is suggested before running this function.

It is recommended to run this function and rectify aberrant records in a PGR passport database before creating a KWIC index using the `KWIC` function.

Value

A list with containing the following components:

`message1`	Indicates whether duplicated values were encountered in `prim.key` field(column) of data frame `x` or not.
`Duplicates`	A data frame of the records with duplicated prim.key values if they were encountered.
`message2`	Indicates whether NULL values were encountered in `prim.key` field(column) of data frame `x` or not.
`NullRecords`	A data frame of the records with NULL prim.key values if they were encountered.

See Also

`DataClean`, `KWIC`

Examples

```
GN <- GN1000
ValidatePrimKey(x=GN, prim.key="NationalID")
## Not run:
# Show error in case of duplicates and NULL values
# in the primary key/ID field "NationalID"
GN[1001:1005,] <- GN[1:5,]
GN[1001,3] <- ""
ValidatePrimKey(x=GN, prim.key="NationalID")
## End(Not run)
```

ViewProbDup *Visualize the probable duplicate sets retrieved in a* ProbDup *object*

Description

`ViewProbDup` plots summary visualizations of accessions within the probable duplicate sets retrieved in a `ProbDup` object according to a grouping factor field(column) in the original database(s).

Usage

```
ViewProbDup(
  pdup,
  db1,
  db2 = NULL,
  factor.db1,
  factor.db2 = NULL,
  max.count = 30,
  select,
  order = "type",
  main = NULL
)
```

Arguments

pdup	An object of class ProbDup.
db1	A data frame of the PGR passport database.
db2	A data frame of the PGR passport database. Required when pdup was created using more than one KWIC Index.
factor.db1	The db1 column to be considered for grouping the accessions. Should be of class character or factor.
factor.db2	The db2 column to be considered for grouping the accessions. Should be of class character or factor. retrieved.
max.count	The maximum count of probable duplicate sets whose information is to be plotted (see **Note**).
select	A character vector of factor names in factor.db1 and/or factor.db2 to be considered for grouping accessions (see **Note**).
order	The order of the type of sets retrieved in the plot. The default is "type" (see **Details**).
main	The title of the plot.

Value

A list containing the following objects:

Summary1	The summary data.frame of number of accessions per factor level.
Summary2	The summary data.frame of number of accessions and sets per each type of sets classified according to facto
SummaryGrob	A grid graphical object (Grob) of the summary visualization plot. Can be plotted using the grid.arrange fur

Note

When any primary ID/key records in the fuzzy, phonetic or semantic duplicate sets are found to be missing from the original databases db1 and db2, then they are ignored and only the matching records are considered for visualization.

This may be due to data standardization of the primary ID/key field using the function DataClean

before creation of the KWIC index and subsequent identification of probable duplicate sets. In such a case, it is recommended to use an identical data standardization operation on the databases db1 and db2 before running this function. For summary and visualization of the set information in the object of class ProbDup by ViewProbDup, the disjoint of the retrieved sets are made use of, as they are more meaningful than the raw sets retrieved. So it is recommended that the disjoint of sets obtained using the DisProbDup be used as the input pdup.

All the accession records in sets with count > max.count will be considered as being unique.

The factor levels in the factor.db1 and/or factor.db2 columns corresponding to those mentioned in select argument alone will be considered for visualization. All other factor levels will be grouped together to a single level named "Others".

The argument order can be used to specify the order in which the type of sets retrieved are to be plotted in the visualization. The default "type" will order according to the kind of sets, "sets" will order according to the number of sets in each kind and "acc" will order according to the number of accessions in each kind.

The individual plots are made using ggplot and then grouped together using gridExtra-package.

See Also

ProbDup, DisProbDup, DataClean, ggplot, gridExtra-package

Examples

```
## Not run:

# Method "b and c"
#=================

# Load PGR passport databases
GN1 <- GN1000[!grepl("^ICG", GN1000$DonorID), ]
GN1$DonorID <- NULL
GN2 <- GN1000[grepl("^ICG", GN1000$DonorID), ]
GN2 <- GN2[!grepl("S", GN2$DonorID), ]
GN2$NationalID <- NULL

GN1$SourceCountry <- toupper(GN1$SourceCountry)
GN2$SourceCountry <- toupper(GN2$SourceCountry)

GN1$SourceCountry <- gsub("UNITED STATES OF AMERICA", "USA", GN1$SourceCountry)
GN2$SourceCountry <- gsub("UNITED STATES OF AMERICA", "USA", GN2$SourceCountry)

# Specify as a vector the database fields to be used
GN1fields <- c("NationalID", "CollNo", "OtherID1", "OtherID2")
GN2fields <- c("DonorID", "CollNo", "OtherID1", "OtherID2")

# Clean the data
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) DataClean(x))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) DataClean(x))
y1 <- list(c("Gujarat", "Dwarf"), c("Castle", "Cary"), c("Small", "Japan"),
           c("Big", "Japan"), c("Mani", "Blanco"), c("Uganda", "Erect"),
           c("Mota", "Company"))
```

```
y2 <- c("Dark", "Light", "Small", "Improved", "Punjab", "SAM")
y3 <- c("Local", "Bold", "Cary", "Mutant", "Runner", "Giant", "No.",
        "Bunch", "Peanut")
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN1[GN1fields] <- lapply(GN1[GN1fields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) MergeKW(x, y1, delim = c("space", "dash")))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) MergePrefix(x, y2, delim = c("space", "dash")))
GN2[GN2fields] <- lapply(GN2[GN2fields], function(x) MergeSuffix(x, y3, delim = c("space", "dash")))

# Remove duplicated DonorID records in GN2
GN2 <- GN2[!duplicated(GN2$DonorID), ]

# Generate KWIC index
GN1KWIC <- KWIC(GN1, GN1fields)
GN2KWIC <- KWIC(GN2, GN2fields)

# Specify the exceptions as a vector
exep <- c("A", "B", "BIG", "BOLD", "BUNCH", "C", "COMPANY", "CULTURE",
          "DARK", "E", "EARLY", "EC", "ERECT", "EXOTIC", "FLESH", "GROUNDNUT",
          "GUTHUKAI", "IMPROVED", "K", "KUTHUKADAL", "KUTHUKAI", "LARGE",
          "LIGHT", "LOCAL", "OF", "OVERO", "P", "PEANUT", "PURPLE", "R",
          "RED", "RUNNER", "S1", "SAM", "SMALL", "SPANISH", "TAN", "TYPE",
          "U", "VALENCIA", "VIRGINIA", "WHITE")

# Specify the synsets as a list
syn <- list(c("CHANDRA", "AH114"), c("TG1", "VIKRAM"))

GNdupc <- ProbDup(kwic1 = GN1KWIC, kwic2 = GN2KWIC, method = "c",
                  excep = exep, fuzzy = TRUE, phonetic = TRUE,
                  encoding = "primary", semantic = TRUE, syn = syn)

GNdupcView <- ViewProbDup(GNdupc, GN1, GN2, "SourceCountry", "SourceCountry",
                          max.count = 30, select = c("INDIA", "USA"), order = "type",
                          main = "Groundnut Probable Duplicates")

library(gridExtra)
grid.arrange(GNdupcView$SummaryGrob)


## End(Not run)
```

Index