Package ‘sdglinkage’

April 27, 2020

Type Package

Title Synthetic Data Generation for Linkage Methods Development

Version 0.1.0

Author Haoyuan Zhang <howardhyzhang@gmail.com>,
Katie Harron <k.harron@ucl.ac.uk>,
Harvey Goldstein <h.goldstein.uk@gmail.com>,
Andrew Boyd <A.W.Boyd@bristol.ac.uk>,
Ruth Gilbert <r.gilbert@ucl.ac.uk>

Maintainer Haoyuan Zhang <howardhyzhang@gmail.com>

Description A tool for synthetic data generation that can be used for linkage method development, with elements of i) gold standard file with complete and accurate information and ii) linkage files that are corrupted as we often see in raw dataset.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

Imports bnlearn (>= 4.4.1), synthpop (>= 1.5.1), reshape (>= 0.8.8),
geplot2 (>= 3.1.1), visNetwork (>= 2.0.6), arsenal (>= 3.3.0)

Suggests mlr (>= 2.16.0), PostcodesioR (>= 0.1.1), reclin, dplyr,
knitr, rmarkdown, testthat

RoxygenNote 7.0.2

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2020-04-27 16:20:03 UTC

R topics documented:

acquire_error_flag ............................................... 3
address_uk .......................................................... 4
R topics documented:

add_dependent_error ........................................ 5
add_random_error ............................................. 6
add_variable .................................................. 6
adult .................................................................. 8
bn_flag_inference .............................................. 8
check_swap_char ................................................ 9
compare_cart .................................................... 10
compare_sdg ...................................................... 10
compare_two_df ................................................ 12
damage_gold_standard ......................................... 13
diff_two_strings ................................................ 14
do_ocr_replacement ........................................... 15
do_pho_replacement ........................................... 15
do_typo_replacement .......................................... 16
extract_address ................................................. 17
firstname_uk ..................................................... 17
firstname_uk_variant .......................................... 18
firstname_us .................................................... 18
gen_address ..................................................... 19
gen_bn_elicit ................................................... 19
gen_bn_learn .................................................... 20
gen_cart ........................................................ 21
gen_dob ........................................................... 22
gen_firstname ................................................... 23
gen_lastname ..................................................... 24
gen_nhsid ......................................................... 24
get_address ....................................................... 25
get_transformation_del ....................................... 26
get_transformation_insert ................................... 26
get_transformation_name_variant ......................... 27
get_transformation_ocr ....................................... 27
get_transformation_pho ....................................... 28
get_transformation_trans_char ............................... 29
get_transformation_trans_date .............................. 29
get_transformation_typo ..................................... 30
lastname_uk ....................................................... 31
lastname_uk_variant .......................................... 31
lastname_us ...................................................... 32
ocr_rules ......................................................... 32
pho_rules ........................................................ 33
plot_bn .......................................................... 33
plot_compared_sdg ........................................... 34
replace_firstname .............................................. 35
replace_lastname ............................................... 36
replace_nhsid ................................................... 37
slavo_germanic ................................................ 37
split_data ......................................................... 38

Index .............................................................. 39
acquire_error_flag

Add a column of error flags given two data frames.

Description

compare_two_df compares the vars of data frames given an uniqueId.

Usage

acquire_error_flag(df1, diffs.table, var_name, error_type)

Arguments

df1: Data frame 1.
diffs.table: A data frame of differences between two data frames given by compare_two_df.
var_name: A string of variable name that we want to check if there is error.
error_type: A string of error type name:
1. missing: if the value of var_name is NA in df2, it will be flagged as 1, otherwise, 0;
2. del: if the value of var_name in df2 equals to var_name in df1 with a letter being deleted (see get_transformation_del), it will be flagged as 1, otherwise, 0;
3. trans_char: if the value of var_name in df2 equals to var_name in df1 with two of its letters’ position being transposed (see get_transformation_trans_char), it will be flagged as 1, otherwise, 0;
4. trans_date: if the value of var_name in df2 equals to var_name in df1 with day and month being transposed (see get_transformation_trans_date), it will be flagged as 1, otherwise, 0;
5. insert: if the value of var_name in df2 equals to var_name in df1 with an additional letter being inserted (see get_transformation_insert), it will be flagged as 1, otherwise, 0;
6. typo: if the value of var_name in df2 equals to var_name in df1 with a typo error (see get_transformation_typo), it will be flagged as 1, otherwise, 0;
7. ocr: if the value of var_name in df2 equals to var_name in df1 with an ocr error (see get_transformation_ocr), it will be flagged as 1, otherwise, 0;
8. pho: if the value of var_name in df2 equals to var_name in df1 with a phonetic error (see get_transformation_pho), it will be flagged as 1, otherwise, 0;
9. variant: if the value of var_name in df2 equals to a variant of var_name in df1 (see get_transformation_name_variant), it will be flagged as 1, otherwise, 0;

Value

It returns a data frame of df1 with an additional error flag column called var_name.
Examples

df <- data.frame(firstname_variant=character(20), lastname_variant=character(20))
df <- add_variable(df, "nhsid")
df <- add_variable(df, "firstname", country = "uk", gender_dependency= FALSE,
age_dependency = FALSE)
df <- add_variable(df, "lastname", country = "uk", gender_dependency= FALSE,
age_dependency = FALSE)
df$firstname_variant <- as.character(df$firstname_variant)
df$lastname_variant <- as.character(df$lastname_variant)
for (i in 1:nrow(df)){
  df$firstname_variant[i] = strsplit(get_transformation_name_variant(df$firstname[i]),
'\''\''\''\',\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\''\'
add_dependent_error

Add two dependent error flags to a data frame.

Description

`add_dependent_error` adds two column of dependent error flags (between 0 and 1) to a data frame.

Usage

```r
add_dependent_error(
  dataset,
  error_names,
  prior_probs = c(0.5, 0.5),
  cond_probs = c(0.95, 0.05, 0.85, 0.15)
)
```

Arguments

- **dataset**: A data frame of the dataset.
- **error_names**: A string of the variable names and type of the error in the form of 'variable 1_variable 2_error type'. The error of variable 2 depends on the error of variable 1. The error type can be either: 'missing', 'insert', 'variant', 'typo', 'pho', 'ocr', 'trans_date' or 'trans_char'.
- **prior_probs**: A vector of two numerical probabilities, where the first one is the prior probability of variable 1 being 0 (no error) and the second one is the prior probability of variable 1 being 1 (having error).
- **cond_probs**: A vector of four numerical probabilities, where the first two probabilities are the probabilities of variable 2 being 0 and 1 given variable 1 being 0, and the last two are the probabilities of variable 2 being 0 and 1 given variable 1 being 1.

Value

A data frame of the dataset with two additional dependent column of binary encoded error.

Examples

```r
adult_with_flag <- add_dependent_error(adult[1:100,], "race_sex_typo")
adult_with_flag <- add_dependent_error(adult[1:100,], "age_sex_missing",
  prior_probs = c(0.99, 0.01),
  cond_probs = c(0.95, 0.05, 0.4, 0.6))
```
add_random_error  \hspace{1cm} \textit{Add random error flags to a data frame.}

Description

add_random_error adds a column of error flags (between 0 and 1) to a data frame based on the prob.

Usage

\begin{verbatim}
add_random_error(dataset, error_name, prob = c(0.95, 0.05))
\end{verbatim}

Arguments

- dataset: A data frame of the dataset.
- error_name: A string of the name and type of the error in the form of 'error name_error type'. The error name should be one of the variable name in the dataset, and the error type can be either: 'missing', 'insert', 'variant', 'typo', 'pho', 'ocr', 'trans_date' or 'trans_char'.
- prob: A vector of two numerical probabilities, where the first one is the probability of being 0 and the second one is the probability of being 1.

Value

A data frame of the dataset with an additional column of binary encoded error.

Examples

\begin{verbatim}
adult_with_flag <- add_random_error(adult[1:100,], prob = c(0.97, 0.03), "age_missing")
adult_with_flag <- add_random_error(adult_with_flag, prob = c(0.65, 0.35), "education_typo")
\end{verbatim}

add_variable  \hspace{1cm} \textit{Add a synthetic but realistic variable to a dataset following some rules.}

Description

add_variable adds a column of new variable to a dataset. This new variable generated by some realistic rules. Several type of variables are included:

1. nhsid: each row is assigned with an identical 10-digit id that is randomly generated following the Modulus 11 Algorithm;
2. dob: if the age_dependency is TRUE and there is a variable called 'age' in the dataset, the dob is generated based on the value of age and end_date. If age_dependency is FALSE, the dob is randomly generated between start_date and end_date;
3. address: a random UK address sampled from 30,000 UK addresses, see gen_address;

4. firstname: randomly sample a firstname from the selected database:
   - country If is ‘uk’ and gender_dependency and age_dependency are both TRUE, the
     generated firstnames will automatically sample a firstname that based on the gender and
     age of the individuals within the dataset. The uk firstname database was extracted from
     ONS containing firstnames and their frequencies in England and Wales from 1996 to
     2018.
   - If country is ’us’ and gender_dependency and race_dependency are both TRUE, the
     generated firstnames will automatically sample a firstname that based on the gender and
     ethnicity of the individuals within the dataset. The us firstname database was extracted
     from randomNamesData. Current ethnicity codes are: 1 American Indian or Native
     Alaskan, 2 Asian or Pacific Islander, 3 Black (not Hispanic), 4 Hispanic, 5 White (not
     Hispanic) and 6 Middle-Eastern, Arabic.

5. lastname: randomly sample a lastname from the selected database:
   - If country is ’uk’, the generated lastnames will automatically sample a lastname from a
     extracted lastname database. The lastname database was extracted from ONS.
   - If country is ’us’ and race_dependency is TRUE, the generated lastnames will automatic-
     ically sample a lastname that based on the individual’s ethnicity. The us lastname database
     was extracted from randomNamesData.

Usage

add_variable(
  dataset, 
  type, 
  country = "uk", 
  start_date = "1900-01-01", 
  end_date = "2020-01-01", 
  age_dependency = FALSE, 
  gender_dependency = FALSE, 
  race_dependency = FALSE
)

Arguments

dataset A data frame of the dataset.
type A string of the type of variable we want to add: 'nhsid', 'dob', 'address', 'first-
       name' or 'lastname'.
country A string variable with a default of 'uk'. It can be either 'uk' or 'us'.
start_date A Date variable with a default of '1900-01-01'.
end_date A Date variable with a default of '2020-01-01'.
age_dependency A logical variable with a default of FALSE
gender_dependency A logical variable with a default of FALSE
race_dependency A logical variable with a default of FALSE.
**Value**

A data frame of the dataset with a new generated variable.

**Examples**

```r
tmp1 <- add_variable(adult[1:100,], "nhsid")
tmp2 <- add_variable(adult[1:100,], "dob", end_date = "2015-03-02", age_dependency = TRUE)
tmp3 <- add_variable(adult[1:100,], "address")
tmp4 <- add_variable(adult[1:100,], "firstname", country = "uk", age_dependency = TRUE, gender_dependency = TRUE)
tmp5 <- add_variable(adult[1:100,], "lastname", country = "uk")
tmp6 <- add_variable(adult[1:100,], 'firstname', country = 'us', gender_dependency=True, race_dependency=True)
tmp7 <- add_variable(adult[1:100,], 'lastname', country='us', race_dependency = TRUE)
```

---

**adult**

*Adult dataset.*

---

**Description**

The Adult dataset was extracted from the US Census database in 1994; it contains 48,842 individual records with 13 personal variables. It is often used as a prediction task to determine whether a person makes over $50,000 a year given personal information.

**Usage**

```r
adult
```

**Format**

A data frame with 13 variables: age, workclass, marital_status, occupation, relationship, race, sex, capital_gain, capital_loss, hours_per_week, native_country and income.

---

**bn_flag_inference**

*Bayesian inference for error prediction.*

---

**Description**

bn_flag_inference use the trained Bayesian Network model to predict the errors that may happen in the dataset.

**Usage**

```r
bn_flag_inference(dataset, fit_model)
```
check_swap_char

Arguments

dataset       A data frame.
fit_model     A bn fit model generated by `bn.fit` or `gen_bn_learn` or `gen_bn_elicit`.

Value
The output is a data frame inferred error flags.

Examples

```r
adult_with_flag <- add_random_error(adult[1:100,], prob = c(0.97, 0.03), "age_missing")
adult_with_flag <- split_data(adult_with_flag, 70)
bn_learn <- gen_bn_learn(adult_with_flag$training_set, "hc")
syn_error_occurrence <- bn_flag_inference(bn_learn$gen_data, bn_learn$fit_model)
syn_error_occurrence
```

check_swap_char

Check if two strings are the same after we swaped the position of two letters.

Description
check_swap_char check if two strings are the same after we swaped the position of two letters.

Usage

```r
check_swap_char(string1, string2)
```

Arguments

string1       A string.
string2       A string.

Value
It returns TRUE if two strings are the same after we swaped the position of two letters, otherwise, it returns FALSE.
**compare_cart**  
*Compare the synthetic data generated by CART with the real data.*

**Description**

*compare_cart* compare the synthetic data generated by CART with the real data.

**Usage**

```r
compare_cart(training_set, fit_model, var_list)
```

**Arguments**

- `training_set`: A data frame of the training data. The generated data will have the same size as the `training_set`.
- `fit_model`: A `syn` object.
- `var_list`: A string vector of the names of variables that we want to compare.

**Value**

A plot of the comparison of the distribution of synthetic data vs real data.

**Examples**

```r
adult_data <- split_data(adult[1:100,], 70)
cart <- gen_cart(adult_data$training_set)
compare_cart(adult_data$training_set, cart$fit_model, c("age", "workclass", "sex"))
```

---

**compare_sdg**  
*Compare the performance of generators.*

**Description**

*compare_sdg* compares the predictive performance of models trained by synthetic data with model trained by real data.

**Usage**

```r
compare_sdg(
  learner,
  measurement,
  target_var,
  real_dataset,
  generated_data1,
```

---
Arguments

learner A learner object from `makeLearners`.
measurement A list of performance measurements for `benchmark`.
target_var A string of the response variable name.
real_dataset A list of data frames with a training_set data frame and a testing_set data frame. You can get this list from `split_data`.

generated_data1 A data frame of synthetic data 1.
generated_data2 A data frame of synthetic data 2.
generated_data3 A data frame of synthetic data 3.
generated_data4 A data frame of synthetic data 4.
generated_data5 A data frame of synthetic data 5.
generated_data6 A data frame of synthetic data 6.

Details

This function returns the measured performance of predictive models trained by the synthetic data. We assume good quality synthetic data would allow us to draw the same analytic conclusions as we can draw from real data. Hence, we compare the predictive performance of several machine learning algorithms that are trained with the synthetic data and tested by real data with those trained and tested both by real data.

Value

The output is a `benchmark` object. It compares the the predictive performance of selected models trained by the real data and validated by the testing data with models trained by the generated data and validated by the testing data.

Examples

```r
library(mlr)
adult_data <- adult[c('age', 'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'income')]
adult_data <- split_data(adult_data[1:100,], 70)
bn_learn <- gen_bn_learn(adult_data$training_set, "hc")
```
lrns <- makeLearners(c("rpart", "logreg"), type = "classif", predict.type = "prob")
measurements <- list(acc, ber)
bmr <- compare_sdg(lrns, measurement = measurements, target_var = "income", real_dataset = adult_data, generated_data = bn_learn$gen_data)
names(bmr$results) <- c("real_dataset", "bn_learn")
bmr

c

### compare_two_df

**Compare two data frames.**

**Description**

compare_two_df compares the vars of data frames given an uniqueId.

**Usage**

```r
compare_two_df(df1, df2, vars, uniqueId)
```

**Arguments**

- `df1`: Data frame 1.
- `df2`: Data frame 2.
- `vars`: A list of vector of variables to be compared. In each vector, the first variable name belongs to df1, and the second variable name belongs to df2.
- `uniqueId`: A string of unique ID that is used to matched df2 with df1.

**Value**

It returns a data frame of 7 variables:

1. `var.x`: the name of the first variable name in each vector of vars;
2. `var.y`: the name of the second variable name in each vector of vars;
3. `uniqueId`: the unique ID given by uniqueId;
4. `values.x`: the value of the first variable name in each vector of vars;
5. `values.y`: the value of the second variable name in each vector of vars;
6. `row.x`: the row of the values.x in df1;
7. `row.y`: the row of the values.y in df2;
Examples

df <- data.frame(firstname_variant=character(100), lastname_variant=character(100))
df <- add_variable(df, "nhsid")
df <- add_variable(df, "firstname", country = "uk", gender_dependency= FALSE, age_dependency = FALSE)
df <- add_variable(df, "lastname", country = "uk", gender_dependency= FALSE, age_dependency = FALSE)
df$firstname_variant <-as.character(df$firstname_variant)
df$lastname_variant <-as.character(df$lastname_variant)
for (i in 1:nrow(df)){
  df$firstname_variant[i] = strsplit(get_transformation_name_variant(df$firstname[i]), ',')[1][1]
  df$lastname_variant[i] = strsplit(get_transformation_name_variant(df$lastname[i]), ',')[1][1]
}
df1 = df[c('nhsid', 'firstname', 'lastname')]
df2 = df[c('nhsid', 'firstname_variant', 'lastname_variant')]
df2[1:3, 'firstname_variant'] = NA
vars = list(c('firstname', 'firstname_variant'), c('lastname', 'lastname_variant'))
diffs.table = compare_two_df(df1, df2, vars, 'nhsid')

Description

damage_gold_standard damage the gold_standard file into a linkage files. The damage actions are instructued by the error flags in syn_error_occurrence. These actions are:

1. missing: assign 'NA' to the flagged data point;
2. del: randomly delete one character on the flagged data point;
3. trans_char: randomly transpose two neighboring characters on the flagged data point;
4. trans_date: randomly transpose the day and the month of a date on the flagged data point;
5. insert: randomly insert one character to the flagged data point;
6. typo: randomly assign a typo error to the flagged data point;
7. ocr: randomly assign an ocr error to the flagged data point;
8. pho: randomly assign a phonetic error to the flagged data point;
9. variant: randomly assign a name variant to the flagged data point.

Usage

damage_gold_standard(gold_standard, syn_error_occurrence)

Arguments

gold_standard A data frame of the gold standard dataset, see add_variable.
syn_error_occurrence A data frame of one-hot encoded error flags, see bn_flag_inference.
Value

A list of two data frame: i) the linkage_file having the same dimension as the gold_standard but some of the variables are damaged; ii) the error_log records the damages have made on the linkage file.

Examples

adult_with_flag <- add_random_error(adult[1:50,], prob = c(0.97, 0.03), "age_missing")
adult_with_flag <- add_random_error(adult_with_flag, prob = c(0.65, 0.35), "firstname_variant")
adult_with_flag <- split_data(adult_with_flag, 70)
bn_evidence <- "age >=18 & capital_gain>=0 & capital_loss >=0 & hours_per_week>=0 & hours_per_week<=100"
bn_learn <- gen_bn_learn(adult_with_flag$training_set, "hc", bn_evidence)
dataset_smaller_version <- bn_learn$gen_data
syn_dependent <- dataset_smaller_version[, !grepl("flag", colnames(dataset_smaller_version))]
gold_standard <- add_variable(syn_dependent, "firstname", country = "uk", gender_dependency = TRUE, age_dependency = TRUE)
syn_error_occurrence <- bn_flag_inference(dataset_smaller_version, bn_learn$fit_model)
linkage_file <- damage_gold_standard(gold_standard, syn_error_occurrence)

diff_two_strings

Find all letters in string1 which are not in string2. diff_two_strings is adopted from package vecsets function vsetdiff; it returns all letters in string1 which are not in string2.

Description

Find all letters in string1 which are not in string2. diff_two_strings is adopted from package vecsets function vsetdiff; it returns all letters in string1 which are not in string2.

Usage

diff_two_strings(string1, string2, multiple = TRUE)

Arguments

string1 A string.
string2 A string.
multiple A logical variable with a default of TRUE. If multiple is TRUE, it will non-unique letters, otherwise, only unique letters.
do_ocr_replacement

Description

`do_ocr_replacement` replace a string with its OCR error.

Usage

`do_ocr_replacement(s, where, orgpat, newpat)`

Arguments

- `s` A string.
- `where` A string. The location of the pat, it can be one of: 'ALL', 'START', 'END', 'MIDDLE'.
- `orgpat` A string. The original pat.
- `newpat` A string. The new pat.

Value

It returns a new pat.

---

do_pho_replacement

Description

`do_pho_replacement` replace a string with its phonetic error.

Usage

`do_pho_replacement(s, where, orgpat, newpat, precond, postcond, existcond, startcond)`

Arguments

- `s` A string.
- `where` A string. The location of the pat, it can be one of: 'ALL', 'START', 'END', 'MIDDLE'.
- `orgpat` A string. The original pat.
- `newpat` A string. The new pat.
- `precond` A string.
- `postcond` A string.
- `existcond` A string.
- `startcond` A string.
do_typo_replacement

Arguments

- **s** A string.
- **where** A string. The location of the pat, it can be one of: 'ALL','START','END','MIDDLE'.
- **orgpat** A string. The original pat.
- **newpat** A string. The new pat.
- **precond** A string. Pre-condition (default 'None') can be 'V' for vowel or 'C' for consonant.
- **postcond** A string. Post-condition (default 'None') can be 'V' for vowel or 'C' for consonant.
- **existcond** A string. Exist-condition (default 'None').
- **startcond** A string. Start-condition (default 'ALL').

Value

It returns a new pat.

---

do_typo_replacement  Replace a string with its typo error.

Description

do_ocr_replacement replace a string with its typo error.

Usage

do_typo_replacement(s)

Arguments

- **s** A string.

Value

It returns a new pat.
extract_address

Extract addresses.

**Description**

`extract_address` extract addresses using `get_address`.

**Usage**

`extract_address(n = 100, postcode = NA)`

**Arguments**

- `n` A number.
- `postcode` A string.

**Value**

The output is `n` addresses in the form of a data framework with `n` observations with 5 variables:

1. postcode of the UK address,
2. country,
3. primary_care_trust,
4. longitude of the address,
5. latitude of the address.

If `postcode` is given, the return addresses are addresses having the same outward postcode.

firstname_uk

*Baby birth first names in England and Wales.*

**Description**

Full baby birth name data provided by the ONS. This includes all names with at least 5 uses in England and Wales from 1996 to 2018. The frequency was calculated by the number of uses in each name divided by the number of birth population within each birth year. Details can be found in https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/livebirths/bulletins/babynamesenglandandwales/2018/relateddata.

**Usage**

`firstname_uk`
**Format**

A data frame with 4 variables:

- **firstname** first name
- **freq** probability of being named as **firstname** as a **sex** and born at **birthyear**
- **sex** gender
- **birthyear** the year was born.

**firstname_us**  
*First names in the US census.*

**Description**

The US firstname database was extracted from `randomNamesData`. Its origin is the US census.

**Usage**

`firstname_us`

**Format**

A data frame with 4 variables:

- **firstname** first name
- **freq** probability of being named as **firstname** as a **sex** and is **race**
- **sex** gender
- **race** 1) American Indian or Native Alaskan, 2) Asian or Pacific Islander, 3) Black (not Hispanic), 4) Hispanic, 5) White (not Hispanic) and 6) Middle-Eastern, Arabic.

**firstname_uk_variant**  
*First name variants in the UK.*

**Description**

A record of first name variants in the UK, provided by ONS.

**Usage**

`firstname_uk_variant`

**Format**

A data frame with 3 variables:

- **forename** the reference name
- **forename2** the variant of the **forename**
- **freq** probability of entering **forename2** as a variant of **forename**.

**firstname_uk_variant**  
*First name variants in the UK.*
gen_address

Generate an address.

Description

gen_address randomly return a UK address out of 10,000 UK addresses. The UK addresses were extracted from extract_address.

Usage

gen_address(address_file = sdglinkage::address_uk)

Arguments

address_file A data frame of addresses. The default is UK addresses.

Value

The output is a data frame with 1 observation of 5 variables:

1. postcode of the UK address,
2. country,
3. primary_care_trust,
4. longitude of the address,
5. latitude of the address.

Examples

gen_address()

---

gen_bn_elicit

Generate synthetic data using BN parameter learning with an elicted structure.

Description

gen_bn_elicit uses Bayesian parameter learning (Maximum Likelihood Estimation, MLE) to learn the values of the parameters based on the given dependencies of the variables and the input data.

Usage

gen_bn_elicit(training_set, bn_structure, evidences = NA)
Arguments

- **training_set**: A data frame of the training data. The generated data will have the same size as the training_set.  
- **bn_structure**: A string of the relationships between variables from modelstring.  
- **evidences**: A string of evidence that is used to constraint the sampling of the generated data.

Value

The output is a list of three objects: i) structure: the structure of the BN indicating the relationship between the variables (a **bn-class** object); ii) fit_model: the fitted model showing the parameter distributions between the variables ((a **bn.fit** object and iii) gen_data: the generated synthetic data - if there is evidence to constraint the values for some of the variables, the generated synthetic data will be sampled according to the criteria.

Examples

```r
code example
```

---

**Description**

**gen_bn_learn** uses Bayesian structure learning to simultaneously learn the dependencies and the value of the parameters from the input data.

**Usage**

```r
gen_bn_learn(training_set, structure_learning_algorithm, evidences = NA)
```

**Arguments**

- **training_set**: A data frame of the training data. The generated data will have the same size as the training_set.  
- **structure_learning_algorithm**: A string of the structure learning algorithm from bnlearn.  
- **evidences**: A string of evidence that is used to constraint the sampling of the generated data.
Details


Value

The output is a list of three objects: i) structure: the structure of the learned BN indicating the relationship between the variables (a bn-class object); ii) fit_model: the fitted model showing the parameter distributions between the variables (a bn.fit object) and iii) gen_data: the generated synthetic data - if there is evidence to constraint the values for some of the variables, the generated synthetic data will be sampled according to the criteria.

Examples

```r
adult_data <- split_data(adult[1:100,], 70)
bn_learn1 <- gen_bn_learn(adult_data$training_set, "hc")
bn_evidence <- "age >=18 & capital_gain>=0 & capital_loss >=0 & hours_per_week>=0 & hours_per_week<=100"
bn_learn2 <- gen_bn_learn(adult_data$training_set, "hc", bn_evidence)
```

Description

**gen_cart** uses Classification and Regression Trees (CART) to generate synthetic data by sequentially predicting the value of each variable depending on the value of other variables. Details can be found in syn.

Usage

```r
gen_cart(training_set, structure = NA)
```

Arguments

- **training_set**: A data frame of the training data. The generated data will have the same size as the training_set.
- **structure**: A string of the relationships between variables from modelstring. If structure is NA, the default structure would be the sequence of the variables in the training_set data frame.
Value

The output is a list of three objects: i) structure: the dependency/relationship between the variables (a \texttt{bn-class} object); ii) fit_model: the fitted CART model ((a \texttt{syn} object and iii) gen_data: the generated synthetic data.

Examples

```r
adult_data <- split_data(adult[1:100,,], 70)
cart <- gen_cart(adult_data$training_set)
bn_structure <- "[native_country][income][age|marital_status:education]"
bn_structure = paste0(bn_structure, "[sex][race[native_country][marital_status][race:sex]]")
bn_structure = paste0(bn_structure,"[relationship][marital_status][education][sex:race]")
bn_structure = paste0(bn_structure,"[occupation][education][workclass][occupation]")
bn_structure = paste0(bn_structure,"[hours_per_week][occupation:workclass]")
bn_structure = paste0(bn_structure,"[capital_gain][occupation:workclass:income]")
cart_elicit <- gen_cart(adult_data$training_set, bn_structure)
```

---

**gen_dob**

*Generate a record of date of birth.*

Description

\texttt{gen_dob} randomly return a record of date of birth.

Usage

```r
gen_dob(start = "1900-01-01", end = "2020-01-01")
```

Arguments

- **start**: A Date variable with a default of '1900-01-01'.
- **end**: A Date variable with a default of '2020-01-01'.

Value

The output is a record of date of birth in Date format between 1900-01-01 and 2020-01-01. If \texttt{start} is given, the return date will be between the updated start date and 2020-01-01. If \texttt{end} is also given, the return date will be between the updated start date and updated end date.

Examples

```r
gen_dob()
gen_dob(start = "1995-01-01")
gen_dob(end = "2000-01-01")
gen_dob(start = "1909-01-01", end = "2000-01-01")
```
**Description**

`gen_firstname` randomly sample a firstname from the selected database:

1. If `country` is 'uk', the function will automatically sample a firstname that based on the gender and birthyear. The uk firstname database was extracted from [https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/livebirths/bulletins/babynamesenglandandwales/2018/relateddata](https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/livebirths/bulletins/babynamesenglandandwales/2018/relateddata) containing firstnames and their frequences in England and Wales from 1996 to 2018.

2. If `country` is 'us', the function will automatically sample a firstname that based on the gender and race. The us firstname database was extracted from randomNamesData.

**Usage**

```r
gen_firstname(country = "uk", gender = NA, birthyear = NA, race = NA)
```

**Arguments**

- `country` A string variable with a default of 'uk'. It is either 'uk' or 'us'.
- `gender` A string variable either 'male' or 'female'.
- `birthyear` A number from 1996 to 2018. For number smaller than 1996 will assumes as 1996 and greater than 2018 will assumes as 2018.
- `race` A number or a string of the ethnicity code: 1 American Indian or Native Alaskan, 2 Asian or Pacific Islander, 3 Black (not Hispanic), 4 Hispanic, 5 White (not Hispanic) and 6 Middle-Eastern, Arabic.

**Value**

A name string.

**Examples**

```r
gen_firstname(country = "uk", gender = "male", birthyear = 2013)
gen_firstname(country = "us", gender = "male", race = 2)
gen_firstname(country = "us", gender = "male", race = "Hispanic")
```
**Description**

`gen_lastname` randomly sample a lastname from the selected database:

1. **country** If is 'uk', the function will automatically sample a lastname from a extracted lastname database. The lastname database was extracted from ONS.
2. If country is 'us', the function will automatically sample a lastname that based on the race. The us lastname database was extracted from randomNamesData.

**Usage**

```r
gen_lastname(country = "uk", race = NA)
```

**Arguments**

- **country** A string variable with a default of 'uk'. It is either 'uk' or 'us'.
- **race** A number or a string of the ethnicity code: 1 American Indian or Native Alaskan, 2 Asian or Pacific Islander, 3 Black (not Hispanic), 4 Hispanic, 5 White (not Hispanic) and 6 Middle-Eastern, Arabic.

**Value**

A name string.

**Examples**

```r
gen_lastname(country = "uk")
gen_lastname(country = "us", race = 2)
gen_lastname(country = "us", race = 'Hispanic')
```

---

**Description**

`gen_nhsid` randomly return a 10-digit nhsid that is generated following the Modulus 11 Algorithm;

**Usage**

```r
gen_nhsid()
```
**get_address**

**Value**

The output is string with 10 numbers.

**Examples**

```r
gen_nhsid()
```

---

**Description**

`get_address` get an address using an API from `random_postcode`. The API sample a real UK address from `https://api.postcodes.io/random/postcodes`.

**Usage**

```r
callMenuStrip
```

**Arguments**

- `postcode`: A string

**Value**

The output is a list of 5 variables: 1) postcode of the UK address, 2) country, 3) primary_care_trust 4) longitude of the address and 5) latitude of the address. # If postcode is given, the return address is an address with the defined outward postcode

**Examples**

```r
callMenuStrip
```

```r
callMenuStrip
```
get_transformation_del

Delete a character randomly.

**Description**

*get_transformation_del* randomly delete a character of a string.

**Usage**

```python
get_transformation_del(string)
```

**Arguments**

- `string` A string.

**Value**

It returns the string with one of the characters was randomly deleted. It also comes with the change log of the transformation.

**Examples**

```python
get_transformation_del('how are you?')
```

get_transformation_insert

Insert a character/digit/space/symbol randomly.

**Description**

*get_transformation_insert* randomly insert a character/digit/space/symbol a string.

**Usage**

```python
get_transformation_insert(string)
```

**Arguments**

- `string` A string.

**Value**

It returns the string with an additional character/digit/space/symbol. It also comes with the change log of the transformation.
get_transformation_name_variant

Examples

get_transformation_insert('how are you?')

get_transformation_name_variant

Randomly assign a name to its variant.

Description

get_transformation_name_variant randomly assign a name to its variant. The name variant databases are extracted from Febrl.

Usage

get_transformation_name_variant(string)

Arguments

string A name string.

Value

It returns the name variant of string together with the change log of the transformation. If no name variant was recorded in the database, it returns the same name string with a note of 'no recorded variants'.

Examples

get_transformation_name_variant("ed")
get_transformation_name_variant("shelly")
get_transformation_name_variant("MORRIS")

get_transformation_ocr

Encode OCR error to a string.

Description

get_transformation_ocr randomly assign a Optical Character Recognition (OCR) error to a string. This function was converted from the Python code in Febrl (developed by Agus Pudjijono in 2008, refers to reference https://link.springer.com/chapter/10.1007/978-3-642-01307-2_47.
get_transformation_ocr

Usage

get_transformation_ocr(string)

Arguments

string A string.

Value

It returns the string with a randomly assigned OCR error following rules extracted in the ocr_rules dataset. It also comes with the change log of the transformation.

Examples

get_transformation_ocr('how are you?')

generate_transformation_pho

Encode phonetic error to a string.

get_transformation_pho

Description

get_transformation_pho randomly assign a Phonetic error to a string. This function was converted from the Python code in Febrl (developed by Agus Pudijono in 2008, refers to reference https://link.springer.com/chapter/10.1007/978-3-642-01307-2_47.

Usage

get_transformation_pho(string)

Arguments

string A string.

Value

It returns the string with a randomly assigned phonetic error following rules extracted in the pho_rules dataset. It also comes with the change log of the transformation.

Examples

get_transformation_pho('how are you?')
**get_transformation_trans_char**

*Randomly transpose two neighbouring characters.*

**Description**

`get_transformation_trans_char` randomly transpose two neighbouring characters of a string.

**Usage**

`get_transformation_trans_char(string)`

**Arguments**

- `string` A string.

**Value**

It returns the string with two of the neighbouring characters were randomly transposed. It also comes with the change log of the transformation.

**Examples**

`get_transformation_del('how are you?')`

---

**get_transformation_trans_date**

*Transpose the position of day and month.*

**Description**

`get_transformation_trans_date` transpose the position of day and month of a Date format variable.

**Usage**

`get_transformation_trans_date(date)`

**Arguments**

- `date` A Date variable.
Value

The output is the transposition of day and month of date and the change log of the transposition. If the day of date is greater than 12, the transposition will fail and return the same date with a log saying "cannot transpose due to day >12".

Examples

```python
get_transformation_trans_date("1995-01-11")
get_transformation_trans_date("1995-01-13")
```

---

get_transformation_typo

Encode typographic error to a string.

Description

get_transformation_typo randomly assign a typographic error to a string. This function was converted from the Python code in Febrl (developed by Agus Pudijono in 2008, refers to reference https://link.springer.com/chapter/10.1007/978-3-642-01307-2_47.

Usage

```python
get_transformation_typo(string)
```

Arguments

- `string`: A string.

Value

It returns the string with a randomly assigned typographic error following rules extracted in the typo_rules. It also comes with the change log of the transformation.

Examples

```python
get_transformation_typo('how are you?')
```
UK last name dataset was provided by the ONS. The frequency was calculated by the number of uses in each name divided by the number of the population within the dataset.

**Usage**

lastname_uk

**Format**

A data frame with 2 variables: surname and freq.

---

A record of last name variants in the UK, provided by ONS.

**Usage**

lastname_uk_variant

**Format**

A data frame with 3 variables:

lastname1    the reference name
lastname2    the variant of the lastname1
freq         probability of entering lastname2 as a variant of lastname1.
**lastname_us**  
*Last names in the US census.*

**Description**

The US lastname database was extracted from `randomNamesData`. Its origin is the US census.

**Usage**

`lastname_us`

**Format**

A data frame with 3 variables:

- `lastname`  last name
- `freq`  probability of being named as `lastname` as a sex and is `race`
- `race` 1) American Indian or Native Alaskan, 2) Asian or Pacific Islander, 3) Black (not Hispanic), 4) Hispanic, 5) White (not Hispanic) and 6) Middle-Eastern, Arabic..

**ocr_rules**  
*Look up table of Optical Character Recognition (OCR) errors.*

**Description**

A list of OCR errors that may happen, provided by Febrl.

**Usage**

`ocr_rules`

**Format**

A data frame with 3 variables:

- `position`  the position of the error within a string
- `orgpat`  the original pat
- `newpat`  the error pat of `orgpat` that is misrecognised by the OCR system.
**pho_rules**  
*Look up table of phonetic errors.*

**Description**
A list of phonetic errors that may happen, provided by Febrl.

**Usage**
```
pho_rules
```

**Format**
A data frame with 7 variables:
- `where` the position of the error within a string, can be one of: 'ALL', 'START', 'END', 'MIDDLE'
- `orgpat` the original pat
- `newpat` the error pat of `orgpat` that is misheard
- `precond` pre-condition (default 'None') can be 'V' for vowel or 'C' for consonant
- `postcond` post-condition (default 'None') can be 'V' for vowel or 'C' for consonant
- `existcond` exist-condition (default 'None')
- `startcond` start-condition (default 'None').

**plot_bn**  
*Plot the BN structure.*

**Description**
plot_bn generates a plot of the Bayesian Network structure.

**Usage**
```
plot_bn(structure, ht = "400px")
```

**Arguments**
- `structure` A string of the relationships between variables from `modelstring`.
- `ht` The height of the plot.

**Value**
The output is a plot of the Bayesian Network structure.
plot_compared_sdg

Examples

```r
adult_data <- split_data(adult[1:100,], 70)
bn_learn = gen_bn_learn(adult_data$training_set, 'hc')
plot_bn(bn_learn$structure)
```

---

**plot_compared_sdg**  
Plot the distribution of a variable from the synthetic data comparing with the real data.

**Description**

`plot_compared_sdg` return a plot of the comparison of the distribution of synthetic data vs real data.

**Usage**

```r
plot_compared_sdg(
  target_var,  
  training_set,  
  syn_data_names,  
  generated_data1,  
  generated_data2 = NA,  
  generated_data3 = NA,  
  generated_data4 = NA,  
  generated_data5 = NA,  
  generated_data6 = NA
)
```

**Arguments**

- `target_var`  
  A string of the comparison variable name.
- `training_set`  
  A data frame of the training data.
- `syn_data_names`  
  A string vector of names of the generators.
- `generated_data1`  
  A data frame of synthetic data 1.
- `generated_data2`  
  A data frame of synthetic data 2.
- `generated_data3`  
  A data frame of synthetic data 3.
- `generated_data4`  
  A data frame of synthetic data 4.
- `generated_data5`  
  A data frame of synthetic data 5.
- `generated_data6`  
  A data frame of synthetic data 6.
replace_firstname

Value

The output is a plot of the comparison of the distribution of synthetic data vs real data. If the target_var is discrete, the plot is a bar plot. If the target_var is continuous, the plot is a density plot.

Examples

```r
adult_data <- split_data(adult[1:100,], 70)
bn_learn <- gen_bn_learn(adult_data$training_set, "hc")
plot_compared_sdg(target_var = "age",
                  training_set = adult_data$training_set,
                  syn_data_names = c("bn_learn"),
                  generated_data1 = bn_learn$gen_data)
plot_compared_sdg(target_var = "race",
                  training_set = adult_data$training_set,
                  syn_data_names = c("bn_learn"),
                  generated_data1 = bn_learn$gen_data)
```

replace_firstname

Replace the firstnames with values from another database.

Description

replace_firstname replaces the firstname in dataset with firstname from another database (see firstname_uk and firstname_us) in case they are too sensitive.

Usage

```r
replace_firstname(
  dataset,
  country = "uk",
  age_dependency = TRUE,
  gender_dependency = TRUE,
  race_dependency = FALSE
)
```

Arguments

dataset  A data frame of the dataset.
country  A string variable with a default of ‘uk’. It is either ‘uk’ or ‘us’.
age_dependency  A logical variable with a default of TRUE.
gender_dependency  A logical variable with a default of TRUE.
race_dependency  A logical variable with a default of FALSE.
replace_lastname

Replace the lastnames with values from another database.

Description

replace_lastname replaces the lastname in dataset with lastname from another database (see lastname_uk and lastname_us) in case they are too sensitive.

Usage

replace_lastname(dataset, country = "uk", race_dependency = FALSE)

Arguments

dataset A data frame of the dataset.
country A string variable with a default of 'uk'. It is either 'uk' or 'us'.
race_dependency A logical variable with a default of FALSE.

Value

A data frame of the dataset with the lastname column being replaced by another lastname database.

Examples

df <- data.frame(sex=sample(c('male', 'female'), 100, replace = TRUE))
df$race <- sample(1:6, 100, replace = TRUE)
df <- add_variable(df, "nhsid")
df <- add_variable(df, "firstname", country = "uk", gender_dependency= TRUE, age_dependency = FALSE)
replace_firstname(df, country = 'us', age_dependency = FALSE)
df <- data.frame(sex=sample(c('male', 'female'), 30, replace = TRUE))
df <- add_variable(df, "nhsid")
df <- add_variable(df, "firstname", country = "uk", gender_dependency= TRUE, age_dependency = FALSE)
replace_firstname(df, country = 'us', age_dependency = FALSE)
**replace_nhsid**

Replace nhsid with another random nhsid.

**Description**

*replace_nhsid* replaces the nhsid in dataset with another random nhsid in case they are too sensitive.

**Usage**

```r
replace_nhsid(dataset)
```

**Arguments**

- **dataset**
  
  A data frame of the dataset.

**Value**

A data frame of the dataset with the nhsid column being replaced by random nhsid.

**Examples**

```r
df <- data.frame(sex=sample(c('male', 'female'), 100, replace = TRUE))
df$race <- sample(1:6, 100, replace = TRUE)
df <- add_variable(df, "nhsid")
replace_nhsid(df)
```

**slavo_germanic**

Detect if it has slavo transformation.

**Description**

*slavo_germanic* detect if a string has slavo transformation.

**Usage**

```r
slavo_germanic(str)
```

**Arguments**

- **str**
  
  A string.

**Value**

It returns 1 or 0.
split_data

Split the data into a training_set and a testing_set.

Description

split_data split the data into a training_set and a testing_set based on the training_percentage.

Usage

split_data(dataset, training_percentage)

Arguments

dataset: A data frame of the dataset.
training_percentage: A number between 0 and 100 indicating the percentage of the training dataset.

Value

A list with two data frame: training_set and testing_set.

Examples

adult_data <- split_data(adult[1:100], 70)
Index

*Topic datasets
  address_uk, 4
  adult, 8
  firstname_uk, 17
  firstname_uk_variant, 18
  firstname_us, 18
  lastname_uk, 31
  lastname_uk_variant, 31
  lastname_us, 32
  ocr_rules, 32
  pho_rules, 33

acquire_error_flag, 3
add_dependent_error, 5
add_random_error, 6
add_variable, 6, 13
address_uk, 4
adult, 8

benchmark, 11
bn.fit, 9, 20, 21
bn_flag_inference, 8, 13
bnlearn, 20

check_swap_char, 9
compare_cart, 10
compare_sdg, 10
compare_two_df, 3, 12

damage_gold_standard, 13
diff_two_strings, 14
doocr_replacement, 15
do_pho_replacement, 15
do_typo_replacement, 16

extract_address, 4, 17, 19
firstname_uk, 17, 35
firstname_uk_variant, 18
firstname_us, 18, 35

genaaddress, 7, 19
gen_bn_elicit, 9, 19
gen_bn_learn, 9, 20
gen_cart, 21
gen_dob, 22
gen_firstname, 23
gen_lastname, 24
gen_nhsid, 24
get_address, 17, 25
get_transformation_del, 3, 26
get_transformation_insert, 3, 26
get_transformation_name_variant, 3, 27
get_transformation_ocr, 3, 27
get_transformation_pho, 3, 28
get_transformation_trans_char, 3, 29
get_transformation_trans_date, 3, 29
get_transformation_typo, 3, 30

lastname_uk, 31, 36
lastname_uk_variant, 31
lastname_us, 32, 36

makeLearners, 11
modelstring, 20, 21, 33

ocr_rules, 32
pho_rules, 33
plot_bn, 33
plot_compared_sdg, 34

random_postcode, 4, 25
randomNamesData, 7, 18, 23, 24, 32
replace_firstname, 35
replace_lastname, 36
replace_nhsid, 37

slavo_germanic, 37
split_data, 11, 38
syn, 10, 21, 22