Package 'riskutility'

June 22, 2026

Title Disclosure Risk and Data Utility Metrics for Synthetic and Anonymized Data

Version 0.1.0

Description Provides comprehensive methods to measure disclosure risk and data utility for anonymized and synthetic data. Implements attribution-based risk metrics including Correct Attribution Probability (CAP), Targeted CAP (TCAP), Within Equivalence Class Attribution Probability (WEAP), and RAPID (Risk of Attribute Prediction-Induced Disclosure). Also provides distance-based privacy metrics such as Distance to Closest Record (DCR), Nearest Neighbor Distance Ratio (NNDR), and Identical Match Share (IMS). Utility assessment includes propensity score analysis, distribution comparisons, and various statistical tests. Methods are based on Taub et al. (2018) <doi:10.1007/978-3-319-99771-1_9> and related literature. Designed for integration with 'simPop' S4 classes.

Depends R (>= 4.1.0)

Imports ggplot2, data.table, MASS, VIM, randomForest, ranger, reshape2, stats, graphics, utils

Suggests simPop, synthpop, caretEnsemble, kernelshap, uwot, Rtsne, dbscan, vip, rpart, xgboost, partykit, testthat (>= 3.0.0), torch, knitr, rmarkdown, plotly, misc3d, sdcMicro, caret, robustbase, gridExtra, Hmisc, robCompositions, clue

License GPL-3

URL https://github.com/matthias-da/riskutility

BugReports https://github.com/matthias-da/riskutility/issues

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Matthias Templ [aut, cre] (ORCID: <https://orcid.org/0000-0002-8638-5276>), Oscar Thees [ctb] (ORCID: <https://orcid.org/0009-0001-9378-4988>)

Maintainer Matthias Templ <matthias.templ@gmail.com>

Repository CRAN

Date/Publication 2026-06-22 15:40:09 UTC

Contents

`riskutility-package` *riskutility: Disclosure Risk and Data Utility for Anonymized and Synthetic Data*

Description

Provides a comprehensive toolkit for measuring disclosure risk and data utility in anonymized and synthetic datasets. The package supports multivariate Risk-Utility (R-U) evaluation following the framework described in "Beyond the Trade-off Curve" (Thees, Mueller, Templ 2026).

Details

The package covers the following metric families:

Attribution-based risk: CAP-family measures (`dcap`, `tcap`, `weap`, `disco`) that assess whether adversaries can infer sensitive attributes from quasi-identifiers.

ML-based risk: The `rapid` metric uses machine learning models to predict sensitive attributes, capturing complex non-linear relationships that CAP methods may miss.

Distance-based risk: Holdout-based metrics (`dcr`, `nndr`, `ims`) detect memorization by comparing distances to training vs. holdout data.

Utility measures: Functions for assessing how well synthetic data preserves the statistical properties of the original, including `pMSE`, `specks`, `hellinger`, `energy_distance`, `ci_proximity`, `mmd`, `tstr`, `copula_fidelity`, `tail_fidelity`, `subgroup_utility`, `regression_fidelity`, `contingency_fidelity`, `propscore`, `gower`, and `mqs`.

Privacy models: Classical privacy model assessments including `kanonymity`, `ldiversity`, `individual_risk`, and `suda`.

Extended privacy risk: Additional risk models including `tcloseness`, `nnaa`, `singling_out`, `linkability`, `population_uniqueness`, `epsilon_identifiability`, `delta_presence`, `hitting_rate`, `attacker_risk`, `drisk`, `domias`, and `mia_classifier`.

Information-theoretic measures: Entropy and divergence measures (`KLDiv`, `JSDiv`, `RenyiEntropy`, `mutualInformation`, `information_surprisal`, `max_info_leakage`) for quantifying information leakage.

Comparison functions: Direct comparison tools for original and synthetic data including visualization, statistical tests, dimensionality reduction, and predictive model comparisons.

R-U mapping: The `rumap` function provides comprehensive multivariate Risk-Utility evaluation with Pareto frontier identification.

Visualization Functions

`compare_histograms()` Compare distributions using histograms (weighted and unweighted, faceted).

`compare_boxplots()` Compare numeric variable distributions using weighted/unweighted boxplots.

Statistical Tests

`compare_ks_test()` Kolmogorov-Smirnov test for comparing distributions.

`compare_chisq_gof()` Chi-square Goodness-of-Fit test with multi-dimensional frequency tables (weighted/unweighted).

Distributional Comparisons

`compare_wasserstein()` Calculate Wasserstein distances between numeric or nominal variables.

`compare_multivariate_distribution()` Compare joint distributions using Mahalanobis distance or mutual information.

Summary Statistics

`compare_means_frequencies()` Compare means, medians, robust statistics, skewness, kurtosis for numeric data, and frequencies for categorical data.

`compare_correlation_matrices()` Compare correlations: Pearson, Spearman, robust correlations for numeric, categorical, and mixed data types.

`compare_multivariate_summary_statistics()` Calculate joint summary statistics for numeric and categorical data.

Dimensionality Reduction

`compare_pca()` Principal Component Analysis comparison with biplot options.

`compare_embedding()` t-SNE, UMAP, or Sammon's Mapping (MDS) visualizations for embedding comparisons.

Machine Learning and Predictive Modeling

`compare_model_performance()` Evaluate predictive model performance (accuracy, precision, recall, F1-score, AUC, R-squared, RMSE) using cross-validation.

`compare_feature_importance()` Evaluate stability of feature importance measures from Random Forests, Decision Trees, Gradient Boosting, Permutation Importance, and SHAP values.

Dependencies

The package builds on `ggplot2` and `data.table`; uses `VIM`, `MASS`, `randomForest`, and `ranger` for core computations; and optionally integrates with `simPop`, `synthpop`, `Rtsne`, `uwot`, `caret`, and `robustbase` when installed.

Missing-value handling

Two parameter conventions are used deliberately, according to how much choice a function offers. A logical `na.rm` (the standard R idiom) appears where the only decision is whether to drop incomplete cases before computing a statistic. A character selector appears where several treatments are supported: `na` (`"remove"`, `"impute"`, `"stop"`, and in `pMSE` additionally `"indicator"`) in `propscore`, `pMSE`, and `mqs`; and `na_strategy` (`"constant"`, `"drop"`, `"median"`) in `rapid`, which acts specifically on the sensitive target attribute.

Author(s)

Matthias Templ

See Also

`ggplot`, `data.table`

Examples

```
# Create example datasets
set.seed(123)
X <- data.frame(
  income = rnorm(100, mean = 50000, sd = 10000),
  age = rnorm(100, mean = 40, sd = 10),
  gender = sample(c("Male", "Female"), 100, replace = TRUE),
  weight = runif(100, 0.5, 1.5)
)
Y <- data.frame(
  income = rnorm(100, mean = 48000, sd = 12000),
  age = rnorm(100, mean = 42, sd = 11),
  gender = sample(c("Male", "Female"), 100, replace = TRUE),
  weight = runif(100, 0.5, 1.5)
)

# Example for histogram comparison
compare_histograms(X, Y, num_var = "income", cat_vars = c("gender"),
                   weight_X = "weight", weight_Y = "weight")

# Example for PCA comparison
X_pca <- data.frame(
  income = rnorm(100, mean = 50000, sd = 10000),
  age = rnorm(100, mean = 40, sd = 10),
  expenditure = rnorm(100, mean = 2000, sd = 500)
)
Y_pca <- data.frame(
  income = rnorm(100, mean = 48000, sd = 12000),
  age = rnorm(100, mean = 42, sd = 11),
  expenditure = rnorm(100, mean = 2200, sd = 600)
)
res_pca <- compare_pca(X_pca, Y_pca,
                       vars = c("income", "age", "expenditure"),
                       biplot = TRUE)
```

attacker_risk *Attacker Risk Models (Prosecutor/Journalist/Marketer)*

Description

Computes re-identification risk under three canonical attacker models from Statistical Disclosure Control (SDC). Each model makes different assumptions about the attacker's background knowledge, leading to different risk quantifications based on quasi-identifier (QI) frequencies.

Usage

```
attacker_risk(X, ...)

## S3 method for class 'synth_pair'
attacker_risk(X, data = c("synthetic", "original"), ...)

## Default S3 method:
attacker_risk(
  X,
  key_vars,
  sampling_fraction = 0.01,
  model = c("all", "prosecutor", "journalist", "marketer"),
  na.rm = TRUE,
  ...
)
```

Arguments

X	data frame to assess, or a synth_pair object
...	additional arguments passed to methods (currently unused)
data	character, which dataset to assess: "synthetic" (default) or "original". Only used by the synth_pair method.
key_vars	character vector of quasi-identifier variable names
sampling_fraction	numeric, the fraction of the population represented by the data (default: 0.01). Used for the journalist model to estimate population frequencies.
model	character, which attacker model(s) to compute. One of "prosecutor", "journalist", "marketer", or "all" (default: "all")
na.rm	logical, remove records with NA in key variables (default: TRUE)

Details

The three attacker models differ in their assumptions:

Prosecutor Model: The attacker knows a specific individual is in the dataset and attempts to re-identify them. For each record in equivalence class k with frequency f_k, the individual risk is:

$$risk_i = 1/f_k$$

The global prosecutor risk is the mean over all records.

Journalist Model: The attacker does not know whether the target is in the dataset. This requires estimating the population frequency F_k from the sample frequency f_k using the sampling fraction:

$$F_k \approx f_k/sampling\_fraction$$

$$risk_i = 1/F_k = sampling\_fraction/f_k$$

This typically yields lower risks than the prosecutor model because population frequencies are larger than sample frequencies.

Marketer Model: The attacker does not target any specific individual but wants to re-identify as many records as possible. The global risk is the expected number of successful re-identifications divided by the total number of records:

$$risk = (1/n)\sum_{i=1}^{n} 1/f_k(i)$$

Note that for the marketer model, the global risk equals the prosecutor global risk (both are mean 1/f_k).

Interpretation:

- Risk close to 1: High re-identification risk (many unique records)
- Risk > 0.1: Elevated risk, privacy protection may be insufficient
- Risk < 0.05: Generally acceptable
- Risk close to 0: Low re-identification risk (large equivalence classes)

Value

An object of class "attacker_risk" containing:

- risk_per_record: data.frame with columns: prosecutor, journalist (if applicable), freq (f_k)
- global_risk: named list with prosecutor (mean 1/f_k), journalist (mean 1/F_k), marketer (1/n * sum(1/f_k))
- n_uniques: count of records with f_k = 1
- pct_uniques: fraction of unique records
- freq_table: table of QI combination frequencies
- model: model(s) used
- key_vars: quasi-identifier variables used
- sampling_fraction: sampling fraction used
- n_records: number of records assessed
- privacy_pass: logical, prosecutor global risk <= 0.1

Author(s)

Matthias Templ

References

Hundepool, A., Domingo-Ferrer, J., Franconi, L., Giessing, S., Schulte Nordholt, E., Spicer, K. & de Wolf, P.-P. (2012). *Statistical Disclosure Control*. John Wiley & Sons. doi:10.1002/9781118348239

Elliot, M. (2006). A Computational Algorithm for Handling the Special Uniques Problem. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 14(Supp. 1), 45-59.

Templ, M. (2017). *Statistical Disclosure Control for Microdata: Methods and Applications in R*. Springer. doi:10.1007/9783319502724

See Also

kanonymity for k-anonymity assessment, individual_risk for model-based individual risk, suda for special uniques detection

Other privacy-models: delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
data <- data.frame(
  age = sample(c("young", "middle", "old"), 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE),
  income = sample(c("low", "medium", "high"), 200, replace = TRUE)
)

# Compute all three attacker models
result <- attacker_risk(data, key_vars = c("age", "gender", "region"))
print(result)
summary(result)


plot(result)

# Prosecutor model only
result_p <- attacker_risk(data, key_vars = c("age", "gender", "region"),
                          model = "prosecutor")
print(result_p)

# Journalist model with different sampling fraction
result_j <- attacker_risk(data, key_vars = c("age", "gender"),
                          model = "journalist", sampling_fraction = 0.05)
print(result_j)
```

`chisq_utility` *Chi-Square Utility Measures*

Description

Computes chi-square based utility measures comparing original and synthetic data frequency tables. Implements VW (Voas-Williamson), FT (Freeman-Tukey), G (log-likelihood), and JSD (Jensen-Shannon Divergence) statistics.

Usage

```
chisq_utility(X, ...)

## S3 method for class 'synth_pair'
chisq_utility(X, vars = NULL, max_cells = 10000, ...)

## Default S3 method:
chisq_utility(
  X,
  Y,
  vars,
  max_cells = 10000,
  weight_X = NULL,
  weight_Y = NULL,
  ...
)
```

Arguments

`X`	original data frame, or a `synth_pair` object
`...`	additional arguments passed to methods
`vars`	character vector of variable names to include in tables
`max_cells`	integer, maximum number of cells in cross-tabulation (default: 10000)
`Y`	synthetic data frame (not required if X is a synth_pair)
`weight_X`	optional, name of weight variable in X or numeric vector
`weight_Y`	optional, name of weight variable in Y or numeric vector

Details

These measures compare frequency tables from original and synthetic data.

Chi-Square (χ^2): The standard Pearson chi-square statistic:

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

where O is observed (synthetic) and E is expected (original proportions scaled).

Voas-Williamson (VW): A normalized chi-square measure (Voas & Williamson, 2001):

$$VW = \frac{\chi^2 - df}{N}$$

where df is degrees of freedom and N is sample size. Lower is better; 0 indicates perfect replication.

Freeman-Tukey (FT): Uses a variance-stabilizing transformation:

$$FT = 4 \sum (\sqrt{O} - \sqrt{E})^2$$

G-Test (Log-Likelihood Ratio):

$$G = 2 \sum O \log(O/E)$$

Jensen-Shannon Divergence (JSD): A symmetric, bounded divergence measure:

$$JSD = \frac{1}{2}[KL(P||M) + KL(Q||M)]$$

where M = (P + Q)/2, and KL is Kullback-Leibler divergence.

Interpretation:

- VW close to 0: excellent utility
- VW < 0.01: good utility
- VW > 0.05: potential utility concerns
- JSD ranges 0-1: 0 = identical, 1 = completely different

Value

An object of class "chisq_utility" containing:

- chi2: standard chi-square statistic
- df: degrees of freedom
- p_value: p-value for chi-square test
- VW: Voas-Williamson statistic (normalized chi-square)
- FT: Freeman-Tukey statistic
- G: G-test (log-likelihood ratio) statistic
- JSD: Jensen-Shannon Divergence
- n_cells: number of cells in the table
- n_empty_orig: empty cells in original
- n_empty_synth: empty cells in synthetic
- pct_utility: overall utility percentage (based on VW)
- table_orig: frequency table for original data
- table_synth: frequency table for synthetic data

Author(s)

Matthias Templ

References

Voas, D., & Williamson, P. (2001). Evaluating goodness-of-fit measures for synthetic microdata. *Geographical and Environmental Modelling*, 5(2), 177-200.

Freeman, M. F., & Tukey, J. W. (1950). Transformations related to the angular and the square root. *Annals of Mathematical Statistics*, 21, 607-611.

See Also

`propscore` for propensity score utility, `specks` for SPECKS utility measure

Other utility: `ci_proximity()`, `contingency_fidelity()`, `copula_fidelity()`, `energy_distance()`, `gower()`, `hellinger()`, `mmd()`, `mqs()`, `pMSE()`, `plot.rumap()`, `propscore()`, `regression_fidelity()`, `rumap()`, `specks()`, `subgroup_utility()`, `tail_fidelity()`, `tstr()`

Examples

```
# Create example data
set.seed(123)
orig <- data.frame(
  age = sample(c("young", "middle", "old"), 500, replace = TRUE),
  gender = sample(c("M", "F"), 500, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 500, replace = TRUE)
)

# Synthetic data with some differences
synth <- data.frame(
  age = sample(c("young", "middle", "old"), 500, replace = TRUE,
               prob = c(0.35, 0.4, 0.25)),
  gender = sample(c("M", "F"), 500, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 500, replace = TRUE)
)

# Compute chi-square utility
result <- chisq_utility(orig, synth, vars = c("age", "gender"))
print(result)
summary(result)
plot(result)
```

`ci_overlap` *Comparison of confidence intervals*

Description

Given a sample, how much higher is the change compared to the sampling error.

Usage

```
ci_overlap(x, y, estimator = median, R = 1000)
```

Arguments

x	a vector
y	a vector
estimator	a function with a vector of lenght 1 as output
R	number of bootstrap replicates

Value

A list with the following elements:

m_x The point estimate of the function provided in function argument estimator for the vector x.

m_y The point estimate of the function provided in function argument estimator for the vector x.

ci_x The confidence interval of x.

ci_y The confidence interval of y.

overlap Overlap in confidence intervals.

overlap_perc Overlap in confidence intervals in percentage

ratioSE How much differ is the point estimate of y compared to the half of the confidence interval.

Author(s)

Matthias Templ

See Also

Other comparison: compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
x <- rnorm(100, 10)
y <- x + runif(100, 0, 1)

ci_overlap(x, y, R = 1000)

x <- factor(sample(1:2, size = 100, replace = TRUE))
y <- factor(sample(1:2, size = 100, replace = TRUE))

my_func <- function(x){
  table(x)[1]
}
```

```
ci_overlap(x, y, estimator = my_func)
```

`ci_proximity` *Confidence Interval Proximity*

Description

Computes the proximity of confidence intervals for numeric variables between original and synthetic datasets. This utility measure assesses how well the synthetic data preserves point estimates and their uncertainty.

Usage

```
ci_proximity(X, ...)

## S3 method for class 'synth_pair'
ci_proximity(X, ...)

## Default S3 method:
ci_proximity(
  X,
  Y,
  vars = NULL,
  conf.level = 0.95,
  weight_X = NULL,
  weight_Y = NULL,
  na.rm = TRUE,
  ...
)
```

Arguments

`X`	A data.frame or data.table containing the original dataset.
`...`	additional arguments passed to methods (currently unused)
`Y`	A data.frame or data.table containing the synthetic/anonymized dataset.
`vars`	Character vector of numeric variable names to compare. If NULL (default), all common numeric variables are used.
`conf.level`	Numeric, confidence level for intervals. Default 0.95.
`weight_X`	Optional character string specifying sampling weight variable in X.
`weight_Y`	Optional character string specifying sampling weight variable in Y.
`na.rm`	Logical, whether to remove NA values. Default TRUE.

Details

For each numeric variable, this function computes:

- **CI Overlap**: Proportion of the CIs that overlap, ranging from 0 (no overlap) to 1 (complete overlap or one contains the other)
- **Relative Error**: |mean_Y - mean_X| / |mean_X|, measuring how close the synthetic mean is to the original mean
- **Proximity Score**: Combined measure accounting for both mean accuracy and CI overlap

The CI overlap coefficient is computed as:

$$Overlap = \frac{\max(0, \min(U_X, U_Y) - \max(L_X, L_Y))}{\max(U_X, U_Y) - \min(L_X, L_Y)}$$

where L and U are lower and upper CI bounds.

For utility assessment, higher proximity scores indicate better preservation of statistical properties in synthetic data.

Value

An object of class "ci_proximity" containing:

- per_variable: data.frame with CI comparison for each variable
- proximity_mean: mean proximity score across all variables
- overlap_mean: mean overlap coefficient across all variables
- relative_error_mean: mean relative error of synthetic means
- n_vars: number of variables compared
- vars: variable names used
- conf.level: confidence level used

Author(s)

Matthias Templ

See Also

`ci_overlap` for basic CI overlap calculation, `compare_ks_test` for distribution comparison

Other utility: `chisq_utility()`, `contingency_fidelity()`, `copula_fidelity()`, `energy_distance()`, `gower()`, `hellinger()`, `mmd()`, `mqs()`, `pMSE()`, `plot.rumap()`, `propscore()`, `regression_fidelity()`, `rumap()`, `specks()`, `subgroup_utility()`, `tail_fidelity()`, `tstr()`

Examples

```
set.seed(123)
# Original data
X <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  age = rnorm(500, mean = 40, sd = 10),
  score = rnorm(500, mean = 100, sd = 15)
)

# Good synthetic data (similar means)
Y_good <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  age = rnorm(500, mean = 40, sd = 10),
  score = rnorm(500, mean = 100, sd = 15)
)

# Poor synthetic data (shifted means)
Y_poor <- data.frame(
  income = rnorm(500, mean = 55000, sd = 10000),
  age = rnorm(500, mean = 45, sd = 10),
  score = rnorm(500, mean = 90, sd = 15)
)

result_good <- ci_proximity(X, Y_good)
print(result_good)
summary(result_good)

result_poor <- ci_proximity(X, Y_poor)
print(result_poor)
```

---

compare_boxplots *Compare Boxplots of a Numeric Variable in Two Datasets*

Description

This function compares the distribution of a numeric variable between two datasets (e.g., an original dataset and an anonymized/synthetic version) using boxplots. It allows conditional faceting on categorical variables and supports different visualization styles, including side-by-side, overlapping, and separate boxplots. Sampling weights can be considered to compute weighted medians and quartiles.

Usage

```
compare_boxplots(X, ...)

## S3 method for class 'synth_pair'
compare_boxplots(X, ...)

## Default S3 method:
```

```
compare_boxplots(
  X,
  Y,
  num_var,
  cat_vars,
  weight_X = NULL,
  weight_Y = NULL,
  facet_type = "wrap",
  plot_type = "side",
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
num_var	A character string specifying the numeric variable to compare.
cat_vars	A character vector of categorical variables used for faceting.
weight_X	Optional. A character string specifying the sampling weight variable in X.
weight_Y	Optional. A character string specifying the sampling weight variable in Y.
facet_type	Character string. The faceting type: "wrap" (default) for facet_wrap() or "grid" for facet_grid().
plot_type	Character string. Type of plot: • "side" (default): Side-by-side boxplots. • "overlap": Overlapping boxplots. • "separate": Separate boxplots for each dataset-category combination (Original above, Anonymized below).

Value

A ggplot2 boxplot visualization comparing the distributions.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(
  age = sample(20:80, 500, replace = TRUE),
  gender = sample(c("Male", "Female"), 500, replace = TRUE),
  income = rnorm(500, mean = 50000, sd = 10000),
  weight = runif(500, 0.5, 1.5)
)

Y <- data.frame(
  age = sample(20:80, 1000, replace = TRUE),
  gender = sample(c("Male", "Female"), 1000, replace = TRUE),
  income = rnorm(1000, mean = 48000, sd = 12000),
  weight = runif(1000, 0.5, 1.5)
)

compare_boxplots(X, Y, num_var = "income", cat_vars = c("gender"),
                 weight_X = "weight", weight_Y = "weight",
                 facet_type = "wrap", plot_type = "side")

compare_boxplots(X, Y, num_var = "income", cat_vars = c("gender"),
                 weight_X = "weight", weight_Y = "weight",
                 facet_type = "wrap", plot_type = "separate")
```

`compare_chisq_gof` *Compare Frequencies using Chi-square Goodness-of-Fit Test with Optional Grouping, Weights, and Simulated p-values*

Description

This function performs a Chi-square goodness-of-fit test to compare the observed frequencies in an anonymized/synthetic dataset (Y) with the expected frequencies from an original dataset (X), based on one or more categorical variables (cat_vars). If sampling weights are provided via weight_X and weight_Y, weighted frequency tables are computed using xtabs with a weight formula; otherwise, unweighted counts are computed. Additionally, if group_vars is provided, the Chi-square test is computed separately for each unique group defined by the grouping variables. For each group (or overall if no grouping is provided), the expected frequencies from X are scaled to match the total count in Y before performing the test.

Usage

```
compare_chisq_gof(X, ...)

## S3 method for class 'synth_pair'
compare_chisq_gof(X, ...)

## Default S3 method:
compare_chisq_gof(
```

```
  X,
  Y,
  cat_vars,
  group_vars = NULL,
  weight_X = NULL,
  weight_Y = NULL,
  simulate_p = TRUE,
  B = 2000,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
cat_vars	A character vector specifying the categorical variables to form the contingency table.
group_vars	Optional. A character vector specifying the grouping variables for which the test should be performed separately. Default is NULL, in which case the test is applied to the overall datasets.
weight_X	Optional. A character string specifying the sampling weight variable in X.
weight_Y	Optional. A character string specifying the sampling weight variable in Y.
simulate_p	Logical. Whether to simulate p-values in the Chi-square test. Default is TRUE.
B	Numeric. The number of simulations to perform if simulate_p is TRUE. Default is 2000.

Details

The function computes frequency tables for the variables in cat_vars for both datasets X and Y. If sampling weights are provided, xtabs is used with a formula of the form "weight ~ var1 + var2 + ..."; otherwise, unweighted counts are computed using xtabs. A complete grid of all possible combinations of factor levels is created to ensure that missing cells are filled with zeros. The expected frequencies from X are scaled to match the total count in Y before performing the Chi-square goodness-of-fit test. When group_vars is provided, the function iterates over each unique combination of grouping variables and performs the test on each subset. The simulation of p-values (when simulate_p is TRUE) helps to handle cases with small expected frequencies.

Value

A data.table containing the grouping variables (if provided), the Chi-square statistic, degrees of freedom, and p-value for each group (or one overall row if group_vars is NULL).

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(
  gender = factor(sample(c("Male", "Female"), 500, replace = TRUE)),
  region = factor(sample(c("North", "South", "East", "West"), 500, replace = TRUE)),
 occupation = factor(sample(c("Engineer", "Doctor", "Artist", "Teacher"), 500, replace = TRUE)),
  weight = runif(500, 0.5, 1.5)
)

Y <- data.frame(
  gender = factor(sample(c("Male", "Female"), 1000, replace = TRUE)),
  region = factor(sample(c("North", "South", "East", "West"), 1000, replace = TRUE)),
 occupation = factor(sample(c("Engineer", "Doctor", "Artist", "Teacher"), 500, replace = TRUE)),
  weight = runif(1000, 0.5, 1.5)
)

# Overall test (weighted) with simulated p-values
result_overall <- compare_chisq_gof(X, Y, cat_vars = c("gender", "region"),
                                    weight_X = "weight", weight_Y = "weight",
                                    simulate_p = FALSE, B = 2000)
print(result_overall)

# Test conditionally by grouping variables (e.g., "region" and "gender")
result_by_group <- compare_chisq_gof(X, Y,
  cat_vars = c("gender"),
  group_vars = c("region", "occupation"),
  weight_X = "weight", weight_Y = "weight",
  simulate_p = TRUE, B = 2000)
print(result_by_group)
```

`compare_correlation_matrices`

Compare Correlation Matrices between Two Datasets

Description

This function computes and compares the correlation matrices for a specified set of variables between two datasets (X and Y), such as an original dataset and an anonymized/synthetic version. For continuous variables, the user can choose Pearson, Spearman, or robust correlation methods (using either MM-estimation or the MCD estimator). For categorical variables, the function computes Cramér's V. For variable pairs with one continuous and one nominal variable, if the nominal

variable is binary, the point-biserial correlation is computed; otherwise, the correlation ratio (eta squared) is used and transformed into a correlation-like measure.

Usage

```
compare_correlation_matrices(X, ...)

## S3 method for class 'synth_pair'
compare_correlation_matrices(X, ...)

## Default S3 method:
compare_correlation_matrices(
  X,
  Y,
  vars,
  method = "pearson",
  mixed_method = "point_biserial",
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
vars	A character vector specifying the variables to include in the correlation analysis.
method	A character string specifying the correlation method for continuous variables. Options are "pearson", "spearman", "robust_mcd", or "robust_mm". For categorical variable pairs, the function uses Cramér's V.
mixed_method	A character string specifying the method for continuous vs nominal pairs. Options are "point_biserial" (if the nominal variable is binary) or "eta_squared" (for nominal variables with more than two levels). The default is "point_biserial".

Details

The function determines the type of each variable (continuous or categorical) based on its class. Continuous variables are correlated using the specified method. For robust methods, the MCD estimator from the robustbase package or MM-estimation via MASS::cov.rob is used. For categorical variables, Cramér's V is computed from the chi-square statistic. For mixed pairs, if one variable is continuous and the other nominal, point-biserial correlation is used when the nominal variable has two levels, and the correlation ratio (eta squared) is computed otherwise.

Value

A list with three elements:

corr_X The correlation matrix computed from dataset X.

corr_Y The correlation matrix computed from dataset Y.

diff The absolute difference matrix between the two correlation matrices.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  age = rnorm(500, mean = 40, sd = 10),
  gender = factor(sample(c("Male", "Female"), 500, replace = TRUE)),
  region = factor(sample(c("North", "South", "East", "West"), 500, replace = TRUE))
)

Y <- data.frame(
  income = rnorm(1000, mean = 48000, sd = 12000),
  age = rnorm(1000, mean = 42, sd = 11),
  gender = factor(sample(c("Male", "Female"), 1000, replace = TRUE)),
  region = factor(sample(c("North", "South", "East", "West"), 1000, replace = TRUE))
)

# Continuous correlation using Pearson
res1 <- compare_correlation_matrices(X, Y, vars = c("income", "age"), method = "pearson")
print(res1$corr_X)
print(res1$corr_Y)
print(res1$diff)

# Categorical correlation using Cramér's V (computed automatically)
res2 <- compare_correlation_matrices(X, Y, vars = c("gender", "region"), method = "pearson")
print(res2$corr_X)
print(res2$corr_Y)

# Mixed example: income (continuous) vs gender (nominal); point-biserial or eta-squared is computed.
res3 <- compare_correlation_matrices(X, Y,
  vars = c("income", "gender"),
  method = "pearson",
  mixed_method = "point_biserial")
print(res3$corr_X)
print(res3$corr_Y)
```

compare_distributions_cont *Compare continuous distributions conditionally*

Description

This function calculates and compares the empirical cumulative distribution functions (ECDF) of a sample (`X`) and a population (`Y`) for specified variables. It can handle weighted samples and provides options for conditional CDFs, and approximation.

Usage

```
compare_distributions_cont(X, ...)

## S3 method for class 'synth_pair'
compare_distributions_cont(X, ...)

## Default S3 method:
compare_distributions_cont(
  X,
  Y,
  vars = NULL,
  kind = NULL,
  conditional = NULL,
  weights = NULL,
  approx = c(FALSE, TRUE),
  n_approx = 10000,
  bounds = TRUE,
  ...
)
```

Arguments

`X`	A data frame or data table.
`...`	additional arguments passed to methods
`Y`	A data frame or data table to compare with `X`.
`vars`	A character vector specifying the variables for which the comparisons are made. Must be present in both `X` and `Y`. The class of variables determines the default kind of calculation. See details. This argument was formerly named `variables`; that name is still accepted for backward compatibility but `vars` is preferred for consistency with the rest of the package (e.g. `rumap`).
`kind`	To be set when changing the default kind of calculation. See details.
`conditional`	A character vector specifying conditioning variables, or `NULL` if no conditioning is desired. Must be of length 1 if specified.

weights	A character vector of length 1 specifying the weights variable in X and/or Y, or NULL if no weights are used.
approx	A logical vector of length 2 indicating whether to approximate the CDFs in X and Y, respectively. TRUE is recommended for large data.
n_approx	A numeric value specifying the number of points to use for the approximation.
bounds	A logical value indicating whether to use bounds in the CDF calculation.

Details

The following default calculations are taken:

If vars links to one or two numeric variables: the (weighted) ECDF is calculated for both data sets. With kind one can change to "density", "density_bayes", "density_ratio" and "density_ratio_bayes".

If vars links to one, two or three categorical variables: barcharts and mosaic plots are made.

If vars links to one numeric and one categorical variable: boxplots statistics are calculated.

If weights are provided, the weighted versions are computed. The function supports conditional estimates, which are computed by conditioning on the specified variable.

If the approx argument is set to TRUE, the ECDFs are approximated using the specified number of points (n_approx).

Value

A list with class "compare" containing the following components:

formula A formula representing the CDF comparison.

ecdf A data frame with the ECDF values.

kind A character string indicating the type of comparison ("numeric").

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
# Example data 1: Complex sample and population data
S <- data.frame(
  age = sample(20:80, 100, replace = TRUE),
  income = rnorm(100, mean = 50000, sd = 10000),
  weights = runif(100, 0.5, 2)
)
```

```
U <- data.frame(
  age = sample(20:80, 10000, replace = TRUE),
  income = rnorm(10000, mean = 50000, sd = 10000)
)

# Compare ECDFs for age and income
result <- compare_distributions_cont(S, U, vars = c("age", "income"), weights = "weights")

# View the result
print(result$formula)
head(result$ecdf)
plot(result)
plot(result, which = "density")
plot(result, which = "density_bayes")

# Example data 2: Complex sample and complex sample data
X <- data.frame(
  age = sample(20:80, 100, replace = TRUE),
  income = rnorm(100, mean = 50000, sd = 10000),
  weights = runif(100, 0.5, 2)
)
Y <- data.frame(
  age = sample(20:80, 100, replace = TRUE),
  income = rnorm(10000, mean = 50000, sd = 10000),
  weights = runif(100, 0.5, 2)
)

# Compare ECDFs for age and income
result <- compare_distributions_cont(X, Y, vars = c("age", "income"), weights = "weights")

# View the result
print(result$formula)
head(result$ecdf)
plot(result)
plot(result, which = "density")
plot(result, which = "density_bayes")
plot(result, which = "density_ratio")

# Example data 3: Sample and sample data
X <- data.frame(
  age = sample(20:80, 100, replace = TRUE),
  income = rnorm(100, mean = 50000, sd = 10000)
)
Y <- data.frame(
  age = sample(20:80, 100, replace = TRUE),
  income = rnorm(100, mean = 50000, sd = 10000)
)

# Compare ECDFs for age and income
result <- compare_distributions_cont(X, Y, vars = c("age", "income"), n_approx = 100)

# View the result
print(result$formula)
```

```
head(result$ecdf)
plot(result)
plot(result, which = "density")
plot(result, which = "density_bayes")
```

`compare_embedding` *Compare Dimensionality Reduction Methods (t-SNE, UMAP, MDS-Sammon) for Two Datasets*

Description

This function compares two datasets using dimensionality reduction methods: t-distributed Stochastic Neighbor Embedding (t-SNE), Uniform Manifold Approximation and Projection (UMAP), and Multidimensional Scaling (MDS) with Sammon's mapping. It allows visualizing complex data structures, such as clusters or outliers, in two-dimensional space.

Usage

```
compare_embedding(X, ...)

## S3 method for class 'synth_pair'
compare_embedding(X, ...)

## Default S3 method:
compare_embedding(
  X,
  Y,
  vars,
  method = "tsne",
  perplexity = 30,
  n_neighbors = 15,
  side_by_side = FALSE,
  seed = NULL,
  ...
)
```

Arguments

`X, Y`	data.frame or data.table: datasets to compare.
`...`	additional arguments passed to methods
`vars`	Character vector: numeric variables for dimensionality reduction.
`method`	Character: one of 'tsne', 'umap', or 'mds'.
`perplexity`	Numeric: t-SNE perplexity (default: 30).
`n_neighbors`	Numeric: UMAP neighbors (default: 15).
`side_by_side`	Logical: if TRUE, plots embeddings for X and Y side by side.
`seed`	Integer: random seed for reproducibility of stochastic methods (t-SNE, UMAP). Default NULL uses current random state.

Value

A list containing the embeddings and a ggplot2 visualization.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
# MDS (Sammon) embedding - doesn't require optional packages
set.seed(123)
X <- data.frame(
  var1 = rnorm(100, 50, 10),
  var2 = rnorm(100, 40, 8),
  var3 = rnorm(100, 30, 5)
)
Y <- data.frame(
  var1 = rnorm(100, 48, 12),
  var2 = rnorm(100, 42, 9),
  var3 = rnorm(100, 28, 6)
)
res_mds <- compare_embedding(X, Y, vars = c("var1", "var2", "var3"),
                             method = 'mds')


# t-SNE (requires Rtsne package)
if (requireNamespace("Rtsne", quietly = TRUE)) {
  res_tsne <- compare_embedding(X, Y, vars = c("var1", "var2", "var3"),
                                method = 'tsne')
}

# UMAP (requires uwot package)
if (requireNamespace("uwot", quietly = TRUE)) {
  res_umap <- compare_embedding(X, Y, vars = c("var1", "var2", "var3"),
                                method = 'umap')
}
```

`compare_feature_importance` *Compare Feature Importance between Real and Synthetic Data*

Description

Compares feature importance rankings from models trained on real (X) and synthetic/anonymized (Y) datasets.

Usage

```
compare_feature_importance(X, ...)

## S3 method for class 'synth_pair'
compare_feature_importance(X, ...)

## Default S3 method:
compare_feature_importance(
  X,
  Y,
  formula,
  method,
  importance_type = c("model", "permutation", "shap"),
  pred_wrapper = NULL,
  metric = "Accuracy",
  nsim = 5,
  ...
)
```

Arguments

`X`	Original dataset (data.frame or data.table). Must include the target variable.
`...`	additional arguments passed to methods
`Y`	Synthetic/anonymized dataset (data.frame or data.table). Must include the target variable.
`formula`	A formula specifying the target and predictor variables (e.g., `target ~ .`).
`method`	Modeling method (e.g., "rf", "rpart", "glm", etc.).
`importance_type`	Type of importance measure: "model", "permutation", or "shap".
`pred_wrapper`	Prediction function wrapper required for permutation and SHAP importance. If `NULL` and `importance_type = "shap"`, a default wrapper is used which returns the probability of the first class for classification tasks, or raw predictions for regression.
`metric`	Performance metric for permutation importance. For classification tasks, common choices include

- "Accuracy"
- "Kappa"
- "ROC AUC" (if probabilities are provided)

For regression tasks, you can use:

- "RMSE" (Root Mean Squared Error)
- "MAE" (Mean Absolute Error)
- "R-squared"
- "MAPE" (Mean Absolute Percentage Error)

Additionally, a custom metric function can be supplied that takes the observed values and predictions as inputs and returns a single numeric performance value.

nsim — Number of simulations used by permutation importance.

Details

This function supports three types of feature importance measures:

- **Model-based:** Uses caret::varImp.
- **Permutation-based:** Uses vip::vi_permute. A prediction wrapper is required.
- **SHAP-based:** Computes mean absolute Shapley values using the kernelshap package.

- If the target variable in X is a factor, the target variable in Y is coerced to a factor with the same levels.
- For SHAP importance, the function uses the kernelshap package. The returned importance is the mean absolute SHAP value per feature.
- For classification tasks in the SHAP branch, if no pred_wrapper is provided, predictions are taken as the probability of the first class.

Value

A list containing:

comparison — A data.table with columns for feature names, importance values for datasets X and Y, and the absolute difference.

model_X — The model trained on X.

model_Y — The model trained on Y.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
# Model-based importance (Decision Tree) - basic example
set.seed(123)
X <- data.frame(
  age = rnorm(100, 40, 10),
  income = 50000 + rnorm(100, 0, 10000),
  outcome = factor(sample(c("Yes", "No"), 100, replace = TRUE))
)
Y <- data.frame(
  age = rnorm(100, 42, 12),
  income = 48000 + rnorm(100, 0, 12000),
  outcome = factor(sample(c("Yes", "No"), 100, replace = TRUE))
)
res_tree <- compare_feature_importance(X, Y, outcome ~ .,
                                       method = "rpart",
                                       importance_type = "model")


# Permutation importance (requires vip package)
if (requireNamespace("vip", quietly = TRUE)) {
  pred_fn <- function(object, newdata) predict(object, newdata, type = "raw")
  res_perm <- compare_feature_importance(
    X, Y, outcome ~ ., method = "rf",
    importance_type = "permutation",
    pred_wrapper = pred_fn, metric = "Accuracy", nsim = 5
  )
}
```

`compare_histograms` *Compare Histograms of a Numeric Variable in Two Datasets*

Description

This function compares the distribution of a numeric variable between two datasets (e.g., an original dataset and an anonymized/synthetic version) using histograms. It allows conditional faceting on categorical variables and supports different visualization styles, including overlapping, side-by-side, stacked, and separate histograms.

Usage

```
compare_histograms(X, ...)

## S3 method for class 'synth_pair'
compare_histograms(X, ...)

## Default S3 method:
compare_histograms(
```

```
  X,
  Y,
  num_var,
  cat_vars,
  weight_X = NULL,
  weight_Y = NULL,
  facet_type = "wrap",
  bins = 30,
  transparency = 0.4,
  plot_type = "overlap",
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
num_var	A character string specifying the numeric variable to compare.
cat_vars	A character vector of categorical variables used for faceting.
weight_X	Optional. A character string specifying the sampling weight variable in X.
weight_Y	Optional. A character string specifying the sampling weight variable in Y.
facet_type	Character string. The faceting type: "wrap" (default) for facet_wrap() or "grid" for facet_grid().
bins	Integer. Number of bins for the histogram (default: 30).
transparency	Numeric. Transparency level for overlapping histograms (default: 0.4).
plot_type	Character string. Type of plot: • "overlap" (default): Overlapping histograms with transparency. • "side": Side-by-side histograms. • "stack": Stacked histograms. • "separate": Separate histograms for each dataset-category combination.

Value

A ggplot2 histogram visualization comparing the distributions.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(
  age = sample(20:80, 500, replace = TRUE),
  gender = sample(c("Male", "Female"), 500, replace = TRUE),
  income = rnorm(500, mean = 50000, sd = 10000),
  weight = runif(500, 0.5, 1.5)
)

Y <- data.frame(
  age = sample(20:80, 1000, replace = TRUE),
  gender = sample(c("Male", "Female"), 1000, replace = TRUE),
  income = rnorm(1000, mean = 48000, sd = 12000),
  weight = runif(1000, 0.5, 1.5)
)

# Overlapping histograms with transparency
compare_histograms(X, Y, num_var = "income", cat_vars = c("gender"),
                   weight_X = "weight", weight_Y = "weight",
                   facet_type = "wrap", plot_type = "overlap")
# Side-by-side histograms
compare_histograms(X, Y, num_var = "income", cat_vars = c("gender"),
                   weight_X = "weight", weight_Y = "weight",
                   facet_type = "wrap", plot_type = "side")

# Stacked histograms
compare_histograms(X, Y, num_var = "income", cat_vars = c("gender"),
                   weight_X = "weight", weight_Y = "weight",
                   facet_type = "wrap", plot_type = "stack")

# Separate histograms
compare_histograms(X, Y, num_var = "income", cat_vars = c("gender"),
                   weight_X = "weight", weight_Y = "weight",
                   facet_type = "wrap", plot_type = "separate")
```

---

compare_ks_test *Compare Distributions using the Kolmogorov-Smirnov Test*

Description

This function performs a non-parametric Kolmogorov-Smirnov (KS) test to compare the cumulative distribution functions of a numeric variable between an original dataset (X) and an anonymized/synthetic dataset (Y). The test measures the maximum deviation between the two empirical cumulative distribution functions. The function can also perform the test separately for groups defined by one or more categorical variables.

Usage

```
compare_ks_test(X, ...)
```

```
## S3 method for class 'synth_pair'
compare_ks_test(X, ...)

## Default S3 method:
compare_ks_test(X, Y, num_var, cat_vars = NULL, ...)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
num_var	A character string specifying the numeric variable to compare.
cat_vars	Optional. A character vector of categorical variables used for grouping. Default is NULL, in which case the KS test is performed on the entire datasets.

Value

A data.table with columns for the grouping variables (if provided), the KS test statistic, and the corresponding p-value. If no grouping is provided, a data.table with one row is returned.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(
  age = sample(20:80, 500, replace = TRUE),
  gender = sample(c("Male", "Female"), 500, replace = TRUE),
  region = sample(c("North", "South", "East", "West"), 500, replace = TRUE),
  income = rnorm(500, mean = 50000, sd = 10000)
)

Y <- data.frame(
  age = sample(20:80, 1000, replace = TRUE),
  gender = sample(c("Male", "Female"), 1000, replace = TRUE),
  region = sample(c("North", "South", "East", "West"), 500, replace = TRUE),
  income = rnorm(1000, mean = 48000, sd = 12000)
)
```

```
# Perform KS test on the entire datasets
compare_ks_test(X, Y, num_var = "income")

# Perform KS test within groups defined by gender
compare_ks_test(X, Y, num_var = "income", cat_vars = c("gender"))
compare_ks_test(X, Y, num_var = "income", cat_vars = c("gender","region"))
```

`compare_means_frequencies` *Compare Means and Frequencies between Two Datasets*

Description

This function compares the central tendencies of continuous variables and the frequency distributions of categorical variables between two datasets (e.g., an original dataset and an anonymized/synthetic dataset). For continuous variables, the function computes a set of summary statistics for each variable, including arithmetic mean, median, Huber mean, standard deviation, interquartile range (IQR), median absolute deviation (MAD), skewness, and kurtosis. Weighted estimates are computed if sampling weights are provided. In addition, conditional summaries can be produced if grouping variables are specified. For categorical variables, frequency and relative frequency tables are computed, using weights if available.

Usage

```
compare_means_frequencies(X, ...)

## S3 method for class 'synth_pair'
compare_means_frequencies(X, ...)

## Default S3 method:
compare_means_frequencies(
  X,
  Y,
  cont_vars,
  cat_vars,
  group_vars = NULL,
  weight_X = NULL,
  weight_Y = NULL,
 stats = c("mean", "median", "sd", "IQR", "mad", "huber", "skewness", "kurtosis"),
  ...
)
```

Arguments

`X`	A data.frame or data.table containing the original dataset.
`...`	additional arguments passed to methods
`Y`	A data.frame or data.table containing the anonymized/synthetic dataset.

cont_vars	A character vector specifying the continuous variables to compare.
cat_vars	A character vector specifying the categorical variables for frequency comparison.
group_vars	Optional. A character vector specifying the grouping variables for computing conditional summaries. Default is NULL.
weight_X	Optional. A character string specifying the sampling weight variable in X.
weight_Y	Optional. A character string specifying the sampling weight variable in Y.
stats	A character vector of summary measures to compute for continuous variables. Options include "mean", "median", "sd", "IQR", "mad", "huber", "skewness", and "kurtosis". Default is c("mean", "median", "sd", "IQR", "mad", "huber", "skewness", "kurtosis").

Details

For continuous variables, if sampling weights are provided the function computes weighted estimates. The arithmetic mean and standard deviation are computed using `weighted.mean` and a weighted variance formula. The median is computed using `Hmisc::wtd.quantile` when weights are provided. The Huber mean is computed via `MASS::huber` (weighted Huber is not implemented). The IQR and MAD are computed using base functions. For skewness and kurtosis, if weights are provided, the function computes them using custom formulas; otherwise, it computes the unweighted versions.

For categorical variables, frequencies are computed as the sum of weights (if provided) or as raw counts, with relative frequencies calculated accordingly.

Value

A list containing three elements:

continuous_summary A data.table with overall summary statistics for each continuous variable in X and Y.

conditional_summary A data.table with conditional summary statistics computed by the grouping variables (if provided); otherwise, NULL.

categorical_summary A data.table with frequency and relative frequency tables for each categorical variable in X and Y.

Author(s)

Matthias Templ

See Also

Other comparison: `ci_overlap()`, `compare_boxplots()`, `compare_chisq_gof()`, `compare_correlation_matrices()`, `compare_distributions_cont()`, `compare_embedding()`, `compare_feature_importance()`, `compare_histograms()`, `compare_ks_test()`, `compare_missing_values()`, `compare_model_performance()`, `compare_multivariate_distribu`
`compare_multivariate_summary_statistics()`, `compare_outliers()`, `compare_pca()`, `compare_wasserstein()`, `densitydiff_1d_num()`, `densitydiff_kl_num()`, `densitydiff_pca()`

Examples

```
set.seed(123)
X <- data.frame(
  age = sample(20:80, 500, replace = TRUE),
  income = rnorm(500, mean = 50000, sd = 10000),
  gender = sample(c("Male", "Female"), 500, replace = TRUE),
  region = sample(c("North", "South", "East", "West"), 500, replace = TRUE),
  weight = runif(500, 0.5, 1.5)
)

Y <- data.frame(
  age = sample(20:80, 1000, replace = TRUE),
  income = rnorm(1000, mean = 48000, sd = 12000),
  gender = sample(c("Male", "Female"), 1000, replace = TRUE),
  region = sample(c("North", "South", "East", "West"), 1000, replace = TRUE),
  weight = runif(1000, 0.5, 1.5)
)

result <- compare_means_frequencies(X, Y,
             cont_vars = c("age", "income"),
             cat_vars = c("gender", "region"),
             group_vars = c("region"),
             weight_X = "weight", weight_Y = "weight",
          stats = c("mean", "median", "sd", "IQR", "mad", "huber", "skewness", "kurtosis"))

print(result$continuous_summary)
print(result$conditional_summary)
print(result$categorical_summary)
```

---

compare_missing_values

Compare Missing Value Patterns Between Original and Synthetic Data

---

Description

This function compares missing value patterns in an original dataset (X) and a synthetic/anonymized dataset (Y). It assesses whether the synthetic data mimics the same percentage and distribution of missingness across variables.

Usage

```
compare_missing_values(X, ...)

## S3 method for class 'synth_pair'
compare_missing_values(X, ...)

## Default S3 method:
compare_missing_values(X, Y, method = "percentage", ...)
```

Arguments

X	A data frame or matrix of original data.
...	additional arguments passed to methods
Y	A data frame or matrix of synthetic/anonymized data with the same structure as X.
method	Character. The method to use for comparison: "percentage" (default) or "pattern".

Details

Supported methods include:

- "percentage": Computes and compares missing value percentages for each variable.
- "pattern": Summarizes the frequency of missingness patterns across rows.

Both X and Y should be numeric data frames or matrices with the same structure.

Value

An object of class "missingCompare":

- For "percentage": a list with a data frame (summary) reporting per-variable missing counts and percentages.
- For "pattern": a list with two tables (patterns_X and patterns_Y) reporting the frequency of missingness patterns.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_model_performance(), compare_multivariate_distr compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(a = c(rnorm(95), rep(NA, 5)), b = c(rnorm(90), rep(NA, 10)))
Y <- data.frame(a = c(rnorm(95), rep(NA, 5)), b = c(rnorm(90), rep(NA, 10)))

# Compare missing value percentages
res_pct <- compare_missing_values(X, Y, method = "percentage")
print(res_pct)
summary(res_pct)
plot(res_pct)
```

```
# Compare missing value patterns
res_pat <- compare_missing_values(X, Y, method = "pattern")
print(res_pat)
summary(res_pat)
plot(res_pat)
```

`compare_model_performance` *Compare Predictive Model Performance between Two Datasets*

Description

This function evaluates and compares predictive model performance between an original dataset (X) and a synthetic/anonymized dataset (Y). The model is trained separately within each dataset using cross-validation, and various performance metrics are computed.

Usage

```
compare_model_performance(X, ...)

## S3 method for class 'synth_pair'
compare_model_performance(X, ...)

## Default S3 method:
compare_model_performance(
  X,
  Y,
  formula,
  method = "rpart",
 metric = ifelse(is.factor(X[[as.character(formula[[2]])]]), "Accuracy", "RMSE"),
  trControl = NULL,
  ...
)
```

Arguments

`X`	Original data.frame or data.table.
`...`	additional arguments passed to methods
`Y`	Synthetic or anonymized data.frame or data.table.
`formula`	Formula specifying the model structure.
`method`	A string specifying the model type (e.g., 'rf' for random forest, 'rpart' for CART). See `caret` package documentation.
`metric`	Metric for model tuning (default 'Accuracy' for classification, 'RMSE' for regression).
`trControl`	trainControl object from `caret` specifying cross-validation strategy (default 10-fold CV).

Value

List containing trained models, performance metrics for X and Y, and comparison results.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_multivariate_distribu compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
if (requireNamespace("caret", quietly = TRUE)) {
library(caret)
set.seed(123)
X <- data.frame(income = rnorm(500, 50000, 10000),
                age = rnorm(500, 40, 10),
                gender = factor(sample(c("M", "F"), 500, replace = TRUE)),
                target = factor(sample(c("Yes", "No"), 500, replace = TRUE)))

Y <- data.frame(income = rnorm(500, 48000, 12000),
                age = rnorm(500, 42, 11),
                gender = factor(sample(c("M", "F"), 500, replace = TRUE)),
                target = factor(sample(c("Yes", "No"), 500, replace = TRUE)))

result <- compare_model_performance(X, Y,
           formula = target ~ income + age + gender,
           method = 'rpart',
           metric = 'Accuracy')

print(result$comparison)
}
```

---

compare_multivariate_distribution

Compare Multivariate Distributions using Mahalanobis Distance or Jensen-Shannon Divergence

---

Description

This function compares the joint distribution of a set of variables between two datasets (X and Y) using one of two methods. For continuous variables, it computes the weighted (if weights are provided) Mahalanobis distance between the mean vectors, using a pooled covariance matrix (or its generalized inverse). For nominal variables, it discretizes the data (if necessary) and computes a divergence based on the Jensen-Shannon divergence of the joint frequency distributions, which serves as a symmetric measure of distributional difference.

Usage

```
compare_multivariate_distribution(X, ...)

## S3 method for class 'synth_pair'
compare_multivariate_distribution(X, ...)

## Default S3 method:
compare_multivariate_distribution(
  X,
  Y,
  vars,
  method = "mahalanobis",
  weight_X = NULL,
  weight_Y = NULL,
  n_bins = 10,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
vars	A character vector specifying the variables to compare.
method	A character string indicating which method to use. Either "mahalanobis" (default) or "mutual_information".
weight_X	Optional. A character string specifying the sampling weight variable in X (used only for method "mahalanobis").
weight_Y	Optional. A character string specifying the sampling weight variable in Y (used only for method "mahalanobis").
n_bins	Numeric. The number of bins to use when discretizing variables for the mutual_information method. Default is 10.

Details

For the "mahalanobis" method, if sampling weights are provided, the function computes the weighted mean vector for each dataset and a weighted pooled covariance matrix. The Mahalanobis distance

is then computed as

$$D_M = \sqrt{(\mu_X - \mu_Y)^T S^{-1} (\mu_X - \mu_Y)}$$

where S is the pooled covariance matrix (or its generalized inverse if singular). For the "mutual_information" method, numeric variables are discretized into `n_bins` bins using `cut()`, and joint frequency tables are constructed. The tables are normalized into probability vectors, and the Jensen-Shannon divergence is computed as

$$JS(p, q) = \frac{1}{2} KL(p||m) + \frac{1}{2} KL(q||m)$$

where $m = \frac{1}{2}(p + q)$ and KL is the Kullback-Leibler divergence.

Value

A list containing:

distance The computed distance measure (Mahalanobis distance or Jensen-Shannon divergence).

method The method used ("mahalanobis" or "mutual_information").

additional A list of additional computed quantities (e.g., weighted means and pooled covariance for Mahalanobis, or probability vectors for mutual_information).

Author(s)

Matthias Templ

See Also

Other comparison: `ci_overlap()`, `compare_boxplots()`, `compare_chisq_gof()`, `compare_correlation_matrices()`, `compare_distributions_cont()`, `compare_embedding()`, `compare_feature_importance()`, `compare_histograms()`, `compare_ks_test()`, `compare_means_frequencies()`, `compare_missing_values()`, `compare_model_performance()`, `compare_multivariate_summary_statistics()`, `compare_outliers()`, `compare_pca()`, `compare_wasserstein()`, `densitydiff_1d_num()`, `densitydiff_kl_num()`, `densitydiff_pca()`

Examples

```
# Continuous example using Mahalanobis distance (weighted)
set.seed(123)
X <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  age = rnorm(500, mean = 40, sd = 10),
  weight = runif(500, 0.5, 1.5)
)
Y <- data.frame(
  income = rnorm(1000, mean = 48000, sd = 12000),
  age = rnorm(1000, mean = 42, sd = 11),
  weight = runif(1000, 0.5, 1.5)
)
result_mahal <- compare_multivariate_distribution(X, Y, vars = c("income", "age"),
                                                  method = "mahalanobis",
                                                weight_X = "weight", weight_Y = "weight")
```

```
print(result_mahal)

# Nominal example using mutual information (Jensen-Shannon divergence)
set.seed(456)
X_nom <- data.frame(
  gender = factor(sample(c("Male", "Female"), 500, replace = TRUE)),
  region = factor(sample(c("North", "South", "East", "West"), 500, replace = TRUE))
)
Y_nom <- data.frame(
  gender = factor(sample(c("Male", "Female"), 1000, replace = TRUE)),
  region = factor(sample(c("North", "South", "East", "West"), 1000, replace = TRUE))
)
result_js <- compare_multivariate_distribution(X_nom, Y_nom, vars = c("gender", "region"),
                                               method = "mutual_information", n_bins = 5)
print(result_js)
```

`compare_multivariate_summary_statistics`
Compare Multivariate Summary Statistics between Two Datasets

Description

This function computes higher-order summary statistics for multiple variables in two datasets (e.g., an original dataset and an anonymized/synthetic dataset) to assess whether the synthetic data replicate the joint properties of the original data. For continuous variables, the function calculates the multivariate mean vector, variance–covariance matrix, and correlation matrix. When sampling weights are provided, weighted estimates are computed. For categorical variables, joint frequency tables (and relative frequencies) are constructed. The function returns the statistics for each dataset along with the differences between them.

Usage

```
compare_multivariate_summary_statistics(X, ...)

## S3 method for class 'synth_pair'
compare_multivariate_summary_statistics(X, ...)

## Default S3 method:
compare_multivariate_summary_statistics(
  X,
  Y,
  cont_vars,
  cat_vars,
  weight_X = NULL,
  weight_Y = NULL,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
cont_vars	A character vector specifying the continuous variables to compare.
cat_vars	A character vector specifying the categorical variables for joint frequency comparison.
weight_X	Optional. A character string specifying the sampling weight variable in X.
weight_Y	Optional. A character string specifying the sampling weight variable in Y.

Details

For continuous variables, if sampling weights are provided the function computes weighted means using `weighted.mean` and weighted covariance matrices using a custom function. The correlation matrices are derived from the covariance matrices. For categorical variables, if weights are provided the frequencies are computed as the sum of weights; otherwise, raw counts are used. Joint frequency tables are normalized to obtain relative frequencies.

Value

A list with two elements:

continuous A list containing:

- **mean_X** The multivariate mean vector for X.
- **mean_Y** The multivariate mean vector for Y.
- **mean_diff** The difference between the mean vectors (X minus Y).
- **cov_X** The variance–covariance matrix for X.
- **cov_Y** The variance–covariance matrix for Y.
- **cov_diff** The difference between the covariance matrices (X minus Y).
- **cor_X** The correlation matrix for X.
- **cor_Y** The correlation matrix for Y.
- **cor_diff** The absolute difference between the correlation matrices.

categorical A list containing:

- **freq_X** The joint frequency table for the categorical variables in X.
- **freq_Y** The joint frequency table for the categorical variables in Y.
- **rel_freq_X** The relative frequency table for X.
- **rel_freq_Y** The relative frequency table for Y.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  age = rnorm(500, mean = 40, sd = 10),
  gender = factor(sample(c("Male", "Female"), 500, replace = TRUE)),
  region = factor(sample(c("North", "South", "East", "West"), 500, replace = TRUE)),
  weight = runif(500, 0.5, 1.5)
)

Y <- data.frame(
  income = rnorm(1000, mean = 48000, sd = 12000),
  age = rnorm(1000, mean = 42, sd = 11),
  gender = factor(sample(c("Male", "Female"), 1000, replace = TRUE)),
  region = factor(sample(c("North", "South", "East", "West"), 1000, replace = TRUE)),
  weight = runif(1000, 0.5, 1.5)
)

result <- compare_multivariate_summary_statistics(X, Y,
              cont_vars = c("income", "age"),
              cat_vars = c("gender", "region"),
              weight_X = "weight", weight_Y = "weight")

print(result$continuous)
print(result$categorical)
```

---

compare_outliers *Compare Outlier Detection Between Original and Synthetic Data*

Description

This function compares the prevalence of outliers in an original dataset (X) and an anonymized/synthetic dataset (Y) using various outlier detection methods. Supported methods include:

- "zscore": Uses z-scores (default threshold = 3) to flag outliers.
- "iqr": Flags values outside 1.5 times the interquartile range.
- "dbscan": Uses DBSCAN clustering to identify noise points as outliers.
- "robust": Uses robust Mahalanobis distances based on the Minimum Covariance Determinant.

Usage

```
compare_outliers(X, ...)

## S3 method for class 'synth_pair'
compare_outliers(X, ...)

## Default S3 method:
compare_outliers(X, Y, method = "zscore", threshold = NULL, ...)
```

Arguments

X	A data frame or matrix of original data with numeric columns.
...	additional arguments passed to the underlying method (e.g., eps and minPts for DBSCAN).
Y	A data frame or matrix of anonymized/synthetic data with the same structure as X.
method	Character. Outlier detection method: "zscore", "iqr", "dbscan", or "robust".
threshold	Numeric. For "zscore", the absolute z-score cutoff (default = 3). For "robust", the significance level for the chi-square cutoff (default = 0.975). Ignored for "iqr" and "dbscan".

Details

Input datasets must be numeric and of the same structure. For multivariate methods, the comparison is performed on the entire dataset.

Value

A list with:

- summary: A data frame summarizing outlier counts and proportions in X and Y.
- details: Method-specific details (e.g., per-variable counts or clustering output).

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(a = rnorm(100), b = rnorm(100))
Y <- data.frame(a = rnorm(100, mean = 0.1), b = rnorm(100, mean = -0.1))

# Using z-score method (no extra packages required)
res_z <- compare_outliers(X, Y, method = "zscore", threshold = 3)

# Using IQR method
res_iqr <- compare_outliers(X, Y, method = "iqr")

# Using DBSCAN (requires the 'dbscan' package)
if (requireNamespace("dbscan", quietly = TRUE)) {
  res_db <- compare_outliers(X, Y, method = "dbscan", eps = 0.5, minPts = 5)
}

# Using robust Mahalanobis (requires the 'robustbase' package)
if (requireNamespace("robustbase", quietly = TRUE)) {
  res_robust <- compare_outliers(X, Y, method = "robust", threshold = 0.975)
}
```

---

compare_pca *Compare Principal Component Analysis (PCA) between Two Datasets with Separate Loadings*

Description

This function performs Principal Component Analysis (PCA) on two datasets (X and Y) for a specified set of numeric variables. When side_by_side is FALSE, a combined PCA is performed on the union of the data for visualization of the point scores, while separate PCA analyses are computed for X and Y to obtain loadings. The function then overlays loadings arrows from X (blue) and Y (red) onto the combined PCA biplot. When side_by_side is TRUE, separate biplots for X and Y are produced and arranged side by side.

Usage

```
compare_pca(X, ...)

## S3 method for class 'synth_pair'
compare_pca(X, ...)

## Default S3 method:
compare_pca(
  X,
  Y,
  vars,
  center = TRUE,
```

```
  scale = TRUE,
  biplot = TRUE,
  side_by_side = FALSE,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
vars	A character vector specifying the numeric variables to include in the PCA.
center	Logical. Should the variables be centered (zero mean)? Default is TRUE.
scale	Logical. Should the variables be scaled to unit variance? Default is TRUE.
biplot	Logical. If TRUE, loadings are added as arrows. Default is TRUE.
side_by_side	Logical. If TRUE, separate PCA biplots are produced for X and Y and arranged side by side. If FALSE (default), a combined PCA biplot is produced with separate loadings for X and Y.

Details

When side_by_side is FALSE, the function first adds a dataset indicator to X and Y, combines them, and performs PCA on the combined dataset for the point scores. Then, PCA is computed separately for X and Y. Loadings from these separate analyses are scaled using the range of the combined PCA scores and overlaid on the combined biplot with blue arrows for X and red arrows for Y.

Value

A list with two elements:

pca If side_by_side = FALSE, a list with three PCA objects: combined, X, and Y. If side_by_side = TRUE, a list with two PCA objects: X and Y.

plot A ggplot2 object: either a combined biplot with separate loadings or separate plots arranged side by side.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  age = rnorm(500, mean = 40, sd = 10)
)
# Add a positively correlated variable
X$expenses <- X$income * 0.5 + rnorm(n = 500, mean = 1000, sd = 500)
Y <- data.frame(
  income = rnorm(1000, mean = 48000, sd = 12000),
  age = rnorm(1000, mean = 42, sd = 11)
)
Y$expenses <- Y$income * 0.5 + rnorm(n = 1000, mean = 1000, sd = 500)
# Combined PCA biplot with separate loadings for X (blue) and Y (red)
res_combined <- compare_pca(X, Y,
  vars = c("income", "age", "expenses"),
  biplot = TRUE, side_by_side = FALSE)
print(res_combined$pca)
print(res_combined$plot)

# Separate PCA biplots for X and Y arranged side by side
res_separate <- compare_pca(X, Y, vars = c("income", "age"), biplot = TRUE, side_by_side = TRUE)
print(res_separate$pca)
print(res_separate$plot)
```

---

compare_wasserstein *Compare Distributions using the Wasserstein Distance*

Description

This function computes the one-dimensional Wasserstein distance between a numeric variable in two datasets, X (original) and Y (anonymized or synthetic). For continuous variables, the Wasserstein distance is approximated by integrating the absolute differences between the quantile functions computed on a grid of probabilities. If sampling weights are provided, weighted quantiles are computed using `Hmisc::wtd.quantile`. For nominal (categorical) variables, the function computes the total variation distance (half the L1 distance between the probability distributions) assuming a unit cost for mismatches.

Usage

```
compare_wasserstein(X, ...)

## S3 method for class 'synth_pair'
compare_wasserstein(X, ...)

## Default S3 method:
compare_wasserstein(
  X,
```

```
  Y,
  num_var,
  cat_vars = NULL,
  weight_X = NULL,
  weight_Y = NULL,
  var_type = "auto",
  n_grid = 1000,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods
Y	A data.frame or data.table containing the anonymized/synthetic dataset.
num_var	A character string specifying the numeric (or nominal) variable to compare.
cat_vars	Optional. A character vector of categorical variables for grouping. Default is NULL.
weight_X	Optional. A character string specifying the sampling weight variable in X.
weight_Y	Optional. A character string specifying the sampling weight variable in Y.
var_type	A character string specifying the type of variable: "continuous" or "nominal". Default is "auto", which infers "continuous" if the variable is numeric, else "nominal".
n_grid	Number of grid points for quantile approximation (for continuous variables). Default is 1000.

Details

When grouping variables are provided (via cat_vars), the Wasserstein distance is computed within each group.

Value

A data.table with the computed Wasserstein distance for each group (or overall if no grouping is provided). For continuous variables, the Wasserstein distance is approximated by the mean absolute difference between the quantile functions. For nominal variables, the result is the total variation distance.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(),

compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), densitydiff_1d_num(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
set.seed(123)
X <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  gender = sample(c("Male", "Female"), 500, replace = TRUE),
  region = sample(c("North", "South", "East", "West"), 500, replace = TRUE),
  weight = runif(500, 0.5, 1.5)
)

Y <- data.frame(
  income = rnorm(1000, mean = 48000, sd = 12000),
  gender = sample(c("Male", "Female"), 1000, replace = TRUE),
  region = sample(c("North", "South", "East", "West"), 1000, replace = TRUE),
  weight = runif(1000, 0.5, 1.5)
)

# Compare overall continuous distributions (weighted)
compare_wasserstein(X, Y, num_var = "income", var_type = "continuous",
                    weight_X = "weight", weight_Y = "weight")

# Compare distributions within groups defined by gender
compare_wasserstein(X, Y, num_var = "income", cat_vars = c("gender"),
                    var_type = "continuous", weight_X = "weight", weight_Y = "weight")
compare_wasserstein(X, Y, num_var = "income", cat_vars = c("gender","region"),
                    var_type = "continuous", weight_X = "weight", weight_Y = "weight")

# Example for nominal variables

# Create a synthetic dataset X with a nominal variable "gender"
set.seed(123)
X_nom <- data.frame(
  gender = sample(c("Male", "Female"), 500, replace = TRUE),
  weight = runif(500, 0.5, 1.5)
)

# Create a synthetic dataset Y with a nominal variable "gender"
Y_nom <- data.frame(
  gender = sample(c("Male", "Female"), 1000, replace = TRUE),
  weight = runif(1000, 0.5, 1.5)
)

# Compare the distributions of the nominal variable using the Wasserstein distance.
# Here, var_type is explicitly set to "nominal" so that the function uses frequency tables.
result_nominal <- compare_wasserstein(X_nom, Y_nom, num_var = "gender",
                                      var_type = "nominal",
                                      weight_X = "weight", weight_Y = "weight")

print(result_nominal)
```

`confint.rapid` *Confidence Intervals for RAPID Risk Estimates*

Description

Computes confidence intervals for the RAPID disclosure risk measure (proportion of records at risk). Three methods are available: Wald (normal approximation), Wilson (score interval, recommended), and bootstrap (percentile).

Usage

```
## S3 method for class 'rapid'
confint(
  object,
  parm = NULL,
  level = 0.95,
  method = c("wilson", "wald", "bootstrap"),
  n_bootstrap = 1000,
  seed = NULL,
  ...
)
```

Arguments

`object`	An object of class `"rapid"`.
`parm`	Unused; included for compatibility with the generic.
`level`	Confidence level (default 0.95).
`method`	One of `"wilson"` (default), `"wald"`, or `"bootstrap"`.
`n_bootstrap`	Number of bootstrap replicates (default 1000). Only used when `method = "bootstrap"`.
`seed`	Optional random seed for the bootstrap.
`...`	Additional arguments (ignored).

Details

The RAPID score is a sample proportion (number at risk / total). Because the model is trained on separate synthetic data and applied to the original records, the per-record risk indicators are conditionally i.i.d. Bernoulli, making binomial confidence intervals valid.

wald Standard normal approximation $\hat{p} \pm z_{\alpha/2}\sqrt{\hat{p}(1-\hat{p})/n}$. Can produce intervals outside $[0, 1]$.

wilson Score interval (Wilson, 1927). Has better coverage near 0 and 1; recommended as the default.

bootstrap Resamples the `at_risk` indicators from `object$records` and returns percentile confidence limits. No model refitting is needed.

Value

A 1×2 matrix with columns for the lower and upper confidence limits, following the standard `confint` convention.

Author(s)

Oscar Thees, Matthias Templ

References

Wilson, E.B. (1927). Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, 22(158), 209–212.

See Also

`rapid`, `rapid_test`

Other rapid: `rapid()`, `rapid_synthesizer_cv()`, `rapid_test()`, `rapid_threshold_select()`

Examples

```
# Small runnable example
set.seed(42)
X <- data.frame(
  age = sample(20:60, 80, replace = TRUE),
  sex = sample(c("M", "F"), 80, replace = TRUE),
  income = rnorm(80, 50000, 10000)
)
Y <- X
Y$income <- Y$income + rnorm(80, 0, 5000)
r <- rapid(X, Y, key_vars = c("age", "sex"),
           target_var = "income", model_type = "lm")
confint(r)
confint(r, method = "wald")


# With random forest model and bootstrap CI
r2 <- rapid(X, Y, key_vars = c("age", "sex"),
            target_var = "income", model_type = "rf")
confint(r2)
confint(r2, method = "bootstrap", n_bootstrap = 500)
```

`contingency_fidelity` *Contingency Table Fidelity for Categorical Dependence Comparison*

Description

Compares bivariate contingency tables (joint distributions) for all pairs of categorical variables between original and synthetic data. For each pair, the total variation (TV) distance between the proportion tables is computed. The mean TV distance across all pairs summarizes how well the synthetic data preserves the bivariate categorical dependence structure.

Usage

```
contingency_fidelity(X, ...)

## S3 method for class 'synth_pair'
contingency_fidelity(X, ...)

## Default S3 method:
contingency_fidelity(X, Y, vars = NULL, na.rm = TRUE, ...)
```

Arguments

`X`	A data.frame or data.table containing the original dataset.
`...`	additional arguments passed to methods (currently unused)
`Y`	A data.frame or data.table containing the synthetic/anonymized dataset.
`vars`	Character vector of categorical (factor/character) variable names to compare. If NULL (default), all common categorical variables are used. Numeric variables are skipped with a message.
`na.rm`	Logical, whether to remove rows with NA values. Default TRUE.

Details

Use this when the data contains multiple categorical variables and you want to check whether their bivariate dependence structure is preserved. This complements `copula_fidelity`, which handles numeric pairs.

The algorithm proceeds as follows:

1. Auto-detect categorical variables (factor or character). Numeric variables are skipped with an informational message.
2. For each pair of categorical variables (i, j):
   - Factor levels are aligned: the union of levels from both datasets is used so that cells present in one dataset but not the other receive a count of zero.
   - The bivariate contingency table (proportions) is computed for both original and synthetic data.

- The total variation distance is computed as

$$TV = \frac{1}{2} \sum_{c} |P_{\text{orig}}(c) - P_{\text{syn}}(c)|$$

where the sum runs over all cells c in the cross-tabulation.

3. The overall fidelity is summarized as `mean_tv`, the mean of all pairwise TV distances. The utility score is `1 - mean_tv`.

Total variation distance is bounded in $[0, 1]$: 0 means identical joint distributions and 1 means completely non-overlapping distributions. Pairs where a variable has only a single level in both datasets are skipped and receive `NA`.

Interpretation (heuristic thresholds):

- utility_score > 0.95: EXCELLENT – dependence structure very well preserved
- utility_score > 0.80: GOOD – reasonably preserved
- utility_score > 0.50: MODERATE – some differences
- utility_score <= 0.50: POOR – significant differences

Note that this measure only captures bivariate (pairwise) categorical associations. Higher-order interactions (3-way or more) are not assessed.

Value

An object of class `"contingency_fidelity"` containing:

- `mean_tv`: mean of all pairwise total variation distances
- `utility_score`: `1 - mean_tv`, in $[0, 1]$ (higher = better)
- `pairwise`: data.frame with columns `var1`, `var2`, `tv_distance` for each pair of variables
- `n_vars`: number of categorical variables used
- `vars`: names of categorical variables used
- `n_X`: number of rows in X (after NA removal)
- `n_Y`: number of rows in Y (after NA removal)

Author(s)

Matthias Templ

References

Snoke, J., Raab, G. M., Nowok, B., Dibben, C., and Slavkovic, A. (2018). General and Specific Utility Measures for Synthetic Data. Journal of the Royal Statistical Society: Series A, 181(3), 663-688.

See Also

copula_fidelity for numeric dependence comparison, compare_chisq_gof for univariate categorical comparison, hellinger for univariate Hellinger distance, regression_fidelity for analysis-specific fidelity, subgroup_utility for stratified utility assessment

Other utility: chisq_utility(), ci_proximity(), copula_fidelity(), energy_distance(), gower(), hellinger(), mmd(), mqs(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), rumap(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
set.seed(42)
n <- 500
# Original data with dependence between cat variables
gender <- sample(c("M", "F"), n, replace = TRUE)
region <- sample(c("N", "S", "E", "W"), n, replace = TRUE)
edu <- ifelse(gender == "M",
              sample(c("low", "mid", "high"), n, replace = TRUE,
                     prob = c(0.2, 0.5, 0.3)),
              sample(c("low", "mid", "high"), n, replace = TRUE,
                     prob = c(0.4, 0.4, 0.2)))
X <- data.frame(gender = gender, region = region, education = edu,
                stringsAsFactors = TRUE)

# Good synthetic data (preserves dependence)
gender2 <- sample(c("M", "F"), n, replace = TRUE)
edu2 <- ifelse(gender2 == "M",
               sample(c("low", "mid", "high"), n, replace = TRUE,
                      prob = c(0.2, 0.5, 0.3)),
               sample(c("low", "mid", "high"), n, replace = TRUE,
                      prob = c(0.4, 0.4, 0.2)))
Y_good <- data.frame(gender = gender2, region = sample(region),
                     education = edu2, stringsAsFactors = TRUE)

result <- contingency_fidelity(X, Y_good)
print(result)
summary(result)

# Using synth_pair
pair <- synth_pair(X, Y_good)
result2 <- contingency_fidelity(pair)

# Selecting specific variables
result3 <- contingency_fidelity(X, Y_good, vars = c("gender", "education"))


plot(result)
```

`copula_fidelity` *Copula Fidelity for Dependence Structure Comparison*

Description

Compares the dependence structure between original and synthetic data using empirical copulas. For each pair of numeric variables, the bivariate empirical copula is estimated via rank-transformation and compared using the Cramer-von Mises (CvM) statistic on a grid. The mean pairwise CvM distance provides an overall measure of how well the synthetic data preserves the joint dependence structure.

Usage

```
copula_fidelity(X, ...)

## S3 method for class 'synth_pair'
copula_fidelity(X, ...)

## Default S3 method:
copula_fidelity(X, Y, vars = NULL, n_grid = 50L, na.rm = TRUE, ...)
```

Arguments

`X`	A data.frame or data.table containing the original dataset.
`...`	additional arguments passed to methods (currently unused)
`Y`	A data.frame or data.table containing the synthetic/anonymized dataset.
`vars`	Character vector of numeric variable names to compare. If NULL (default), all common numeric variables are used. Categorical variables are skipped with a message.
`n_grid`	Integer, grid resolution for CvM evaluation. Default 50. Higher values give more accurate CvM estimates but increase computation.
`na.rm`	Logical, whether to remove rows with NA values. Default TRUE.

Details

The algorithm proceeds as follows:

1. Each numeric variable is rank-transformed to $[0, 1]$: `rank(x) / (length(x) + 1)`.
2. For each pair of variables (i, j), the bivariate empirical CDF is computed for both original and synthetic data:
$$F(u_1, u_2) = \frac{1}{n} \sum_{k=1}^{n} 1(U_{k,i} \leq u_1, U_{k,j} \leq u_2)$$
3. The Cramer-von Mises statistic measures the integrated squared difference between the two empirical copulas on a regular grid:
$$CvM_{ij} = \text{mean}\left((F_{\text{orig}}(u_1, u_2) - F_{\text{syn}}(u_1, u_2))^2\right)$$

4. The overall fidelity is summarized as the mean of all pairwise CvM distances. A utility score is computed as `1 / (1 + mean_cvm * 100)`.

This measure is invariant to monotone transformations of individual variables (since only ranks matter) and focuses purely on the dependence structure. It complements marginal distribution comparisons (e.g., Wasserstein, Hellinger) by assessing whether the joint structure is preserved.

Value

An object of class `"copula_fidelity"` containing:

- `mean_cvm`: mean of all pairwise CvM distances
- `utility_score`: transformed score in $[0, 1]$ (higher = better), computed as `1 / (1 + mean_cvm * 100)`
- `pairwise`: data.frame with columns `var1`, `var2`, `cvm_distance` for each pair of variables
- `n_vars`: number of numeric variables used
- `vars`: names of numeric variables used
- `n_X`: number of rows in X (after NA removal)
- `n_Y`: number of rows in Y (after NA removal)
- `n_grid`: grid resolution used

Author(s)

Matthias Templ

References

Genest, C. and Remillard, B. (2008). Validity of the parametric bootstrap for goodness-of-fit testing in semiparametric models. Annales de l'Institut Henri Poincare, Probabilites et Statistiques, 44(6), 1096-1127.

Deheuvels, P. (1979). La fonction de dependance empirique et ses proprietes. Un test non parametrique d'independance. Academie Royale de Belgique. Bulletin de la Classe des Sciences, 5e Serie, 65, 274-292.

See Also

`compare_correlation_matrices` for linear dependence, `energy_distance` for multivariate distribution comparison, `mmd` for kernel-based comparison

Other utility: `chisq_utility()`, `ci_proximity()`, `contingency_fidelity()`, `energy_distance()`, `gower()`, `hellinger()`, `mmd()`, `mqs()`, `pMSE()`, `plot.rumap()`, `propscore()`, `regression_fidelity()`, `rumap()`, `specks()`, `subgroup_utility()`, `tail_fidelity()`, `tstr()`

Examples

```
set.seed(123)
# Original data with dependence
n <- 300
x1 <- rnorm(n)
x2 <- 0.8 * x1 + rnorm(n, sd = 0.6)
x3 <- rnorm(n)
X <- data.frame(x1 = x1, x2 = x2, x3 = x3)

# Good synthetic data (preserves dependence)
s1 <- rnorm(n)
s2 <- 0.8 * s1 + rnorm(n, sd = 0.6)
s3 <- rnorm(n)
Y_good <- data.frame(x1 = s1, x2 = s2, x3 = s3)

# Poor synthetic data (independent variables)
Y_poor <- data.frame(x1 = rnorm(n), x2 = rnorm(n), x3 = rnorm(n))

result_good <- copula_fidelity(X, Y_good)
print(result_good)

result_poor <- copula_fidelity(X, Y_poor)
print(result_poor)


# Heatmap of pairwise copula distances
plot(result_poor)
```

---

dcap *Correct Attribution Probability (CAP/DCAP)*

Description

Computes the Correct Attribution Probability for synthetic data disclosure risk. Measures the probability that an adversary can correctly infer a sensitive target variable using known quasi-identifier (key) attributes.

Usage

```
dcap(X, ...)

## S3 method for class 'synth_pair'
dcap(X, ...)

## Default S3 method:
dcap(
  X,
  Y,
```

```
  key_vars,
  target_var,
  method = c("exact", "gower"),
  gower_threshold = 0.1,
  cont_bins = 10,
  na.rm = TRUE,
  ...
)
```

Arguments

X	data frame of original data, or a synth_pair object
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic/anonymized data (not needed if X is a synth_pair)
key_vars	character vector of quasi-identifier variable names
target_var	character, name of the sensitive target variable
method	character, matching method: "exact" or "gower" (default: "exact")
gower_threshold	numeric, maximum Gower distance for a match (default: 0.1)
cont_bins	integer, number of bins for continuous target variables (default: 10)
na.rm	logical, remove records with NA in key or target (default: TRUE)

Details

The Correct Attribution Probability (CAP) measures attribute disclosure risk in synthetic data. For each original record, it finds matching synthetic records based on key attributes and computes the probability that the correct target value appears among the matches.

$$CAP_i = \frac{|\{j \in Y : keys_j = keys_i \wedge target_j = target_i\}|}{|\{j \in Y : keys_j = keys_i\}|}$$

Higher CAP values indicate higher disclosure risk. A CAP of 1 means the intruder can perfectly infer the target; a CAP equal to baseline means no information leakage beyond random guessing.

Two summaries are returned: the raw mean CAP (cap) and the *differential* CAP (dcap $= \overline{CAP} - baseline$), which subtracts the baseline so that values near 0 indicate no leakage beyond random guessing and larger values indicate more attribute disclosure (Taub et al. 2018).

Value

An object of class "dcap" containing:

- cap_scores: per-record CAP values
- cap: mean CAP (raw correct-attribution probability; synthpop's CAPd)
- cap_median: median CAP
- dcap: differential CAP, cap - baseline (attribution risk above random guessing; Taub et al. 2018)

- dcap_median: `cap_median - baseline`
- n_matched: records with at least one synthetic match
- n_unmatched: records with no matches
- baseline: expected CAP under random guessing
- key_vars, target_var, method: input parameters

Baseline computation

The baseline is computed as the maximum target category frequency in the synthetic data (Y), representing the best guess an intruder could make without any key information.

Note: This differs from synthpop's baseline computation, which uses the original data (X). The riskutility approach is more conservative: it measures what an intruder learns from synthetic data specifically, rather than from the original distribution. Both approaches are valid but answer slightly different questions.

Gower threshold selection

When `method = "gower"`, the `gower_threshold` parameter determines how close records must be to be considered "matching." Guidelines:

- **0.05**: Very strict matching, few matches, may miss disclosure
- **0.1** (default): Balanced threshold, suitable for most cases
- **0.2**: More permissive, catches near-matches
- **0.3+**: Very permissive, may inflate risk estimates

Consider the scale and nature of your quasi-identifiers when choosing. For datasets with many categorical variables, lower thresholds work well. For continuous-heavy data, slightly higher thresholds may be appropriate.

synth_pair method

If `X` is a `synth_pair` object, the function extracts original, synthetic, key_vars, and target_var from the object. The synth_pair must have `key_vars` and `target_var` set.

Author(s)

Matthias Templ

References

Taub, J., Elliot, M., Pampaka, M., & Smith, D. (2018). Differential Correct Attribution Probability for Synthetic Data: An Exploration. *Privacy in Statistical Databases*, 122-137.

See Also

`synth_pair` for creating a comparison pair, `tcap` for per-record CAP, `disco` for disclosive records

Other attribution-risk: `disco()`, `tcap()`, `weap()`

Examples

```
# Create example data
set.seed(123)
X <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 100, replace = TRUE),
  income = sample(c("low", "medium", "high"), 100, replace = TRUE)
)
# Synthetic version with some noise
Y <- X
Y$income <- sample(Y$income)  # Shuffle target

# Compute DCAP (traditional API)
result <- dcap(X, Y,
               key_vars = c("age", "gender", "region"),
               target_var = "income")
print(result)
summary(result)

# Using synth_pair (container API)
pair <- synth_pair(X, Y,
                   key_vars = c("age", "gender", "region"),
                   target_var = "income")
result2 <- dcap(pair)
```

---

dcr *Distance to Closest Record (DCR)*

Description

Computes the Distance to Closest Record privacy metric for synthetic data. DCR compares the distances from synthetic records to their nearest neighbors in the training data versus a holdout set to detect potential privacy leaks.

Usage

```
dcr(X, ...)

## S3 method for class 'synth_pair'
dcr(X, ...)

## Default S3 method:
dcr(
  X,
  Y,
  holdout = NULL,
  holdout_fraction = 0.5,
```

```
  vars = NULL,
  method = c("gower", "euclidean"),
  na.rm = TRUE,
  seed = NULL,
  progress = FALSE,
  null_test = TRUE,
  n_null = 100,
  ...
)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
holdout	data frame of holdout data (optional). If NULL, a holdout set is automatically created by splitting X.
holdout_fraction	numeric, fraction of X to use as holdout if holdout is NULL (default: 0.5). Larger holdouts give more reliable results, especially for smaller datasets. Use 0.1 only for very large datasets.
vars	character vector of variable names to use for distance calculation. If NULL (default), all common variables between X, Y, and holdout are used.
method	character, distance method: "gower" (default, handles mixed types) or "euclidean" (numerical variables only)
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for holdout sampling (default: NULL)
progress	logical, show progress bar for long computations (default: FALSE)
null_test	logical, perform permutation test comparing observed DCR share against a null distribution (default: TRUE). Permutes training/holdout assignment to estimate expected share under no memorization.
n_null	integer, number of permutation samples for null distribution estimation (default: 100). Only used when null_test = TRUE.

Details

Distance to Closest Record (DCR) is a privacy metric that detects whether synthetic data has memorized or copied training records. The key insight is that synthetic records should be equally close to training and holdout data if no information leakage occurred.

The metric works by:

1. Splitting original data into training and holdout (or using provided holdout)
2. For each synthetic record, computing distance to nearest training neighbor
3. For each synthetic record, computing distance to nearest holdout neighbor
4. Comparing these distance distributions using statistical tests

Interpretation:

- **DCR ratio ~ 1**: Good privacy - synthetic equally distant from both
- **DCR ratio < 1**: Privacy concern - synthetic closer to training
- **DCR share ~ 0.5**: Ideal - 50% closer to training, 50% to holdout
- **DCR share > 0.5**: Privacy concern - too many closer to training

Value

An object of class "dcr" containing:

- dcr_train: distances from synthetic to closest training record
- dcr_holdout: distances from synthetic to closest holdout record
- dcr_ratio: ratio of mean distances (train/holdout), ideally ~1
- dcr_share: proportion of synthetic closer to training than holdout
- privacy_pass: logical, TRUE if dcr_share <= 0.55 and Wilcoxon p > 0.05
- wilcox_test: Wilcoxon test result comparing train vs holdout distances
- null_distribution: null distribution statistics (if null_test = TRUE)
- n_synthetic, n_train, n_holdout: dataset sizes
- method, vars: parameters used

Important limitations (The DCR Delusion)

Yao et al. (2025) demonstrate that DCR and related distance-based metrics can fail to detect privacy leakage. Key findings:

- **False sense of security**: Datasets deemed "private" by DCR can still be vulnerable to Membership Inference Attacks (MIAs).
- **Null distribution matters**: DCR values must be compared against a proper null distribution, not interpreted in absolute terms.
- **Not sufficient alone**: DCR should be used alongside other privacy metrics and, ideally, actual MIA evaluations.
- **Metric gaming**: Generators can produce low DCR while still leaking membership information through other channels.

This implementation includes statistical tests and null distribution comparison to provide more rigorous privacy assessment, but users should be aware that passing DCR tests does not guarantee privacy protection.

Holdout splitting (important)

When no external holdout is provided, DCR internally splits the original data X into training and holdout portions. This has important implications:

- **Reduced training set**: With default `holdout_fraction = 0.5`, only half of X is used for distance comparison. This may miss some disclosure risks if the synthetic data memorized records in the holdout.

- **Variability**: Results depend on the random split (use `seed` for reproducibility).
- **Best practice**: Provide a separate holdout set from the synthesis process if available. Many synthesis frameworks support train/test splits.

For small datasets (< 500 records), consider using a smaller holdout fraction (e.g., 0.2-0.3) to maintain sufficient training data for comparison.

Author(s)

Matthias Templ

References

Platzer, M. & Reutterer, T. (2021). Holdout-Based Empirical Assessment of Mixed-Type Synthetic Data. *Frontiers in Big Data*, 4, 679939. doi:10.3389/fdata.2021.679939

Park, N., et al. (2018). Data Synthesis based on Generative Adversarial Networks. *Proceedings of the VLDB Endowment*, 11(10), 1071–1083. doi:10.14778/3231751.3231757

Yao, Z., Krco, N., Ganev, G. & de Montjoye, Y.-A. (2025). The DCR Delusion: Measuring the Privacy Risk of Synthetic Data. https://arxiv.org/abs/2505.01524

Zhao, Z., et al. (2021). CTAB-GAN: Effective Table Data Synthesizing. *Asian Conference on Machine Learning*.

See Also

`nndr` for nearest neighbor distance ratio, `ims` for exact match detection, `nnaa` for nearest-neighbor adversarial accuracy

Other distance-risk: `hitting_rate()`, `ims()`, `nnaa()`, `nndr()`, `repu()`, `rf_privacy()`

Examples

```
# Create example data
set.seed(123)
X <- data.frame(
  age = rnorm(200, 40, 10),
  income = rnorm(200, 50000, 15000),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE)
)

# Good synthetic data (random, no memorization)
Y_good <- data.frame(
  age = rnorm(200, 40, 10),
  income = rnorm(200, 50000, 15000),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE)
)

# Compute DCR
result <- dcr(X, Y_good, seed = 42)
print(result)
```

```
summary(result)
```

`delta_presence` *delta-Presence Risk Assessment*

Description

Computes the delta-presence risk measure for synthetic/released data relative to the original/population data. delta-Presence (Nergiz et al., 2007) bounds the membership probability for any record in the population, measuring how confidently an adversary can determine whether a specific individual contributed to the released dataset.

Usage

```
delta_presence(X, ...)

## S3 method for class 'synth_pair'
delta_presence(X, ...)

## Default S3 method:
delta_presence(X, Y, key_vars, delta_min = 0, delta_max = 1, na.rm = TRUE, ...)
```

Arguments

`X`	data frame of original/population data, or a `synth_pair` object
`...`	additional arguments passed to methods (currently unused)
`Y`	data frame of synthetic/released data
`key_vars`	character vector of quasi-identifier variable names
`delta_min`	numeric, lower bound on acceptable membership probability (default: 0.0)
`delta_max`	numeric, upper bound on acceptable membership probability (default: 1.0)
`na.rm`	logical, remove records with NA in key variables (default: TRUE)

Details

delta-Presence is a formal privacy model that bounds the membership disclosure probability. Given a population dataset (the original data X) and a published subset (the synthetic/released data Y), delta-presence bounds the probability that any record from X appears in Y.

For each quasi-identifier combination k:

- f_k = frequency of combination k in the synthetic data Y
- F_k = frequency of combination k in the original data X
- Membership probability: $Pr(t \in Y | t \in X) = f_k / F_k$

The dataset satisfies $(\delta_{min}, \delta_{max})$-presence if for all records: $\delta_{min} \leq f_k/F_k \leq \delta_{max}$.

Membership probabilities are capped at 1.0 (when f_k > F_k, which can occur when the synthetic data overrepresents certain combinations).

Interpretation:

- Values close to 0: The QI combination is underrepresented in Y relative to X, meaning membership is unlikely
- Values close to 1: The QI combination has similar frequency in both datasets, meaning membership is highly probable
- Values = 0: The QI combination exists in X but not in Y (no membership risk for these records)

QI combinations in Y but not X: If QI combinations appear in the synthetic data Y that do not exist in the original data X, a warning is issued. These represent fabricated combinations that do not correspond to any real records.

Value

An object of class "delta_presence" containing:

- membership_prob: numeric vector of f_k/F_k per record in X
- per_combination: data.frame with QI combination, f_k, F_k, prob
- delta_min, delta_max: bounds used
- n_violations_lower: number of records below delta_min
- n_violations_upper: number of records above delta_max
- pct_violations: fraction of records violating bounds
- satisfies_delta: logical, all records within bounds
- privacy_pass: same as satisfies_delta
- n_original, n_synthetic: dataset sizes
- key_vars: quasi-identifier variables used

Author(s)

Matthias Templ

References

Nergiz, M. E., Atzori, M. & Clifton, C. (2007). Hiding the Presence of Individuals from Shared Databases. *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 665–676. doi:10.1145/1247480.1247554

See Also

`kanonymity` for k-anonymity assessment, `ldiversity` for l-diversity assessment, `tcloseness` for t-closeness assessment, `individual_risk` for probabilistic risk assessment

Other privacy-models: `attacker_risk()`, `disclosure_report()`, `domias()`, `drisk()`, `epsilon_identifiability()`, `individual_risk()`, `inspect_record()`, `kanonymity()`, `ldiversity()`, `linkability()`, `merge_per_record()`, `mia_classifier()`, `population_uniqueness()`, `recordLinkage()`, `risk_by_group()`, `singling_out()`, `suda()`, `tcloseness()`, `top_at_risk()`

Examples

```
# Create example data
set.seed(123)
n <- 200
X <- data.frame(
  age = sample(c("young", "middle", "old"), n, replace = TRUE),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Synthetic data with similar distribution
Y <- data.frame(
  age = sample(c("young", "middle", "old"), n, replace = TRUE),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

result <- delta_presence(X, Y,
                         key_vars = c("age", "gender", "region"),
                         delta_min = 0.1, delta_max = 0.9)
print(result)
summary(result)
plot(result)


# Memorized data - all probabilities near 1
Y_copy <- X
result_bad <- delta_presence(X, Y_copy,
                             key_vars = c("age", "gender", "region"),
                             delta_max = 0.5)
print(result_bad)
```

---

densitydiff_1d_num *Density ratio and entropy*

Description

Computes the density ratio between two vectors and provides entropy measures and more.

Usage

```
densitydiff_1d_num(
  x,
  y,
  bayesspace = TRUE,
  stepsize = 1000,
  strata_x = NULL,
  strata_y = NULL
)
```

Arguments

x	a numeric vector
y	a numeric vector
bayesspace	if TRUE, a compositional approach is taken
stepsize	size of sequence of points where the densities are evaluated
strata_x	vector holding the information which observation is related to which strata. If not NULL, density estimation is applied on each strata.
strata_y	vector holding the information which observation is related to which strata. If not NULL, density estimation is applied on each strata. It must have the same levels than strata_x and there must be observed values for each level.

Value

An object of class "denratio": a list with the evaluated densities, their ratio over a grid of points, and Kullback–Leibler (kl) and Jensen–Shannon (jsd) divergence measures. For stratified input, a list of such per-stratum objects. Has print and plot methods.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_kl_num(), densitydiff_pca()

Examples

```
# Simple example
set.seed(123)
x <- rnorm(200, mean = 40, sd = 10)
y <- rnorm(200, mean = 42, sd = 11)
d <- densitydiff_1d_num(x, y)
plot(d)

# With stratification
strata_x <- factor(sample(c("A", "B"), length(x), replace = TRUE))
strata_y <- factor(sample(c("A", "B"), length(y), replace = TRUE))
d2 <- densitydiff_1d_num(x, y, strata_x = strata_x, strata_y = strata_y)
```

`densitydiff_kl_num` *Kullback-Leibler divergence*

Description

Kullback-Leibler divergence between two variables or two matrices/data.frames.

Usage

```
densitydiff_kl_num(X, Y, stepsize = 1000)
```

Arguments

`X`	a numeric vector or a matrix or data frame with numeric entries.
`Y`	a numeric vector or a matrix or data frame with numeric entries.
`stepsize`	number of interval points where the density is evaluated.

Details

First the probability distributions are estimated before the Kullback-Leiber divergence is calculated. The Kullback-Leibler divergence is defined as:

$$KLD(X,Y) = \sum X \times \log\left(\frac{X}{Y}\right)$$

If the data are multivariate, a Kullback-Leibler inspired divergence is calculated through

$$KLD(X,Y) = \sum X \times \log\left(\frac{X}{Y}\right)$$

$$KLD(X,Y) = \sum_{i,j} X_{i,j}\left(\log\left(\frac{X_{i,j}}{Y_{i,j}}\right) + \log(Y) - \log(X)\right)$$

with

$$X = \sum X_{i,j}$$

and

$$Y = \sum Y_{i,j}$$

Value

The Kullback-Leibler divergence of `X` and `Y`.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_pca()

Examples

```
x <- rnorm(100)
y <- rnorm(100)

densitydiff_kl_num(x, y)

X <- MASS::mvrnorm(100, mu = c(0,0), Sigma = diag(2))
Y <- MASS::mvrnorm(100, mu = c(0,0), Sigma = diag(2))
densitydiff_kl_num(X, Y)

X <- MASS::mvrnorm(100, mu = c(0,0,0), Sigma = diag(3))
Y <- MASS::mvrnorm(100, mu = c(0,0,0), Sigma = diag(3))
densitydiff_kl_num(X, Y)
```

---

densitydiff_pca *Density ratio and entropy of first PC's*

Description

Computes the density ratio between the selected principal components and provides entropy measures and more.

Usage

```
densitydiff_pca(
  X,
  Y,
  bayesspace = TRUE,
  stepsize = 1000,
  strata_x = NULL,
  strata_y = NULL
)
```

Arguments

X	a numeric vector
Y	a numeric vector
bayesspace	if TRUE, a Bayes space compositional density approach is taken

stepsize	size of sequence of points where the densities are evaluated
strata_x	vector holding the information which observation is related to which strata. If it is not set to NULL, density estimation is applied on each strata.
strata_y	vector holding the information which observation is related to which strata. If not NULL, density estimation is applied on each strata. It must have the same levels than strata_x and there must be observed values for each level.

Value

An object of class "denpca": a list with per-principal-component Kullback–Leibler (kl) and Jensen–Shannon (jsd) divergences and the mean (mean_ratio) and standard deviation (sd_ratio) of the density ratio. For stratified input, a list of such per-stratum objects. Has print and plot methods.

Author(s)

Matthias Templ

See Also

Other comparison: ci_overlap(), compare_boxplots(), compare_chisq_gof(), compare_correlation_matrices(), compare_distributions_cont(), compare_embedding(), compare_feature_importance(), compare_histograms(), compare_ks_test(), compare_means_frequencies(), compare_missing_values(), compare_model_performance(), compare_multivariate_distribution(), compare_multivariate_summary_statistics(), compare_outliers(), compare_pca(), compare_wasserstein(), densitydiff_1d_num(), densitydiff_kl_num()

Examples

```
# Simple example with multivariate numeric data
set.seed(123)
X <- data.frame(
  var1 = rnorm(100, 50, 10),
  var2 = rnorm(100, 1000, 200),
  var3 = rnorm(100, 30, 5),
  var4 = rnorm(100, 500, 100)
)
Y <- data.frame(
  var1 = rnorm(100, 52, 11),
  var2 = rnorm(100, 980, 220),
  var3 = rnorm(100, 31, 6),
  var4 = rnorm(100, 510, 110)
)
d1 <- densitydiff_pca(X, Y, bayesspace = FALSE)
d1
```

`disclosure_report` *Comprehensive Disclosure Risk Report*

Description

Computes multiple disclosure risk metrics and produces a comprehensive report for data privacy assessment. Combines attribution-based measures (DCAP, TCAP, WEAP, DiSCO, RAPID), distance-based measures (DCR, NNDR, IMS, dRisk, hitting rate, RF Privacy), privacy models (k-anonymity, l-diversity, t-closeness), and membership inference attacks (singling out, linkability, NNAA, DOMIAS). Works for both synthetic and traditionally anonymized data.

Usage

```
disclosure_report(X, ...)

## S3 method for class 'synth_pair'
disclosure_report(X, ...)

## Default S3 method:
disclosure_report(
  X,
  Y,
  key_vars = NULL,
  target_var = NULL,
  holdout = NULL,
  holdout_fraction = 0.5,
  distance_vars = NULL,
  distance_method = c("gower", "euclidean"),
  compute = "all",
  rapid_model = "rf",
  na.rm = TRUE,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`X`	data frame of original data
`...`	additional arguments passed to methods
`Y`	data frame of synthetic/anonymized data
`key_vars`	character vector of quasi-identifier variable names for attribution-based and privacy model metrics. If NULL, these metrics are skipped.
`target_var`	character, name of the sensitive target variable for attribution-based metrics. If NULL, these metrics are skipped.

holdout	data frame of holdout data for distance-based and membership inference metrics. If NULL, automatically created from X using holdout_fraction.
holdout_fraction	numeric, fraction of X to use as holdout (default: 0.5)
distance_vars	character vector of variables for distance-based metrics. If NULL, all common variables are used.
distance_method	character, distance method for DCR/NNDR: "gower" or "euclidean" (default: "gower")
compute	character vector specifying which metrics to compute. Default is "all". Options: "all", "attribution", "distance", "privacy", "membership", or specific metric names (see Details).
rapid_model	character, model type for RAPID: "rf" (default), "lm", "cart"
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for holdout sampling (default: NULL)
verbose	logical, print progress messages (default: TRUE)

Details

This function provides a one-stop assessment of disclosure risk by computing multiple complementary metrics across four families:

Attribution-Based Metrics (require key_vars and target_var):

- dcap: Overall correct attribution probability
- tcap: Per-record attribution risk with categories
- weap: Within equivalence class attribution (on synthetic only)
- disco: Count of disclosive synthetic records
- rapid: ML-based attribute inference risk (requires ranger package)

Distance-Based Metrics (use holdout comparison):

- dcr: Distance to closest record ratio
- nndr: Nearest neighbor distance ratio
- ims: Identical match share (exact copies)
- drisk: Disclosure risk for continuous variables
- hitting_rate: Fraction of records within threshold distance
- rf_privacy: Random forest proximity-based memorization test (requires ranger)

Privacy Models (single-dataset, require key_vars):

- kanonymity: Minimum equivalence class size
- ldiversity: Sensitive attribute diversity per EC (requires target_var)
- tcloseness: EMD between EC and overall distribution (requires target_var)
- individual_risk: Frequency-based per-record risk

Membership Inference (require holdout):

- singling_out: Predicate-based uniqueness attack (GDPR criterion)
- linkability: Record linkage attack (GDPR criterion)
- nnaa: Nearest-neighbor adversarial accuracy / privacy loss
- domias: Density-based membership inference (requires ranger package)

The overall risk is determined by the number of failed checks:

- Low: All metrics pass
- Medium: 1-2 metrics fail
- High: 3+ metrics fail

Value

An object of class "disclosure_report" containing:

- results: list of individual metric results
- summary: data frame with key metrics and pass/fail status
- overall_risk: character, overall risk assessment ("LOW", "MEDIUM", "HIGH")
- n_pass: number of metrics that passed
- n_warn: number of metrics that warned/failed
- parameters: list of input parameters used

Author(s)

Matthias Templ

See Also

dcap, tcap, weap, disco, rapid, dcr, nndr, ims, drisk, hitting_rate, rf_privacy, kanonymity, ldiversity, tcloseness, individual_risk, singling_out, linkability, nnaa, domias

Other privacy-models: attacker_risk(), delta_presence(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
n <- 200
original <- data.frame(
  age_group = sample(c("18-30", "31-45", "46-60", "60+"), n, replace = TRUE),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE),
  income = sample(c("low", "medium", "high"), n, replace = TRUE)
)
```

```
# Good synthetic data
synthetic <- data.frame(
  age_group = sample(c("18-30", "31-45", "46-60", "60+"), n, replace = TRUE),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE),
  income = sample(c("low", "medium", "high"), n, replace = TRUE)
)

# Generate comprehensive report
report <- disclosure_report(
  original, synthetic,
  key_vars = c("age_group", "gender", "region"),
  target_var = "income",
  seed = 42
)

print(report)
summary(report)
plot(report)
```

---

disco *Disclosive in Synthetic Correct Original (DiSCO)*

Description

Identifies synthetic records that could potentially disclose information about original records. A DiSCO record is one that matches an original record on key variables AND has the same target value, making it potentially disclosive.

Usage

```
disco(X, ...)

## S3 method for class 'synth_pair'
disco(X, ...)

## Default S3 method:
disco(
  X,
  Y,
  key_vars,
  target_var,
  disclosure_type = c("potential", "certain"),
  method = c("exact", "gower"),
  gower_threshold = 0.1,
  na.rm = TRUE,
  ...
)
```

Arguments

X	data frame of original data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
key_vars	character vector of quasi-identifier variable names
target_var	character, name of the sensitive target variable
disclosure_type	character, type of disclosure to measure: • "potential" (default): Counts any synthetic record that matches an original record on key+target. This is the broader measure. • "certain": Only counts disclosures where the key combination uniquely determines the target in the original data (i.e., all original records with that key have the same target value). This is compatible with synthpop's DiSCO measure.
method	character, matching method: "exact" or "gower" (default: "exact")
gower_threshold	numeric, maximum Gower distance for a match when method="gower" (default: 0.1)
na.rm	logical, remove records with NA in key or target (default: TRUE)

Details

DiSCO (Disclosive in Synthetic Correct Original) is a measure developed for the synthpop package to identify attribute disclosure risk. It identifies synthetic records that:

1. Match an original record on all key variables
2. Have the same target value as that original record

Two disclosure types are available:

Potential disclosure (`disclosure_type = "potential"`): Counts any synthetic record that matches an original on key+target. This is a broader measure that identifies all records that *could* potentially leak information. Use this when you want to identify all possible disclosure risks.

Certain disclosure (`disclosure_type = "certain"`): Only counts disclosures where the key combination uniquely determines the target in the original data. This means an intruder who matches a synthetic record to the original would learn the target with certainty. This is compatible with synthpop's DiSCO measure and is more conservative.

The function always computes both metrics for comparison, but the primary output (n_disco, pct_disco, disco_idx) reflects the chosen disclosure_type.

Interpretation:

- High DiSCO count: Many synthetic records could leak original information
- DiSCO/Baseline ratio > 1: Synthetic data leaks more than expected by chance
- DiSCO = 0: No exact key+target matches (lowest risk scenario)
- pct_original_disclosed: For "certain" method, shows what fraction of original records have their target certainly exposed

Value

An object of class "disco" containing:

- disco_idx: indices of DiSCO records in Y (synthetic data)
- disco_records: the actual DiSCO records from Y
- n_disco: number of DiSCO records (based on disclosure_type)
- pct_disco: percentage of synthetic records that are DiSCO
- matched_original_idx: for each DiSCO record, which original record(s) it matches
- n_disco_potential: count using "potential" method (always computed)
- n_disco_certain: count using "certain" method (always computed)
- pct_original_disclosed: percentage of original records disclosed (for "certain" method)
- baseline_disco: expected DiSCO count under random target assignment
- key_vars, target_var, method, disclosure_type: input parameters

Author(s)

Matthias Templ

References

Raab, G.M., Nowok, B., & Dibben, C. (2021). Assessing, visualizing and improving the utility of synthetic data. *arXiv preprint arXiv:2109.12717*.

See Also

`tcap` for targeted correct attribution probability, `weap` for within equivalence class attribution probability, `dcap` for differential correct attribution probability

Other attribution-risk: `dcap()`, `tcap()`, `weap()`

Examples

```
# Create example data
set.seed(42)
X <- data.frame(
  age = sample(20:40, 50, replace = TRUE),
  gender = sample(c("M", "F"), 50, replace = TRUE),
  disease = sample(c("none", "A", "B"), 50, replace = TRUE,
                   prob = c(0.7, 0.2, 0.1))
)
# Synthetic version - some records will match by chance
Y <- data.frame(
  age = sample(20:40, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  disease = sample(c("none", "A", "B"), 100, replace = TRUE,
                   prob = c(0.7, 0.2, 0.1))
)
```

```
# Find DiSCO records using "potential" disclosure (default)
result_potential <- disco(X, Y,
                          key_vars = c("age", "gender"),
                          target_var = "disease",
                          disclosure_type = "potential")
print(result_potential)

# Find DiSCO records using "certain" disclosure (synthpop-compatible)
result_certain <- disco(X, Y,
                        key_vars = c("age", "gender"),
                        target_var = "disease",
                        disclosure_type = "certain")
print(result_certain)

# Compare both methods
cat("Potential DiSCO:", result_potential$n_disco_potential, "\n")
cat("Certain DiSCO:", result_potential$n_disco_certain, "\n")

# View the disclosive records
if (result_potential$n_disco > 0) {
  head(result_potential$disco_records)
}
```

---

domias *Density-based Membership Inference Attack (DOMIAS)*

Description

Detects local overfitting in synthetic data by estimating density ratios between the synthetic data distribution and a reference distribution at each test point. Records where the synthetic density is much higher than the reference density are likely memorized. The AUC of classifying training versus holdout records by their density ratios serves as the attack metric.

Usage

```
domias(X, ...)

## S3 method for class 'synth_pair'
domias(X, ...)

## Default S3 method:
domias(
  X,
  Y,
  holdout = NULL,
  holdout_fraction = 0.5,
  radius = 0.1,
  vars = NULL,
  na.rm = TRUE,
```

```
  seed = NULL,
  ...
)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
holdout	data frame of holdout data (optional). If NULL, a holdout set is automatically created by splitting X.
holdout_fraction	numeric, fraction of X to use as holdout if holdout is NULL (default: 0.5)
radius	numeric, Gower distance radius for neighbor counting (default: 0.1). Smaller values increase sensitivity but require more data.
vars	character vector of variable names to use. If NULL (default), all common variables between X and Y are used.
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for holdout sampling (default: NULL)

Details

DOMIAS (Density-based Overfitting Membership Inference Attack on Synthetic data) detects whether a synthetic data generator has memorized specific training records. The key insight is that if a generator overfits to training data, the synthetic distribution will have higher density around training points compared to unseen holdout points.

The algorithm works by:

1. Splitting original data into training (used for synthesis) and holdout
2. For each test record, estimating synthetic density by counting neighbors within radius `r` in the synthetic data
3. Estimating reference density by counting neighbors within radius `r` in the full original data (training + holdout)
4. Computing the density ratio: (density_synth + 1) / (density_ref + 1), with Laplace smoothing to avoid division by zero
5. Comparing density ratios for training versus holdout records

Interpretation:

- **AUC ~ 0.5**: Good privacy - density ratios cannot distinguish training from holdout (no memorization detected)
- **AUC > 0.5**: Privacy concern - training records have higher density ratios, suggesting local overfitting
- **AUC > 0.6**: Likely memorization detected
- **memorization_score ~ 1**: No differential memorization

- **memorization_score » 1**: Synthetic data concentrated around training records

This implementation uses Gower distance for neighbor counting, which handles mixed data types (numeric, categorical, ordinal). For purely numeric data, the radius parameter corresponds to the Gower distance (normalized between 0 and 1).

Value

An object of class "domias" containing:

- density_ratios_train: numeric vector of density ratios for training records
- density_ratios_holdout: numeric vector of density ratios for holdout records
- auc: AUC of train-vs-holdout classification by density ratio
- mean_ratio_train: mean density ratio for training records
- mean_ratio_holdout: mean density ratio for holdout records
- memorization_score: mean_ratio_train / mean_ratio_holdout (ideally ~1)
- privacy_pass: logical, TRUE if auc <= 0.6
- n_train, n_holdout, n_synthetic: dataset sizes
- radius: radius used for density estimation
- vars: variables used

Holdout splitting

When no external holdout is provided, the original data is split internally. The holdout serves as a control group: if the generator has not memorized training data, density ratios should be similar for training and holdout records. For best results, provide a separate holdout set from the synthesis process.

Choosing the radius

The `radius` parameter controls the bandwidth of the kernel density estimate. Guidelines:

- Smaller radius (0.01-0.05): More sensitive to local memorization but noisier estimates, needs more data
- Default radius (0.1): Good balance for most datasets
- Larger radius (0.2-0.5): Smoother estimates but may miss localized overfitting

Author(s)

Matthias Templ

References

Van Breugel, B., Sun, H., Qian, Z. & van der Schaar, M. (2023). Membership Inference Attacks against Synthetic Data through Overfitting Detection. *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, PMLR 206, 3493–3514. https://proceedings.mlr.press/v206/breugel23a.html

See Also

dcr for distance to closest record, nndr for nearest neighbor distance ratio, nnaa for nearest-neighbor adversarial accuracy

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
n <- 200
X <- data.frame(
  age = rnorm(n, 40, 10),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Good synthetic data (random, no memorization)
Y_good <- data.frame(
  age = rnorm(n, 40, 10),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

result <- domias(X, Y_good, seed = 42)
print(result)
summary(result)


# Memorized data (Y is copy of X) - should show high AUC
Y_copy <- X[sample(nrow(X), n, replace = TRUE), ]
result_bad <- domias(X, Y_copy, seed = 42)
print(result_bad)
```

---

drisk *Disclosure Risk for Continuous Variables (dRisk / dRiskRMD)*

Description

Computes disclosure risk measures for continuous key variables, based on interval overlap (dRisk) and Robust Mahalanobis Distance (dRiskRMD). These methods assess whether original records can be re-identified from synthetic data using numeric quasi-identifiers.

Usage

```
drisk(X, ...)

## S3 method for class 'synth_pair'
drisk(X, ...)

## Default S3 method:
drisk(
  X,
  Y,
  vars = NULL,
  method = c("both", "interval", "rmd"),
  outlier_par = 0.01,
  alpha = 0.05,
  na.rm = TRUE,
  ...
)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
vars	character vector of numeric variable names to use. If NULL (default), all common numeric variables between X and Y are used.
method	character, which risk measure(s) to compute: "interval" (dRisk), "rmd" (dRiskRMD), or "both" (default).
outlier_par	numeric, outlier parameter controlling interval width for the interval method (default: 0.01). Larger values produce wider intervals and higher risk estimates.
alpha	numeric, significance level for the chi-squared threshold in the RMD method (default: 0.05). Smaller alpha means a stricter threshold (fewer flagged records).
na.rm	logical, remove records with NA values (default: TRUE)

Details

Both methods work ONLY on numeric variables. Non-numeric columns are automatically excluded (or cause an error if explicitly requested via vars).

Interval method (dRisk): For each variable v, an interval half-width is computed as:

$$d_v = \text{outlier\_par} \times \text{IQR}(X_v)/(2n)^{1/3}$$

where n = nrow(X). For each original record i, the record is flagged as at-risk if there exists at least one synthetic record j where ALL variables of record i fall within the intervals around record j:

$$|x_{iv} - y_{jv}| \leq d_v \quad \forall v$$

The dRisk score is the fraction of original records that are at-risk.

RMD method (dRiskRMD): Uses the Minimum Covariance Determinant (MCD) estimator from cov.rob to obtain a robust covariance matrix from the original data. For each original record, the minimum Robust Mahalanobis Distance to any synthetic record is computed. Records with minimum RMD below the chi-squared threshold $\chi^2_{1-\alpha,p}$ (where p = number of variables) are flagged as at-risk.

Value

An object of class "drisk" containing:

- drisk_interval: fraction of at-risk records (interval method), NA if not computed
- drisk_rmd: fraction of at-risk records (RMD method), NA if not computed
- at_risk_interval: logical vector per original record (interval method), NULL if not computed
- at_risk_rmd: logical vector per original record (RMD method), NULL if not computed
- min_rmd: numeric vector of minimum Robust Mahalanobis Distance per original record, NULL if method does not include "rmd"
- interval_widths: named numeric vector of interval half-widths d_v per variable, NULL if method does not include "interval"
- method: method(s) used
- privacy_pass: logical, max(drisk_interval, drisk_rmd) <= 0.1
- n_original: number of original records used
- n_synthetic: number of synthetic records used
- vars: variables used
- outlier_par: outlier parameter used
- alpha: significance level used

Interpretation

- **dRisk close to 0**: Low disclosure risk - few original records can be matched by synthetic data within the tolerance intervals.
- **dRisk close to 1**: High disclosure risk - most original records have close matches in the synthetic data.
- **dRiskRMD close to 0**: Low risk - original records are far from synthetic records in robust Mahalanobis distance.
- **dRiskRMD close to 1**: High risk - many original records have suspiciously close synthetic counterparts.

Author(s)

Matthias Templ

References

Templ, M. (2017). *Statistical Disclosure Control for Microdata: Methods and Applications in R*. Springer. doi:10.1007/9783319502724

Templ, M., Kowarik, A. & Meindl, B. (2024). sdcMicro: Statistical Disclosure Control Methods for Anonymization of Data and Risk Estimation. R package. https://CRAN.R-project.org/package=sdcMicro

Rousseeuw, P. J. & Van Driessen, K. (1999). A fast algorithm for the Minimum Covariance Determinant estimator. *Technometrics*, 41(3), 212–223. doi:10.1080/00401706.1999.10485670

See Also

dcr for distance-based privacy with holdout comparison, nndr for nearest neighbor distance ratio, ims for identical match detection

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
n <- 100
X <- data.frame(
  age = rnorm(n, 40, 10),
  income = rnorm(n, 50000, 15000),
  hours = rnorm(n, 40, 8)
)

# Good synthetic data (independent generation)
Y <- data.frame(
  age = rnorm(n, 40, 10),
  income = rnorm(n, 50000, 15000),
  hours = rnorm(n, 40, 8)
)

result <- drisk(X, Y)
print(result)
summary(result)


# Memorized data (Y is copy of X) - should show high risk
Y_copy <- X + rnorm(n * 3, sd = 0.01)
result_bad <- drisk(X, Y_copy)
print(result_bad)
```

`energy_distance` *Energy Distance for Multivariate Numeric Data*

Description

Computes the energy distance between multivariate numeric distributions in two datasets. Energy distance is a statistical distance that characterizes equality of distributions and is zero if and only if the distributions are identical.

Usage

```
energy_distance(X, ...)

## S3 method for class 'synth_pair'
energy_distance(X, ...)

## Default S3 method:
energy_distance(
  X,
  Y,
  vars = NULL,
  standardize = TRUE,
  n_sample = 1000,
  na.rm = TRUE,
  seed = NULL,
  ...
)
```

Arguments

`X`	A data.frame or data.table containing the original dataset.
`...`	additional arguments passed to methods (currently unused)
`Y`	A data.frame or data.table containing the synthetic/anonymized dataset.
`vars`	Character vector of numeric variable names to compare. If NULL (default), all common numeric variables are used.
`standardize`	Logical, whether to standardize variables before computing distances. Default TRUE (recommended for variables on different scales).
`n_sample`	Integer, maximum sample size for computation. If datasets are larger, random sampling is used to reduce computation time. Default 1000. Set to NULL for no sampling (may be slow for large datasets).
`na.rm`	Logical, whether to remove rows with NA values. Default TRUE.
`seed`	Integer, random seed for reproducible sampling. Default NULL.

Details

The energy distance between distributions F and G is defined as:

$$D_E(F,G) = 2E||X-Y|| - E||X-X'|| - E||Y-Y'||$$

where X, X' are independent samples from F, and Y, Y' are independent samples from G. The energy distance:

- Is always non-negative
- Equals zero if and only if F = G
- Is sensitive to differences in both location and scale
- Does not require density estimation

For synthetic data evaluation, lower energy distance indicates better preservation of the multivariate numeric distribution.

The computation uses Euclidean distances. For large datasets, random sampling is applied to keep computation tractable (O(n^2) complexity).

Value

An object of class "energy_distance" containing:

- energy_distance: the computed energy distance
- energy_distance_normalized: energy distance divided by a reference value
- mean_dist_XY: mean distance between X and Y samples
- mean_dist_XX: mean distance within X samples
- mean_dist_YY: mean distance within Y samples
- n_X, n_Y: sample sizes used in computation
- n_vars: number of variables
- vars: variable names used
- standardized: whether standardization was applied
- utility_score: transformed score (higher = better utility)

Author(s)

Matthias Templ

References

Szekely, G. J. and Rizzo, M. L. (2013). Energy statistics: A class of statistics based on distances. Journal of Statistical Planning and Inference, 143(8), 1249-1272.

Rizzo, M. L. and Szekely, G. J. (2016). Energy Distance. WIREs Computational Statistics, 8(1), 27-38.

See Also

hellinger for categorical distributions, compare_wasserstein for univariate Wasserstein distance, gower for mixed-type data

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), gower(), hellinger(), mmd(), mqs(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), rumap(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
set.seed(123)
# Original data
X <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  age = rnorm(500, mean = 40, sd = 10),
  score = rnorm(500, mean = 100, sd = 15)
)

# Good synthetic data (similar distribution)
Y_good <- data.frame(
  income = rnorm(500, mean = 50000, sd = 10000),
  age = rnorm(500, mean = 40, sd = 10),
  score = rnorm(500, mean = 100, sd = 15)
)

# Poor synthetic data (shifted distribution)
Y_poor <- data.frame(
  income = rnorm(500, mean = 60000, sd = 15000),
  age = rnorm(500, mean = 45, sd = 15),
  score = rnorm(500, mean = 90, sd = 20)
)

result_good <- energy_distance(X, Y_good, seed = 42)
print(result_good)

result_poor <- energy_distance(X, Y_poor, seed = 42)
print(result_poor)
```

Entropy *Entropy measures*

Description

Kullback-Leibler and Jensen-Shannon divergence

Usage

```
KLDiv(A, B)

KLDiv_bayes(A, B)
```

```
JSDiv(A, B)

JSDiv_bayes(A, B)

CrossEntropy(A, B)
```

Arguments

A	a vector of probability densities
B	a vector of probability densities

Details

The Kullback-Leibler divergence is defined as:

$$KLD(X,Y) = \sum X \times \log\left(\frac{X}{Y}\right)$$

If the data are multivariate, a Kullback-Leibler inspired divergence is calculated through

$$KLD(X,Y) = \sum_{i,j} X_{i,j}\left(\log\left(\frac{X_{i,j}}{Y_{i,j}}\right) + \log(Y) - \log(X)\right)$$

with

$$X = \sum X_{i,j}$$

and

$$Y = \sum Y_{i,j}$$

The compositional versions of the Kullback-Leibler divergence is given in the reference below.

Value

The divergence measure

Author(s)

Matthias Templ

References

J.A. Martin-Fernandez, M. Bren, C. Barcelo-Vidal, V. Pawlowski-Glahn (1999). A measure of difference for compositional data based on measures of divergence. In S. Lippard, A. Nass, and R. Sinding-Larsen (Eds.), Proceedings of IAMG'99, Volume 1, Trondheim (Norway), pp. 211-215.

See Also

Other information-theory: `EntropyMeasures`, `information_surprisal()`, `max_info_leakage()`, `mutualInformation()`, `positive_information_disclosure()`, `privacy_score()`, `systemAnonymityLevel()`

Examples

```
# Kullback-Leibler divergence between two probability vectors
p <- c(0.4, 0.3, 0.2, 0.1)
q <- c(0.25, 0.25, 0.25, 0.25)
KLDiv(p, q)

# Bayesian KL divergence
KLDiv_bayes(p, q)

# Jensen-Shannon divergence (symmetric)
JSDiv(p, q)

# Bayesian Jensen-Shannon divergence
JSDiv_bayes(p, q)

# Cross-entropy
CrossEntropy(p, q)
```

EntropyMeasures *Additional Entropy-Based Privacy Measures*

Description

This set of functions implements various entropy-based measures used in privacy and information theory. All functions assume that the input vectors p or matrices are valid probability distributions.

Usage

```
RenyiEntropy(p, alpha = 2)

MaxEntropy(p)

MinEntropy(p)

NormalizedEntropy(p)

ConditionalEntropy(joint)

CumulativeEntropy(x)
```

Arguments

p	A vector of probability values (must sum to 1 for entropy-based methods).
alpha	The order of Renyi entropy (must be > 0 and not equal to 1).
joint	A joint probability matrix (rows = X, columns = Y) for ConditionalEntropy.
x	A numeric vector for CumulativeEntropy.

Details

`RenyiEntropy(p, alpha = 2)` Computes Rényi entropy of order `alpha` for a probability vector `p`.

`MaxEntropy(p)` Returns the Hartley (max) entropy, defined as the logarithm of the number of non-zero categories.

`MinEntropy(p)` Computes the min-entropy, defined as the negative logarithm of the maximum probability.

`NormalizedEntropy(p)` Returns the Shannon entropy normalized by the maximum entropy, resulting in a value between 0 and 1.

`ConditionalEntropy(joint)` Computes the conditional entropy H(Y|X), given a joint probability matrix (rows = X, columns = Y).

`CumulativeEntropy(x)` Estimates cumulative entropy based on the empirical cumulative distribution function of a numeric vector `x`.

Value

A numeric value representing the corresponding entropy or risk measure.

Author(s)

Matthias Templ

References

Rényi, A. (1961). On measures of entropy and information. In Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability (Vol. 1, pp. 547–561).

Shannon, C.E. (1948). A Mathematical Theory of Communication. Bell System Technical Journal, 27(3), 379–423.

Bender, S., Haas, A., & Klose, C. (2001). The IAB Employment Sample. Journal of Applied Social Science Studies, 121(2), 183–190.

Ichim, D. (2009). Local neighbourhood-based record linkage and its application to the Canadian census. In Privacy in Statistical Databases (pp. 207–221). Springer.

See Also

Other information-theory: `Entropy`, `information_surprisal()`, `max_info_leakage()`, `mutualInformation()`, `positive_information_disclosure()`, `privacy_score()`, `systemAnonymityLevel()`

Examples

```
# Renyi entropy of order 2
p <- c(0.4, 0.3, 0.2, 0.1)
RenyiEntropy(p, alpha = 2)

# Maximum (Hartley) entropy
MaxEntropy(p)

# Min-entropy
```

```
MinEntropy(p)

# Normalized Shannon entropy (0 to 1)
NormalizedEntropy(p)

# Conditional entropy H(Y|X) from a joint probability matrix
joint <- matrix(c(0.1, 0.2, 0.3, 0.4), nrow = 2)
ConditionalEntropy(joint)

# Cumulative entropy of a numeric vector
x <- c(1.2, 3.4, 2.1, 5.0, 4.3)
CumulativeEntropy(x)
```

`epsilon_identifiability` *Epsilon Identifiability*

Description

Computes the Epsilon Identifiability risk metric for synthetic data, a distance-entropy hybrid from the SynthEval framework. For each synthetic record, the metric finds its weighted Gower distance to the closest original record, where variable weights are the inverse of Shannon entropy (rare attributes are penalized more heavily). Records with minimum distance below the epsilon threshold are flagged as identifiable.

Usage

```
epsilon_identifiability(X, ...)

## S3 method for class 'synth_pair'
epsilon_identifiability(X, ...)

## Default S3 method:
epsilon_identifiability(
  X,
  Y,
  epsilon = 0.05,
  vars = NULL,
  na.rm = TRUE,
  seed = NULL,
  n_bins = 20L,
  ...
)
```

Arguments

`X`	data frame of original/training data

...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
epsilon	numeric, distance threshold below which a synthetic record is considered identifiable (default: 0.05). Must be between 0 and 1.
vars	character vector of variable names to use. If NULL (default), all common variables between X and Y are used.
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for reproducibility (default: NULL). Currently unused but reserved for future extensions.
n_bins	integer, number of bins for discretizing numeric variables before computing entropy (default: 20)

Details

The Epsilon Identifiability metric combines distance-based and information-theoretic approaches:

1. Compute Shannon entropy H_v for each variable v in the original data. For numeric variables, values are discretized into `n_bins` equal-width bins. For categorical variables, entropy is computed directly from category frequencies.
2. Compute inverse-entropy weights $w_v = 1/H_v$. Variables with zero entropy (constant) receive zero weight. Weights are normalized to sum to 1.
3. Compute weighted Gower distance from each synthetic record to all original records using `VIM::gowerD` with the entropy-based weights.
4. For each synthetic record, find the minimum weighted distance.
5. Flag records where distance < epsilon.
6. Compute identifiability rate = fraction of flagged records.

The intuition is that variables with low entropy (few distinct values or skewed distributions) carry more identifying power. A close match on a rare-valued variable is more concerning than a close match on a high-entropy variable.

Note: This is the SynthEval distance-entropy variant (Lautrup et al., 2025). It shares the name but not the definition of the probabilistic ϵ-identifiability of Yoon et al. (2020, ADS-GAN); the two should not be conflated.

Value

An object of class "epsilon_identifiability" containing:

- distances: numeric vector of minimum weighted distances per synthetic record
- flagged: logical vector, TRUE where distance < epsilon
- identifiability_rate: fraction of synthetic records flagged
- entropy_weights: named numeric vector of per-variable weights (normalized, sum to 1)
- entropies: named numeric vector of per-variable Shannon entropies
- epsilon: threshold used

- n_flagged: count of flagged records
- privacy_pass: logical, TRUE if identifiability_rate <= 0.1
- n_original: number of original records
- n_synthetic: number of synthetic records
- vars: variables used

Choosing epsilon

The default `epsilon = 0.05` is a conservative threshold. Smaller values are more strict (fewer flagged records). The appropriate threshold depends on the number of variables and the data domain. Use `plot(result, which = 1)` to visualize the distance distribution and assess whether the threshold is appropriate.

Author(s)

Matthias Templ

References

Lautrup, A., Hyrup, T., Zimek, A. & Blockeel, H. (2025). SynthEval: A Framework for Detailed Utility and Privacy Evaluation of Tabular Synthetic Data. *Data Mining and Knowledge Discovery*, 39(1). doi:10.1007/s10618024010814

See Also

`dcr` for distance to closest record, `nndr` for nearest neighbor distance ratio, `ims` for identical match share

Other privacy-models: `attacker_risk()`, `delta_presence()`, `disclosure_report()`, `domias()`, `drisk()`, `individual_risk()`, `inspect_record()`, `kanonymity()`, `ldiversity()`, `linkability()`, `merge_per_record()`, `mia_classifier()`, `population_uniqueness()`, `recordLinkage()`, `risk_by_group()`, `singling_out()`, `suda()`, `tcloseness()`, `top_at_risk()`

Examples

```
# Create example data
set.seed(123)
n <- 200
X <- data.frame(
  age = sample(20:70, n, replace = TRUE),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Good synthetic data (random, no memorization)
Y_good <- data.frame(
  age = sample(20:70, n, replace = TRUE),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
```

```
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

result <- epsilon_identifiability(X, Y_good)
print(result)
summary(result)


# Memorized data (Y is a copy of X) - should show high risk
Y_copy <- X[sample(nrow(X), n, replace = TRUE), ]
result_bad <- epsilon_identifiability(X, Y_copy)
print(result_bad)
```

`evaluation_stats` *Evaluation statistics*

Description

Several kinds of evaluation statistics (MAPE, MAE, MSE, RMSE, AIT)

Usage

```
mape(x, y)

mae(x, y)

mse(x, y)

rmse(x, y)

ait(x, y)
```

Arguments

x	numeric vector
y	numeric vector

Value

The MAPE, MAE, MSE, RMSE, and (normalized) AIT (Aitchison distance) between two vectors.

Author(s)

Matthias Templ

See Also

rapid

Examples

```
x <- rnorm(10)
y <- rnorm(10)
mape(x, y)
mae(x, y)
mse(x, y)
rmse(x, y)
ait(x, y)
```

`from_sdcMicro` *Extract comparison data from sdcMicro object*

Description

Creates a `synth_pair` object from an sdcMicro `sdcMicroObj`.

Usage

```
from_sdcMicro(x, target_var = NULL, use_weights = TRUE, ...)
```

Arguments

`x`	An sdcMicro object of class "sdcMicroObj"
`target_var`	Character string naming the sensitive target variable (optional). If NULL and sensible variables are defined in the sdcMicro object, the first sensible variable is used.
`use_weights`	Logical, whether to extract and use sampling weights from the sdcMicro object (default: TRUE)
`...`	Additional arguments passed to `synth_pair`

Details

The sdcMicro package provides statistical disclosure control methods for microdata (local suppression, recoding, microaggregation, PRAM, noise addition, etc.). The `sdcMicroObj` S4 class stores both the original and anonymized data. This function extracts both into a `synth_pair` for evaluation with riskutility measures.

Note that sdcMicro produces *anonymized* data (perturbation of real records), not truly synthetic data. Nonetheless, many risk and utility measures apply equally. In particular, attribution-based measures (CAP family), distance-based measures, and all utility measures can be used to compare original vs. anonymized microdata.

Variable roles are automatically extracted from the sdcMicro object:

- **key_vars**: Categorical quasi-identifiers (`keyVars` slot)
- **target_var**: Sensitive variable (`sensibleVar` slot, first entry)
- **weight_original**: Sampling weight (`weightVar` slot)

- **num_vars**: Numeric key variables (`numVars` slot)

The anonymized data is reconstructed via `sdcMicro::extractManipData()`, which combines manipulated key, numeric, and PRAM variables with unmodified columns from the original data.

Value

An object of class "synth_pair"

See Also

`synth_pair`, `from_synthpop`, `from_simPop`

Examples

```
if (requireNamespace("sdcMicro", quietly = TRUE)) {
  # Create a simple sdcMicro object
  data("testdata2", package = "sdcMicro")
  sdc <- sdcMicro::createSdcObj(
    dat = testdata2,
    keyVars = c("urbrur", "roof", "walls", "water", "sex"),
    numVars = c("expend", "income", "savings"),
    weightVar = "sampling_weight"
  )
  # Apply some anonymization
  sdc <- sdcMicro::localSuppression(sdc)
  # Create synth_pair
  pair <- from_sdcMicro(sdc)
  print(pair)
}
```

---

from_simPop *Extract comparison data from simPop object*

Description

Creates a `synth_pair` object from a simPop `simPopObj`.

Usage

```
from_simPop(
  x,
  key_vars = NULL,
  target_var = NULL,
  use_sample_weights = TRUE,
  ...
)
```

Arguments

x	A simPop object of class "simPopObj"
key_vars	Character vector of quasi-identifier variables (optional)
target_var	Character string naming the sensitive target variable (optional)
use_sample_weights	Logical, whether to extract and use sample weights from the simPop object (default: TRUE)
...	Additional arguments passed to synth_pair

Details

The simPop package generates synthetic populations from survey samples. Unlike synthpop which creates synthetic samples, simPop creates a full synthetic population. This has implications for interpretation:

- **Original**: The survey sample (typically has sampling weights)
- **Synthetic**: The generated population (typically no weights, or equal weights)

The simPop object stores both the original sample and the synthetic population, so unlike from_synthpop, you don't need to provide the original data.

Value

An object of class "synth_pair"

See Also

synth_pair, from_synthpop

Examples

```
if (requireNamespace("simPop", quietly = TRUE)) {
  # simPop example would go here
  # Note: simPop has complex dependencies, example kept minimal
}
```

---

from_synthpop *Extract comparison data from synthpop object*

Description

Creates a synth_pair object from a synthpop synds or synds.list object.

Usage

```
from_synthpop(x, original, key_vars = NULL, target_var = NULL, m = 1, ...)
```

Arguments

x	A synthpop object of class "synds" or "synds.list"
original	The original data used to create the synthetic data. This is required as synthpop does not store the original data.
key_vars	Character vector of quasi-identifier variables (optional)
target_var	Character string naming the sensitive target variable (optional)
m	For synds.list objects with multiple syntheses, which one to use (default: 1)
...	Additional arguments passed to synth_pair

Details

The synthpop package creates synthetic data using sequential modeling. It returns objects of class synds (single synthesis) or synds.list (multiple syntheses). This function extracts the synthetic data and combines it with the original data into a synth_pair for analysis.

Note: synthpop does not store the original data in the output object, so you must provide it explicitly.

Value

An object of class "synth_pair"

See Also

synth_pair, from_simPop

Examples

```
if (requireNamespace("synthpop", quietly = TRUE)) {
  library(synthpop)

  # Create some example data
  original <- data.frame(
    age = sample(20:60, 100, replace = TRUE),
    sex = factor(sample(c("M", "F"), 100, replace = TRUE)),
    income = rnorm(100, 50000, 10000)
  )

  # Generate synthetic data
  syn_obj <- syn(original, seed = 123)

  # Create synth_pair
  pair <- from_synthpop(syn_obj, original,
                        key_vars = c("age", "sex"),
                        target_var = "income")
  print(pair)
}
```

gower *Gower distance between two data frames*

Description

Mean per-observation Gower distance between an original data frame and a *row-corresponding* anonymized version (e.g. a perturbative method such as noise addition, microaggregation, or rank swapping, where record i of the anonymized data corresponds to record i of the original). Returns an S3 object of class `"gower"` with print, summary, and plot methods.

Usage

```
gower(X, ...)

## S3 method for class 'synth_pair'
gower(X, ...)

## Default S3 method:
gower(X, Y, ...)
```

Arguments

`X`	data frame (or a `synth_pair` object)
`...`	additional arguments passed to methods
`Y`	data frame

Details

The two data frames must have the same number of rows; rows are compared pairwise, so the reported `gower_distance` is $\frac{1}{n}\sum_{i=1}^{n} d_G(X_i, Y_i)$, the mean Gower distance between corresponding records, in $[0, 1]$. Identical data therefore yields distance `0` and `utility_score` `1`. This measure assumes record correspondence (perturbative anonymization); for fully synthetic data without such correspondence use a distributional measure such as `energy_distance`, `mmd`, or `compare_wasserstein`.

Value

An object of class `"gower"` containing:

- `gower_distance`: the average Gower distance (lower = more similar)
- `utility_score`: `1 - gower_distance`, in $[0, 1]$ (higher = better utility)
- `n_records`: number of observations
- `n_variables`: number of variables
- `n`: alias for `n_records` (for backward compatibility)
- `n_vars`: alias for `n_variables` (for backward compatibility)

Author(s)

Matthias Templ. Based on the gowerD function from Alexander Kowarik in the VIM package.

References

Gower, J.C. (1971). A General Coefficient of Similarity and Some of Its Properties. *Biometrics*, 27(4), 857–871.

See Also

dcr, ims

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), hellinger(), mmd(), mqs(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), rumap(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
# Simple example with mixed data types
X <- data.frame(
  age = c(25, 30, 35, 40),
  income = c(30000, 45000, 50000, 60000),
  gender = factor(c("M", "F", "M", "F"))
)
Y <- data.frame(
  age = c(26, 31, 34, 42),
  income = c(32000, 44000, 52000, 58000),
  gender = factor(c("M", "F", "M", "F"))
)
result <- gower(X, Y)
result
summary(result)

plot(result)
```

---

hellinger *Hellinger Distance for Categorical Distributions*

Description

Computes the Hellinger distance between categorical variable distributions in two datasets. The Hellinger distance measures the similarity between probability distributions and is bounded between 0 (identical) and 1 (no overlap).

Usage

```
hellinger(X, ...)

## S3 method for class 'synth_pair'
hellinger(X, ...)

## Default S3 method:
hellinger(
  X,
  Y,
  vars = NULL,
  weight_X = NULL,
  weight_Y = NULL,
  na.rm = TRUE,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods (currently unused)
Y	A data.frame or data.table containing the synthetic/anonymized dataset.
vars	Character vector of categorical variable names to compare. If NULL (default), all common factor/character variables are used.
weight_X	Optional character string specifying sampling weight variable in X.
weight_Y	Optional character string specifying sampling weight variable in Y.
na.rm	Logical, whether to remove NA values before computation. Default TRUE.

Details

The Hellinger distance between two discrete probability distributions P and Q is:

$$H(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_i (\sqrt{p_i} - \sqrt{q_i})^2}$$

Properties:

- **H = 0**: Distributions are identical
- **H = 1**: Distributions have no overlap (disjoint support)
- Symmetric: H(P,Q) = H(Q,P)
- Bounded: 0 <= H <= 1

For utility assessment, lower Hellinger distance indicates better preservation of categorical distributions in synthetic data.

Value

An object of class "hellinger" containing:

- per_variable: data.frame with Hellinger distance for each variable
- hellinger_mean: mean Hellinger distance across all variables
- hellinger_max: maximum Hellinger distance across variables
- n_vars: number of variables compared
- vars: variable names used
- utility_score: 1 - hellinger_mean (higher = better utility)

Author(s)

Matthias Templ

References

Hellinger, E. (1909). Neue Begruendung der Theorie quadratischer Formen von unendlichvielen Veraenderlichen. Journal fuer die reine und angewandte Mathematik, 136, 210-271.

See Also

compare_chisq_gof for chi-squared goodness of fit, energy_distance for multivariate numeric comparison

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), gower(), mmd(), mqs(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), rumap(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
set.seed(123)
# Original data
X <- data.frame(
  gender = sample(c("Male", "Female"), 500, replace = TRUE),
  region = sample(c("North", "South", "East", "West"), 500, replace = TRUE),
  education = sample(c("High", "Medium", "Low"), 500, replace = TRUE)
)

# Good synthetic data (similar distributions)
Y_good <- data.frame(
  gender = sample(c("Male", "Female"), 500, replace = TRUE),
  region = sample(c("North", "South", "East", "West"), 500, replace = TRUE),
  education = sample(c("High", "Medium", "Low"), 500, replace = TRUE)
)

# Poor synthetic data (different distributions)
Y_poor <- data.frame(
  gender = sample(c("Male", "Female"), 500, replace = TRUE, prob = c(0.9, 0.1)),
  region = sample(c("North", "South", "East", "West"), 500, replace = TRUE,
                  prob = c(0.7, 0.1, 0.1, 0.1)),
```

```
  education = sample(c("High", "Medium", "Low"), 500, replace = TRUE,
                     prob = c(0.1, 0.1, 0.8))
)

result_good <- hellinger(X, Y_good)
print(result_good)

result_poor <- hellinger(X, Y_poor)
print(result_poor)
```

`high_risk_records` *Get high-risk records*

Description

Extract records with risk above a specified threshold.

Usage

```
high_risk_records(x, threshold = NULL)
```

Arguments

`x`	an object of class "individual_risk"
`threshold`	numeric, risk threshold (default: uses threshold from object)

Value

data frame with high-risk record indices and their risk values

---

`hitting_rate` *Hitting Rate*

Description

Computes the Hitting Rate privacy metric for synthetic data. The Hitting Rate measures the fraction of synthetic records that fall within a configurable distance threshold of any original record. It bridges the Identical Match Share (IMS, threshold = 0) and the full Distance to Closest Record (DCR) distribution into a single, interpretable scalar.

Usage

```
hitting_rate(X, ...)

## S3 method for class 'synth_pair'
hitting_rate(X, ...)

## Default S3 method:
hitting_rate(
  X,
  Y,
  threshold = 0.05,
  vars = NULL,
  method = c("gower", "euclidean"),
  na.rm = TRUE,
  ...
)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
threshold	numeric, distance threshold for a "hit" (default: 0.05). A synthetic record with minimum distance to any original record at or below this threshold is counted as a hit.
vars	character vector of variable names to use for distance calculation. If NULL (default), all common variables between X and Y are used.
method	character, distance method: "gower" (default, handles mixed types) or "euclidean" (numerical variables only)
na.rm	logical, remove records with NA values (default: TRUE)

Details

The Hitting Rate provides a threshold-based view of synthetic data privacy. For each synthetic record, the minimum distance to any original record is computed. Records with a minimum distance at or below the threshold are counted as "hits" – near copies that may pose a disclosure risk.

The metric is defined as:

$$HR(\tau) = \frac{|\{y \in Y : \min_{x \in X} d(x, y) \leq \tau\}|}{|Y|}$$

The hitting rate generalises two existing metrics:

- At threshold = 0, the hitting rate equals the Identical Match Share (ims).
- The full distribution of minimum distances is the same as the DCR training distribution in dcr, but without requiring a holdout set.

Interpretation:

- **rate ~ 0**: Good privacy – few synthetic records are close to any original record.
- **rate > 0.1**: Privacy concern – too many near copies.
- **rate ~ 1**: Severe memorization – nearly all synthetic records are close copies of original records.

The default threshold of 0.05 is appropriate for Gower distances (which are bounded between 0 and 1). For Euclidean distances on normalized data, adjust the threshold according to the dimensionality and scale of the data.

Value

An object of class "hitting_rate" containing:

- rate: fraction of synthetic records within threshold distance of any original record
- min_distances: numeric vector of minimum distances per synthetic record
- hits: logical vector indicating which synthetic records are hits (min_distance <= threshold)
- n_hits: count of hits
- threshold: the threshold used
- privacy_pass: logical, TRUE if rate <= 0.1
- n_original: number of original records
- n_synthetic: number of synthetic records
- method: distance method used
- vars: variables used
- rate_at_zero: fraction of synthetic records with exact matches (min_distance == 0), equivalent to IMS

Author(s)

Matthias Templ

References

Platzer, M. & Reutterer, T. (2021). Holdout-Based Empirical Assessment of Mixed-Type Synthetic Data. *Frontiers in Big Data*, 4, 679939. doi:10.3389/fdata.2021.679939

See Also

`dcr` for distance to closest record (with holdout comparison), `ims` for exact match detection (equivalent to hitting rate at threshold 0), `nndr` for nearest neighbor distance ratio

Other distance-risk: `dcr()`, `ims()`, `nnaa()`, `nndr()`, `repu()`, `rf_privacy()`

Examples

```
# Create example data
set.seed(123)
n <- 200
X <- data.frame(
  age = rnorm(n, 40, 10),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Good synthetic data (random, no memorization)
Y_good <- data.frame(
  age = rnorm(n, 40, 10),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

result <- hitting_rate(X, Y_good)
print(result)
summary(result)


# Memorized data (Y is copy of X) -- should show high rate
Y_bad <- X[sample(nrow(X), n, replace = TRUE), ]
result_bad <- hitting_rate(X, Y_bad)
print(result_bad)

# Sweep thresholds visually
plot(result, which = 2)
```

---

ims *Identical Match Share (IMS)*

Description

Computes the Identical Match Share privacy metric for synthetic data. IMS measures the proportion of synthetic records that are exact copies of training records, indicating potential memorization or data leakage.

Usage

```
ims(X, ...)

## S3 method for class 'synth_pair'
ims(X, ...)
```

```
## Default S3 method:
ims(X, Y, vars = NULL, na.rm = TRUE, ...)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
vars	character vector of variable names to use for matching. If NULL (default), all common variables are used.
na.rm	logical, remove records with NA values (default: TRUE)

Details

For a variant that focuses only on unique (singleton) training records, use repu (Replicated Uniques).

Identical Match Share (IMS) is a straightforward privacy metric that detects exact copies of training records in synthetic data. Even well-designed synthetic data generators can occasionally produce records identical to training data by chance, especially for categorical-heavy datasets.

The metric computes:

$$IMS = \frac{|\{y \in Y : \exists x \in X, y = x\}|}{|Y|}$$

Interpretation:

- **IMS = 0%**: No exact copies - ideal for privacy
- **IMS < 1%**: Acceptable - likely coincidental matches
- **IMS > 1%**: Concerning - may indicate memorization
- **IMS > 5%**: Serious privacy issue - significant copying

For datasets with many categorical variables and few unique combinations, some identical matches may occur by chance. Compare IMS against the expected collision rate for your data characteristics.

Value

An object of class "ims" containing:

- ims: identical match share (proportion of synthetic with exact match)
- ims_pct: IMS as percentage
- n_identical: number of synthetic records with exact match
- identical_idx: indices of identical synthetic records in Y
- identical_records: the actual identical records from Y
- match_counts: number of training matches per identical synthetic record
- n_unique_matched: number of unique training records that were copied
- privacy_pass: logical, TRUE if IMS is acceptably low (< 1%)
- n_synthetic, n_train: dataset sizes

Author(s)

Matthias Templ

References

MOSTLY AI (2024). Synthetic data quality assurance. https://mostly.ai/

See Also

dcr for distance to closest record, nndr for nearest neighbor distance ratio, disco for disclosive records with matching target values

Other distance-risk: dcr(), hitting_rate(), nnaa(), nndr(), repu(), rf_privacy()

Examples

```
# Create example data
set.seed(123)
X <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 100, replace = TRUE),
  income = sample(c("low", "medium", "high"), 100, replace = TRUE)
)

# Good synthetic data (random)
Y_good <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 100, replace = TRUE),
  income = sample(c("low", "medium", "high"), 100, replace = TRUE)
)

result_good <- ims(X, Y_good)
print(result_good)

# Bad synthetic data (direct copies)
Y_bad <- X[sample(nrow(X), 100, replace = TRUE), ]
result_bad <- ims(X, Y_bad)
print(result_bad)
```

---

individual_risk *Individual Re-identification Risk*

Description

Estimates individual re-identification risk based on population frequency models. Computes sample-based and model-based risk estimates using log-linear or Poisson regression models for population frequency estimation.

Usage

```
individual_risk(X, ...)

## S3 method for class 'synth_pair'
individual_risk(X, threshold = 0.1, data = c("synthetic", "original"), ...)

## Default S3 method:
individual_risk(
  X,
  key_vars,
  weight = NULL,
  method = c("both", "sample", "model"),
  threshold = 0.1,
  na.rm = TRUE,
  ...
)
```

Arguments

X	data frame to assess, or a synth_pair object
...	additional arguments passed to methods
threshold	numeric, risk threshold for flagging high-risk records (default: 0.1)
data	character, which dataset to assess: "synthetic" (default) or "original". Only used by the synth_pair method.
key_vars	character vector of quasi-identifier variable names
weight	optional, name of weight variable or numeric vector of sampling weights
method	character, risk estimation method: "sample" (sample frequencies only), "model" (log-linear model), or "both" (default)
na.rm	logical, remove records with NA in key variables (default: TRUE)

Details

Individual risk quantifies the probability that a specific record can be re-identified by an intruder who has access to the quasi-identifiers.

Sample-based Risk: For a record in an equivalence class of size f_k, the sample-based risk is:

$$r_k = \frac{1}{f_k}$$

This is a simple estimate assuming uniform probability within each class.

Model-based Risk: When sampling weights are available or for small samples, model-based approaches estimate the population frequency F_k using log-linear models:

$$r_k = \frac{1}{F_k}$$

The function fits a Poisson log-linear model to estimate expected frequencies in the population, accounting for the sampling design.

Risk Interpretation:

- Risk = 1: Unique record, definitely identifiable
- Risk > 0.5: High risk, appears in small equivalence class
- Risk > 0.1: Elevated risk, deserves attention
- Risk < 0.05: Generally acceptable for most applications

Value

An object of class "individual_risk" containing:

- risk: vector of individual risk values
- method: method used for risk estimation
- mean_risk: mean risk across all records
- max_risk: maximum individual risk
- n_high_risk: number of records exceeding threshold
- pct_high_risk: percentage of high-risk records
- threshold: risk threshold used
- risk_distribution: summary statistics of risk distribution
- ec_info: equivalence class information

Author(s)

Matthias Templ

References

Skinner, C. J., & Elliot, M. J. (2002). A measure of disclosure risk for microdata. *Journal of the Royal Statistical Society: Series B*, 64(4), 855-867.

Rinott, Y., & Shlomo, N. (2006). A generalized negative binomial smoothing model for sample disclosure risk estimation. In *Privacy in Statistical Databases*.

See Also

kanonymity for k-anonymity assessment, suda for Special Uniques Detection, ldiversity for l-diversity assessment

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
data <- data.frame(
  age = sample(c("young", "middle", "old"), 500, replace = TRUE),
  gender = sample(c("M", "F"), 500, replace = TRUE),
```

```
  region = sample(c("N", "S", "E", "W"), 500, replace = TRUE),
  income = sample(c("low", "medium", "high"), 500, replace = TRUE)
)

# Compute individual risk
result <- individual_risk(data,
                          key_vars = c("age", "gender", "region"),
                          threshold = 0.2)
print(result)
summary(result)
plot(result)

# With sampling weights
data$weight <- runif(500, 0.5, 2)
result_weighted <- individual_risk(data,
                                   key_vars = c("age", "gender", "region"),
                                   weight = "weight")
print(result_weighted)
```

`information_surprisal` *Information Surprisal*

Description

Computes the self-information (surprisal) of an outcome `x`, given its probability `p(x)`. Surprisal quantifies how unexpected or informative an event is, and can be used to evaluate privacy risks.

Usage

```
information_surprisal(p, logbase = 2)
```

Arguments

`p`	A numeric vector of probabilities (or frequencies normalized to sum to 1).
`logbase`	The logarithm base used for surprisal (default is 2).

Details

In privacy evaluation, information surprisal measures how surprising an adversary would find a specific combination of attributes. The lower the frequency of this combination in the population, the higher the surprisal and the more identifiable the user becomes.

Mathematically, information surprisal is defined as:

$$I(x) = -\log_2(p(x))$$

A low probability `p(x)` results in a high surprisal value, indicating uniqueness or rarity, which may correspond to a higher risk of re-identification.

Value

A numeric vector of surprisal values.

Author(s)

Matthias Templ

References

Wagner, I., & Eckhoff, D. (2018). Technical Privacy Metrics: A Systematic Survey. ACM Computing Surveys, 51(3), Article 57. doi:10.1145/3168389

See Also

Other information-theory: `Entropy`, `EntropyMeasures`, `max_info_leakage()`, `mutualInformation()`, `positive_information_disclosure()`, `privacy_score()`, `systemAnonymityLevel()`

Examples

```
# Example 1: Basic surprisal values
probs <- c(0.5, 0.25, 0.125, 0.125)
information_surprisal(probs)

# Example 2: Using with a psychological dataset (e.g., personality profiles)
# Suppose these are relative frequencies of specific personality profiles
profile_freqs <- c("TypeA" = 0.4, "TypeB" = 0.35, "TypeC" = 0.2, "TypeD" = 0.05)
information_surprisal(profile_freqs)

# Example 3: Privacy evaluation - rare attribute combination
# Suppose an individual has a rare attribute combination with p(x) = 0.0001
information_surprisal(0.0001)
# Output: 13.29 bits - high risk due to uniqueness
```

---

inspect_record *Inspect a Single Record's Linkage Detail*

Description

Returns detailed linkage diagnostics for a single record, including candidate IDs and (optionally) the data for the query record and its candidates. Requires `recordLinkage(..., return_matches = TRUE)`.

Usage

```
inspect_record(x, ...)

## S3 method for class 'recordLinkageRisk'
inspect_record(x, i, data_orig = NULL, data_anon = NULL, ...)

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

Arguments

x	object of class "inspect_record".
...	ignored.
i	integer, the record index to inspect.
data_orig	optional data frame of the original records (query side).
data_anon	optional data frame of the anonymized records (search side).

Value

An object of class "inspect_record" containing:

record_id the record index

risk re-identification risk

d_rank rank of the true match among candidates

risk_band categorical risk band

d_true, d_min distances for the true match and closest candidate

n_candidates number of candidates

true_in_set whether the true match is in the candidate set

candidate_ids vector of candidate indices

query_record optional: data for the query record

candidate_records optional: data for the candidate records

Author(s)

Matthias Templ, Oscar Thees

See Also

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

kanonymity *k-Anonymity Assessment*

Description

Measures k-anonymity of a dataset based on specified quasi-identifier (key) variables. A dataset satisfies k-anonymity if every record is indistinguishable from at least k-1 other records with respect to the key variables.

Usage

```
kanonymity(X, ...)

## S3 method for class 'synth_pair'
kanonymity(X, k = 5, data = c("synthetic", "original"), ...)

## Default S3 method:
kanonymity(X, key_vars, k = 5, na.rm = TRUE, ...)
```

Arguments

X	data frame to assess, or a synth_pair object
...	additional arguments passed to methods (currently unused)
k	integer, the k-anonymity threshold to check against (default: 5)
data	character, which dataset to assess: "synthetic" (default) or "original". Only used by the synth_pair method.
key_vars	character vector of quasi-identifier variable names
na.rm	logical, remove records with NA in key variables (default: TRUE)

Details

k-Anonymity (Sweeney, 2002) is a privacy model that requires each record to be indistinguishable from at least k-1 other records based on quasi-identifiers.

An equivalence class is the set of all records sharing the same values for all key variables. The k-anonymity level is the size of the smallest equivalence class.

Interpretation:

- **k >= 5**: Generally considered acceptable for public release
- **k >= 3**: Minimum for most applications
- **k = 1**: Unique records exist - high re-identification risk
- **k = 2**: Some records have only one match - elevated risk

Records in small equivalence classes (size < k) are at higher risk of re-identification and may need additional protection.

Value

An object of class "kanonymity" containing:

- k_level: the achieved k-anonymity level (minimum equivalence class size)
- satisfies_k: logical, whether the data satisfies k-anonymity for given k
- n_violating: number of records in equivalence classes smaller than k
- pct_violating: percentage of records violating k-anonymity
- n_ec: number of equivalence classes
- ec_sizes: table of equivalence class size distribution
- violating_records: indices of records violating k-anonymity
- equivalence_classes: data frame with equivalence class details
- risk_summary: summary of re-identification risk

Author(s)

Matthias Templ

References

Sweeney, L. (2002). k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5), 557-570.

See Also

`ldiversity` for l-diversity assessment, `tcloseness` for t-closeness assessment, `suda` for special uniques detection, `individual_risk` for probabilistic risk assessment

Other privacy-models: `attacker_risk()`, `delta_presence()`, `disclosure_report()`, `domias()`, `drisk()`, `epsilon_identifiability()`, `individual_risk()`, `inspect_record()`, `ldiversity()`, `linkability()`, `merge_per_record()`, `mia_classifier()`, `population_uniqueness()`, `recordLinkage()`, `risk_by_group()`, `singling_out()`, `suda()`, `tcloseness()`, `top_at_risk()`

Examples

```
# Create example data
set.seed(123)
data <- data.frame(
  age = sample(c("young", "middle", "old"), 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 100, replace = TRUE),
  income = sample(c("low", "medium", "high"), 100, replace = TRUE)
)

# Check k-anonymity
result <- kanonymity(data, key_vars = c("age", "gender", "region"), k = 5)
print(result)
summary(result)
plot(result)
```

```
# Check with more key variables (likely lower k)
result2 <- kanonymity(data, key_vars = c("age", "gender", "region", "income"), k = 3)
print(result2)
```

`ldiversity` *l-Diversity Assessment*

Description

Measures l-diversity of a dataset, which extends k-anonymity by requiring diversity in sensitive attribute values within each equivalence class. Implements distinct l-diversity, entropy l-diversity, and recursive (c,l)-diversity.

Usage

```
ldiversity(X, ...)

## S3 method for class 'synth_pair'
ldiversity(X, l = 2, c = 2, data = c("synthetic", "original"), ...)

## Default S3 method:
ldiversity(X, key_vars, sensitive_var, l = 2, c = 2, na.rm = TRUE, ...)
```

Arguments

`X`	data frame to assess, or a `synth_pair` object
`...`	additional arguments passed to methods (currently unused)
`l`	integer, the l-diversity threshold to check against (default: 2)
`c`	numeric, the c parameter for recursive (c,l)-diversity (default: 2)
`data`	character, which dataset to assess: "synthetic" (default) or "original". Only used by the synth_pair method.
`key_vars`	character vector of quasi-identifier variable names
`sensitive_var`	character, name of the sensitive attribute
`na.rm`	logical, remove records with NA values (default: TRUE)

Details

l-Diversity (Machanavajjhala et al., 2007) addresses limitations of k-anonymity by requiring diversity in sensitive attributes within equivalence classes.

Distinct l-Diversity: Each equivalence class must have at least l distinct values of the sensitive attribute. The achieved level is the minimum number of distinct values across all equivalence classes.

Entropy l-Diversity: Each equivalence class must have entropy of the sensitive attribute >= log(l). Entropy is computed as: $H = -\sum p_i \log(p_i)$

Recursive (c,l)-Diversity: The most common sensitive value should not appear too frequently relative to other values. Specifically: $r_1 < c(r_l + r_{l+1} + ... + r_m)$ where r_i is the frequency of the i-th most common value.

Interpretation:

- **l >= 3**: Good diversity protection
- **l = 2**: Minimum meaningful diversity
- **l = 1**: No diversity - sensitive value can be inferred

Value

An object of class "ldiversityRisk" containing:

- distinct_l: achieved distinct l-diversity level
- entropy_l: achieved entropy l-diversity level
- recursive_cl: whether data satisfies recursive (c,l)-diversity
- satisfies_l: logical, whether data satisfies l-diversity for given l
- n_violating_distinct: records in EC with fewer than l distinct values
- n_violating_entropy: records in EC with entropy < log(l)
- per_ec: data frame with per-equivalence-class diversity measures
- key_vars, sensitive_var: input parameters

Author(s)

Matthias Templ

References

Machanavajjhala, A., Kifer, D., Gehrke, J., & Venkitasubramaniam, M. (2007). l-Diversity: Privacy Beyond k-Anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1), Article 3.

See Also

kanonymity for k-anonymity assessment, tcloseness for t-closeness assessment, tcap for targeted correct attribution probability

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
data <- data.frame(
  age = sample(c("young", "middle", "old"), 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
```

```
  region = sample(c("N", "S"), 200, replace = TRUE),
  disease = sample(c("healthy", "cold", "flu", "covid"), 200, replace = TRUE,
                   prob = c(0.5, 0.2, 0.2, 0.1))
)

# Check l-diversity
result <- ldiversity(data,
                     key_vars = c("age", "gender", "region"),
                     sensitive_var = "disease",
                     l = 2)
print(result)
summary(result)
plot(result)
```

`linkability` *Linkability Risk*

Description

Computes the Linkability Risk for synthetic data, the second of three explicit GDPR anonymization failure criteria (Article 29 Working Party). An attacker who holds two disjoint subsets of columns from the original data tries to link records across these subsets using the synthetic data as a bridge.

Usage

```
linkability(X, ...)

## S3 method for class 'synth_pair'
linkability(X, ...)

## Default S3 method:
linkability(
  X,
  Y,
  holdout = NULL,
  holdout_fraction = 0.5,
  n_attacks = 2000,
  n_neighbors = 1,
  aux_cols = NULL,
  vars = NULL,
  na.rm = TRUE,
  seed = NULL,
  confidence_level = 0.95,
  ...
)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
holdout	data frame of holdout data (optional). If NULL, a holdout set is automatically created by splitting X.
holdout_fraction	numeric, fraction of X to use as holdout if holdout is NULL (default: 0.5)
n_attacks	integer, number of synthetic records to use for the attack (default: 2000). If larger than nrow(Y), all synthetic records are used.
n_neighbors	integer, number of nearest neighbors to consider for a successful link (default: 1). With n_neighbors = 1, only exact NN matches count. With n_neighbors > 1, a link succeeds if the auxiliary NN is among the top-k secret NNs.
aux_cols	character vector of column names for the auxiliary (attacker-known) subset. If NULL (default), columns are randomly split into two equal halves.
vars	character vector of variable names to use. If NULL (default), all common variables between X and Y are used.
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for reproducibility (default: NULL)
confidence_level	numeric, confidence level for Wilson score intervals (default: 0.95)

Details

The linkability attack simulates an adversary who holds two disjoint sets of columns from the original data (e.g., demographics and medical records) and attempts to re-link them using the synthetic data.

For each sampled synthetic record:

1. Find its nearest neighbor in the original data using only auxiliary columns
2. Find its nearest neighbor in the original data using only secret columns
3. If both nearest neighbors point to the same original record, the link is successful

With n_neighbors > 1, a link succeeds if the auxiliary nearest neighbor is among the top-k secret nearest neighbors, making the attack more powerful.

The risk score uses Wilson score intervals for robust estimation:

$$r_{attack} = n_{success}/n_{attacks}$$

$$r_{control} = n_{control\_success}/n_{attacks}$$

$$risk = (r_{attack} - r_{control})/(1 - r_{control})$$

Without holdout data, the absolute attack rate is reported as the risk.

Gower distance is used for nearest-neighbor computation, which handles mixed (numeric and categorical) variable types.

Value

An object of class "linkability" containing:

- risk: residual risk score, bounded between 0 and 1
- risk_ci: confidence interval for risk
- risk_attack: attack success rate (fraction of links in original)
- risk_attack_ci: Wilson CI for attack success rate
- risk_control: control success rate (fraction of links in holdout, NA if none)
- risk_control_ci: Wilson CI for control success rate (NA if none)
- n_attacks: number of attacks performed
- n_success: number of successful links in original
- n_control_success: number of successful links in holdout
- links: logical vector of link successes per attack (original)
- links_control: logical vector of link successes per attack (holdout)
- privacy_pass: logical, risk <= 0.1
- n_original: number of original records (training portion)
- n_synthetic: number of synthetic records
- n_holdout: number of holdout records
- aux_cols: auxiliary columns used
- secret_cols: secret columns used
- n_neighbors: number of neighbors considered
- vars: all variables used
- confidence_level: confidence level used

Column splitting

When `aux_cols` is not specified, the available columns are randomly split into two approximately equal groups. The split is controlled by `seed`. For reproducible results, always set the seed.

When `aux_cols` is specified, the remaining columns form the secret set. At least one column must be in each group.

Holdout splitting

When no external holdout is provided, the original data is split internally. The holdout serves as a control: links found in the holdout represent baseline linkability due to data structure rather than information leakage. The residual risk subtracts this baseline.

Author(s)

Matthias Templ

References

Giomi, M., Boenisch, F., Wehmeyer, C. & Tasnadi, B. (2023). A Unified Framework for Quantifying Privacy Risk in Synthetic Data. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2023(2), 312–328. doi:10.56553/popets20230055

Article 29 Data Protection Working Party (2014). Opinion 05/2014 on Anonymisation Techniques. WP216.

See Also

singling_out for singling out risk, dcap for differential correct attribution probability, tcap for targeted CAP

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
n <- 200
X <- data.frame(
  age = sample(20:70, n, replace = TRUE),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Good synthetic data (independent, no memorization)
Y <- data.frame(
  age = sample(20:70, n, replace = TRUE),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

result <- linkability(X, Y, n_attacks = 500, seed = 42)
print(result)
summary(result)


# Memorized data (Y is copy of X) - should show high risk
Y_copy <- X[sample(nrow(X), n, replace = TRUE), ]
result_bad <- linkability(X, Y_copy, n_attacks = 500, seed = 42)
print(result_bad)
```

`max_info_leakage` *Maximum Information Leakage*

Description

Computes the maximum information leakage from a private variable `X` to an observed output `Y`. It quantifies the worst-case information gain an adversary can achieve by observing any single output value.

Usage

```
max_info_leakage(joint_dist)
```

Arguments

`joint_dist`	A matrix representing the joint distribution of X and Y, where rows correspond to values of X and columns to values of Y. The matrix should sum to 1.

Details

Maximum Information Leakage is defined as:

$$privMIL = \max_{y \in Y} I(X; Y = y)$$

where $I(X; Y = y)$ is the mutual information conditioned on a single observation.

This is a worst-case metric, assuming that the adversary selects the output value y that maximizes information gain.

This metric is relevant in scenarios where one specific output could lead to a disproportionate privacy breach. See Section 5.2.7 in Wagner and Eckhoff (2018) for a detailed discussion.

Value

A numeric value representing the maximum information leakage (in bits).

Author(s)

Matthias Templ

References

Wagner, I., & Eckhoff, D. (2018). Technical Privacy Metrics: A Systematic Survey. ACM Computing Surveys (CSUR), 51(3), Article 57.

See Also

Other information-theory: `Entropy`, `EntropyMeasures`, `information_surprisal()`, `mutualInformation()`, `positive_information_disclosure()`, `privacy_score()`, `systemAnonymityLevel()`

Examples

```
# Basic example: Define joint distribution of X and Y
joint <- matrix(c(0.1, 0.05, 0.05,
                  0.2, 0.1, 0.1,
                  0.1, 0.15, 0.15), nrow = 3, byrow = TRUE)
# Normalize to sum to 1
joint <- joint / sum(joint)
max_info_leakage(joint)

# Example use in privacy evaluation:
# Suppose X represents income brackets (low, medium, high)
# and Y is an anonymized salary category after applying noise.
# We want to know the worst-case information leakage from salary output Y to true income X.
income <- factor(c("low", "low", "medium", "medium", "high", "high"))
salary <- factor(c("cat1", "cat2", "cat2", "cat2", "cat3", "cat1"))
joint_table <- table(income, salary)
joint_prob <- prop.table(joint_table)
max_info_leakage(joint_prob)
```

---

merge_per_record *Merge Per-Record Risks Back to Data*

Description

Joins per-record risk diagnostics back to the original (or synthetic) data frame.

Usage

```
merge_per_record(x, ...)

## S3 method for class 'recordLinkageRisk'
merge_per_record(x, data, ...)
```

Arguments

x	object of class "recordLinkageRisk".
...	ignored.
data	data frame with the same number of rows as x$per_record.

Value

A data frame combining data and x$per_record.

Author(s)

Matthias Templ, Oscar Thees

See Also

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

`mia_classifier` *Membership Inference Attack metric via classification*

Description

Membership Inference Attack metric via classification

Usage

```
mia_classifier(X, ...)

## S3 method for class 'synth_pair'
mia_classifier(X, ...)

## Default S3 method:
mia_classifier(
  X,
  synt_data = NULL,
  hout_data = NULL,
  cols = NULL,
  cat_cols = NULL,
  num_cols = NULL,
  method = "rf",
  num_eval_iter = 5L,
  seed = NULL,
  num_trees = 300L,
  num_threads = 1L,
  Y = NULL,
  holdout = NULL,
  ...
)
```

Arguments

`X`	synth_pair object, or (for the default method) a data.frame of records used to train the synthetic generator.
`...`	additional arguments passed to methods
`synt_data`	data.frame. Synthetic records produced by the generator.
`hout_data`	data.frame. Holdout records from the same population, NOT used to train the generator.

cols	character vector of column names to use (optional). If NULL, all common
cat_cols	character vector of categorical column names (optional). If NULL, inferred as non-numeric columns.
num_cols	character vector of numeric column names (optional). If NULL, inferred as numeric columns.
method	character. Classification back-end to use. Currently supported: "rf" (random forest via ranger).
num_eval_iter	integer. Number of evaluation iterations with different holdout splits. Default 5.
seed	integer or NULL. Random seed for reproducibility.
num_trees	integer. Number of trees (only used for method = "rf").
num_threads	integer. Threads for parallelism. Default 1.
Y	Alias for synt_data, for naming consistency with the rest of the package. If synt_data is NULL and Y is supplied, Y is used.
holdout	Alias for hout_data. If hout_data is NULL and holdout is supplied, holdout is used.

Value

An object of class "mia" with components:

precision Mean precision across iterations.

precision_se Standard error of precision.

recall Mean recall (primary privacy metric).

recall_se Standard error of recall.

macro_f1 Mean macro F1 score.

macro_f1_se Standard error of macro F1.

privacy_pass Logical, TRUE if recall <= 0.55.

num_eval_iter Number of evaluation iterations.

method Classification method used.

Values near 0.5 indicate performance close to random guessing (low risk). Values > 0.5 indicate the synthetic data leaks training membership. Values < 0.5 indicate performance worse than random guessing.

Author(s)

Matthias Templ

See Also

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

mmd *Maximum Mean Discrepancy (MMD) for Multivariate Data*

Description

Computes the Maximum Mean Discrepancy between two datasets using kernel-based two-sample testing. MMD measures whether two samples come from the same distribution by comparing their embeddings in a reproducing kernel Hilbert space (RKHS). It equals zero if and only if the distributions are identical.

Usage

```
mmd(X, ...)

## S3 method for class 'synth_pair'
mmd(X, ...)

## Default S3 method:
mmd(
  X,
  Y,
  vars = NULL,
  kernel = c("gaussian", "rational_quadratic"),
  method = c("exact", "rff"),
  n_features = 500L,
  n_perm = NULL,
  standardize = TRUE,
  na.rm = TRUE,
  seed = NULL,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods (currently unused)
Y	A data.frame or data.table containing the synthetic/anonymized dataset.
vars	Character vector of numeric variable names to compare. If NULL (default), all common numeric variables are used.
kernel	Character, kernel function to use. One of "gaussian" (default) or "rational_quadratic".
method	Character, computation method. One of "exact" (default, O(n^2) complexity) or "rff" (Random Fourier Features, O(nD) approximation for large datasets).
n_features	Integer, number of random Fourier features when method = "rff". Default 500.
n_perm	Integer or NULL, number of permutations for a permutation test. If NULL (default), no permutation test is performed.

standardize	Logical, whether to standardize variables before computing. Default TRUE (recommended for variables on different scales).
na.rm	Logical, whether to remove rows with NA values. Default TRUE.
seed	Integer, random seed for reproducibility. Default NULL.

Details

The unbiased MMD^2 estimator with kernel k is:

$$MMD^2 = \frac{1}{n(n-1)} \sum_{i \neq j} k(x_i, x_j) + \frac{1}{m(m-1)} \sum_{i \neq j} k(y_i, y_j) - \frac{2}{nm} \sum_i \sum_j k(x_i, y_j)$$

The Gaussian kernel is $k(x, y) = \exp(-\|x-y\|^2/(2\sigma^2))$. The rational quadratic kernel is $k(x, y) = (1 + \|x - y\|^2/(2\sigma^2))^{-1}$.

The bandwidth σ is chosen via the median heuristic: the median of all pairwise Euclidean distances (subsampled to 2000 points for efficiency).

For large datasets ($n > 5000$), consider using `method = "rff"` which approximates the kernel via Random Fourier Features at O(nD) cost instead of O(n^2).

The utility score is computed as $\exp(-MMD^2/\sigma^2)$, mapping to $[0, 1]$ where 1 indicates identical distributions.

Value

An object of class `"mmd"` containing:

- `mmd2`: the unbiased MMD^2 statistic
- `sigma`: the kernel bandwidth (median heuristic)
- `kernel`: the kernel used
- `method`: the computation method used
- `n_X`, `n_Y`: sample sizes
- `n_vars`: number of variables
- `vars`: variable names used
- `standardized`: whether standardization was applied
- `utility_score`: transformed score in $[0, 1]$ (higher = better)
- `perm_pvalue`: permutation test p-value (if `n_perm` was set)
- `perm_null`: vector of permutation MMD^2 values (if `n_perm` was set)

Author(s)

Matthias Templ

References

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schoelkopf, B., Smola, A. (2012). A Kernel Two-Sample Test. Journal of Machine Learning Research, 13, 723-773.

Rahimi, A. and Recht, B. (2007). Random Features for Large-Scale Kernel Machines. Advances in Neural Information Processing Systems, 20.

See Also

energy_distance for energy distance, compare_wasserstein for univariate Wasserstein distance, gower for mixed-type data

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), gower(), hellinger(), mqs(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), rumap(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
set.seed(123)
# Original data
X <- data.frame(
  income = rnorm(200, mean = 50000, sd = 10000),
  age = rnorm(200, mean = 40, sd = 10),
  score = rnorm(200, mean = 100, sd = 15)
)

# Good synthetic data (similar distribution)
Y_good <- data.frame(
  income = rnorm(200, mean = 50000, sd = 10000),
  age = rnorm(200, mean = 40, sd = 10),
  score = rnorm(200, mean = 100, sd = 15)
)

# Poor synthetic data (shifted distribution)
Y_poor <- data.frame(
  income = rnorm(200, mean = 60000, sd = 15000),
  age = rnorm(200, mean = 50, sd = 15),
  score = rnorm(200, mean = 80, sd = 25)
)

result_good <- mmd(X, Y_good, seed = 42)
print(result_good)

result_poor <- mmd(X, Y_poor, seed = 42)
print(result_poor)


# With permutation test
result_perm <- mmd(X, Y_poor, n_perm = 200, seed = 42)
summary(result_perm)

# Using Random Fourier Features
result_rff <- mmd(X, Y_good, method = "rff", seed = 42)
print(result_rff)
```

---

mqs *Model Quality Score*

Description

Computes the prediction quality or accuracy of ML models

Usage

```
mqs(X, ...)

## S3 method for class 'synth_pair'
mqs(X, ...)

## Default S3 method:
mqs(
  X,
  Y,
  form,
  methods = c("glm", "knn", "simpls", "rpart", "ranger"),
  na = "remove",
  ntop = length(methods),
  ...
)
```

Arguments

X	data frame
...	additional arguments passed to methods
Y	data frame
form	model formula
methods	classification or regression methods. In principle, all methods supported from R package caret can be used.
na	missing value treatment (remove, impute or stop)
ntop	number of top models considered for the mqs statistics

Details

It computes the prediction quality (RMSE) when the response is numeric, and the accuracy when the response is a factor variable. The computation is done for both, X (typically a non-anonymized data set) and Y (typically the anonymized or synthetisized version of X) and compares the estimates.

The ratio compares these estimates: a value close to 1 indicates comparable prediction quality, a ratio above 1 indicates that the synthetic data have lower prediction quality (they preserve less predictive structure) than the original data, and a ratio below 1 indicates that the synthetic data are at least as predictable as the original.

Value

An object of class "mqs" containing:

- mqs_ratio: the model quality statistics ratio
- mqs_table: data frame with model performance comparison

Author(s)

Matthias Templ

See Also

propscore, compare_model_performance

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), gower(), hellinger(), mmd(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), rumap(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
# Requires the caret, caretEnsemble and rpart packages
if (requireNamespace("caret", quietly = TRUE) &&
    requireNamespace("caretEnsemble", quietly = TRUE) &&
    requireNamespace("rpart", quietly = TRUE)) {
  set.seed(123)
  X <- data.frame(
    y = factor(sample(c("A", "B"), 100, replace = TRUE)),
    x1 = rnorm(100),
    x2 = rnorm(100)
  )
  Y <- data.frame(
    y = factor(sample(c("A", "B"), 100, replace = TRUE)),
    x1 = rnorm(100, 0.1, 1),
    x2 = rnorm(100, 0.1, 1)
  )
  m <- mqs(X, Y, form = y ~ x1 + x2, methods = c("glm", "rpart"))
  m
}
```

---

mutualInformation *Mutual Information Privacy Metrics*

Description

Computes various information-theoretic privacy metrics based on entropy, mutual information, and related measures. Includes:

- Mutual Information (MI)
- Normalized Mutual Information (NMI)
- Conditional Privacy Loss (CPL)
- Conditional Mutual Information (CMI)
- Relative Loss of Anonymity (RLA)

Usage

```
mutualInformation(X, Y, joint, Z = NULL, normalized = FALSE)
```

Arguments

X	A matrix, data frame or probability vector representing the true data distribution.
Y	A matrix, data frame or probability vector representing the observed or obfuscated data.
joint	A joint probability matrix for X and Y.
Z	Optional third variable (matrix or data.frame) representing prior knowledge for conditional metrics.
normalized	Logical; whether to normalize mutual information (default: FALSE).

Details

Mutual Information (MI) quantifies how much information is shared between two random variables X and Y. It is computed as:

$$MI(X, Y) = H(X) - H(X|Y)$$

Conditional Mutual Information (CMI) measures the information shared between X and Y, given prior knowledge Z:

$$CMI(X, Y|Z) = H(X|Z) - H(X|Y, Z)$$

Conditional Privacy Loss (CPL) is a normalized measure based on MI:

$$CPL = 1 - 2^{-MI(X,Y)}$$

Relative Loss of Anonymity (RLA) can be approximated using conditional mutual information. It reflects the maximum leakage given prior knowledge.

The normalized mutual information (`normalized = TRUE`) follows:

$$NMI(X, Y) = \frac{MI(X, Y)}{H(X)}$$

or can be normalized by the number of records (rows in `X`) for privacy interpretation.

Interpreting "greater" MI: Higher MI means more information is leaked. But "greater" only implies higher *relative* risk. If the absolute MI is low (e.g., < 0.1 bits), the disclosure risk might still be negligible, even if it's greater than another case.

Value

A list of privacy metrics: MI, NMI, CPL, CMI, and RLA (if Z is provided).

Author(s)

Matthias Templ

References

Shannon, C.E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3), 379–423.

Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*. 2nd edition. Wiley-Interscience.

See Also

Other information-theory: Entropy, EntropyMeasures, information_surprisal(), max_info_leakage(), positive_information_disclosure(), privacy_score(), systemAnonymityLevel()

Examples

```
# Privacy example with synthetic probability distributions
X <- c(0.2, 0.5, 0.3)
Y <- c(0.25, 0.45, 0.3)
jointXY <- matrix(c(0.1, 0.05, 0.05,
                    0.1, 0.2,  0.15,
                    0.05, 0.2, 0.1), nrow = 3, byrow = TRUE)
mutualInformation(X, Y, jointXY, normalized = TRUE)
```

---

nnaa *Nearest-Neighbor Adversarial Accuracy (NNAA)*

Description

Computes the Nearest-Neighbor Adversarial Accuracy and Privacy Loss metrics for synthetic data evaluation. NNAA uses a 1-nearest-neighbor two-sample test to assess whether synthetic data is distinguishable from real data. Privacy Loss compares adversarial accuracy on training versus hold-out data to detect memorization.

Usage

```
nnaa(X, ...)

## S3 method for class 'synth_pair'
nnaa(X, ...)

## Default S3 method:
nnaa(
  X,
  Y,
  holdout = NULL,
  holdout_fraction = 0.5,
  vars = NULL,
  method = c("gower", "euclidean"),
```

```
  na.rm = TRUE,
  seed = NULL,
  ...
)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
holdout	data frame of holdout data (optional). If NULL, a holdout set is automatically created by splitting X.
holdout_fraction	numeric, fraction of X to use as holdout if holdout is NULL (default: 0.5)
vars	character vector of variable names to use for distance calculation. If NULL (default), all common variables between X, Y, and holdout are used.
method	character, distance method: "gower" (default, handles mixed types) or "euclidean" (numerical variables only)
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for holdout sampling (default: NULL)

Details

Adversarial Accuracy (AA) measures how well a 1-nearest-neighbor classifier can distinguish real from synthetic data. It is defined as:

$$AA(T, S) = 0.5 \cdot (\text{share}(d_{TS} > d_{TT}) + \text{share}(d_{ST} > d_{SS}))$$

where:

- $d_{TS}(i)$: distance from real point i to its nearest synthetic neighbor
- $d_{TT}(i)$: distance from real point i to its nearest real neighbor (self-excluded)
- $d_{ST}(i)$: distance from synthetic point i to its nearest real neighbor
- $d_{SS}(i)$: distance from synthetic point i to its nearest synthetic neighbor (self-excluded)

Interpretation of AA:

- **AA ~ 0.5**: Ideal - real and synthetic are indistinguishable (good utility)
- **AA < 0.5**: Synthetic is too close to real - possible memorization (privacy concern)
- **AA > 0.5**: Synthetic differs from real - poor utility

Privacy Loss compares AA on training versus holdout data:

$$PrivacyLoss = AA(holdout, S) - AA(train, S)$$

Interpretation of Privacy Loss:

- **~0**: Good privacy - synthetic is equally close to training and holdout
- **> 0**: Privacy concern - synthetic is closer to training than holdout
- **~ 0.5**: Severe memorization detected

Value

An object of class "nnaa" containing:

- aa_train: adversarial accuracy on training data vs synthetic
- aa_holdout: adversarial accuracy on holdout data vs synthetic (NA if no holdout)
- privacy_loss: aa_holdout - aa_train (NA if no holdout)
- privacy_pass: logical, TRUE if privacy_loss <= 0.1
- aa_train_left: proportion where real points are closer to other real than to synthetic
- aa_train_right: proportion where synthetic points are closer to real than to other synthetic
- aa_holdout_left: holdout left component (NA if no holdout)
- aa_holdout_right: holdout right component (NA if no holdout)
- d_TS: nearest-neighbor distances from training to synthetic (per record)
- d_TT: nearest-neighbor distances within training, self-excluded (per record)
- n_train, n_synthetic, n_holdout: dataset sizes
- method, vars, holdout_fraction: parameters used

Holdout splitting

When no external holdout is provided, NNAA internally splits the original data. With default `holdout_fraction = 0.5`, half of X becomes the holdout and half is used as the training set. This is standard practice for NNAA. For best results, provide an actual holdout set from the synthesis process.

Author(s)

Matthias Templ

References

Yale, A., Dash, S., Dutta, R., Guyon, I., Pavao, A. & Bennett, K.P. (2020). Generation and Evaluation of Privacy Preserving Synthetic Health Data. *Neurocomputing*, 416, 244–255. doi:10.1016/j.neucom.2019.12.136

See Also

`dcr` for distance to closest record, `nndr` for nearest neighbor distance ratio, `ims` for identical match share

Other distance-risk: `dcr()`, `hitting_rate()`, `ims()`, `nndr()`, `repu()`, `rf_privacy()`

Examples

```
# Create example data
set.seed(123)
n <- 200
X <- data.frame(
  age = rnorm(n, 40, 10),
```

```
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Good synthetic data (random, no memorization)
Y_good <- data.frame(
  age = rnorm(n, 40, 10),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Compute NNAA
result <- nnaa(X, Y_good, seed = 42)
print(result)
summary(result)
```

---

nndr *Nearest Neighbor Distance Ratio (NNDR)*

Description

Computes the Nearest Neighbor Distance Ratio privacy metric for synthetic data. NNDR detects potential memorization by comparing the distance to the closest record versus the second closest record. Low NNDR values indicate suspicious proximity to specific training records.

Usage

```
nndr(X, ...)

## S3 method for class 'synth_pair'
nndr(X, ...)

## Default S3 method:
nndr(
  X,
  Y,
  holdout = NULL,
  holdout_fraction = 0.5,
  vars = NULL,
  method = c("gower", "euclidean"),
  na.rm = TRUE,
  seed = NULL,
  progress = FALSE,
  ...
)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
holdout	data frame of holdout data (optional). If NULL, a holdout set is automatically created by splitting X.
holdout_fraction	numeric, fraction of X to use as holdout if holdout is NULL (default: 0.5)
vars	character vector of variable names to use for distance calculation. If NULL (default), all common variables are used.
method	character, distance method: "gower" (default, handles mixed types) or "euclidean" (numerical variables only)
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for holdout sampling (default: NULL)
progress	logical, show progress bar for long computations (default: FALSE)

Details

The Nearest Neighbor Distance Ratio (NNDR) is defined for each synthetic record as:

$$NNDR = \frac{d_1}{d_2}$$

where d_1 is the distance to the nearest neighbor and d_2 is the distance to the second nearest neighbor.

Interpretation:

- **NNDR close to 1**: The two nearest neighbors are similarly distant, suggesting the synthetic record is not unusually close to any specific training record (good privacy).
- **NNDR close to 0**: The nearest neighbor is much closer than the second nearest, suggesting potential memorization or copying of a specific training record (privacy concern).

The metric compares NNDR distributions for synthetic-to-training versus synthetic-to-holdout. If the training NNDR distribution has significantly more low values, this indicates privacy leakage.

Value

An object of class "nndr" containing:

- nndr_train: NNDR values for synthetic vs training data
- nndr_holdout: NNDR values for synthetic vs holdout data (reference)
- nndr_ratio: ratio of 5th percentile NNDR (train/holdout)
- mean_nndr_train, mean_nndr_holdout: mean NNDR values
- n_suspicious: count of very low NNDR values (< 0.1)
- privacy_pass: logical, TRUE if NNDR distribution is acceptable

Holdout splitting (important)

Like dcr, NNDR internally splits the original data when no external holdout is provided. With default holdout_fraction = 0.5, only half of the original data is used for comparison. This reduces effective sample size and may miss disclosure if the synthetic data memorized records that ended up in the holdout. Providing a separate holdout from the synthesis process is recommended when available. See dcr for detailed guidance.

Author(s)

Matthias Templ

References

MOSTLY AI (2024). Synthetic data quality assurance. https://mostly.ai/

Lowe, D.G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2), 91-110.

See Also

dcr for distance to closest record, ims for exact match detection, nnaa for nearest-neighbor adversarial accuracy

Other distance-risk: dcr(), hitting_rate(), ims(), nnaa(), repu(), rf_privacy()

Examples

```
# Create example data
set.seed(123)
n <- 300
X <- data.frame(
  age = rnorm(n, 40, 10),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Good synthetic data
Y_good <- data.frame(
  age = rnorm(200, 40, 10),
  income = rnorm(200, 50000, 15000),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE)
)

result <- nndr(X, Y_good, seed = 42)
print(result)
summary(result)
plot(result)
```

plot.attacker_risk *Plot method for attacker_risk objects*

Description

Plot method for attacker_risk objects

Usage

```
## S3 method for class 'attacker_risk'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "attacker_risk"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot(s) to show: 1 = Per-record risk distribution (histogram), 2 = Comparison across models (barplot of global risks)

Value

No return value; called for the side effect of producing a plot.

plot.chisq_utility *Plot method for chisq_utility objects*

Description

Plot method for chisq_utility objects

Usage

```
## S3 method for class 'chisq_utility'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "chisq_utility"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = proportion comparison, 2 = residuals

Value

No return value; called for the side effect of producing a plot.

plot.ci_proximity *Plot method for ci_proximity objects*

Description

Plot method for ci_proximity objects

Usage

```
## S3 method for class 'ci_proximity'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "ci_proximity"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = CI comparison plot, 2 = proximity scores bar chart, 3 = error vs overlap scatter

Value

No return value; called for the side effect of producing a plot.

plot.compare_distributions_cont *Plot Method for Objects of Class "compare_distributions_cont"*

Description

This function provides visualizations for objects of class "compare_distributions_cont" produced by the compare_distributions_cont function. It generates plots of the empirical cumulative distribution functions (ECDF) and can optionally produce interactive plots.

Usage

```
## S3 method for class 'compare_distributions_cont'
plot(x, ..., which = "ecdf", interactive = FALSE)
```

Arguments

x	An object of class "compare_distributions_cont".
...	Additional arguments passed to the plotting function.
which	A character string specifying the type of plot to generate. Options include "ecdf", "density", "density_bayes", "density_ratio", and "density_ratio_bayes". Default is "ecdf". Unlike the integer which used by most plot methods in this package, a character string is used here on purpose: each option names a conceptually distinct diagnostic rather than an interchangeable panel index, so a descriptive label is clearer than a number.
interactive	A logical value indicating whether to produce an interactive plot using plotly. Default is FALSE.

Details

The plot.compare_distributions_cont function generates various types of plots for objects of class "compare_distributions_cont". The primary plot type is the ECDF plot, which can be generated by setting which to "ecdf". Additional plot types ("density", "density_bayes", "density_ratio", and "density_ratio_bayes") are currently not implemented and will raise an error if requested.

The function supports both static and interactive plots. If interactive = TRUE, the function will use plotly to create an interactive version of the plot.

Value

Invisibly returns the ggplot2 object or the plotly object if interactive = TRUE.

Examples

```
S <- data.frame(
  age = sample(20:80, 100, replace = TRUE),
  income = rnorm(100, mean = 50000, sd = 10000),
  weights = runif(100, 0.5, 2)
)
U <- data.frame(
  age = sample(20:80, 10000, replace = TRUE),
  income = rnorm(10000, mean = 50000, sd = 10000)
)

# Compare ECDFs for age and income
result <- compare_distributions_cont(S, U, vars = c("age", "income"), weights = "weights")

# Plot the ECDF
plot(result)


# Interactive version of the plot (requires the plotly package)
if (requireNamespace("plotly", quietly = TRUE)) {
  plot(result, interactive = TRUE)
}
```

plot.compare_feature_importance *Plot method for compare_feature_importance objects*

Description

Plot method for compare_feature_importance objects

Usage

```
## S3 method for class 'compare_feature_importance'
plot(x, y = NULL, which = 1, ...)
```

Arguments

x	an object of class "compare_feature_importance"
y	not used
which	integer, which plot to produce (default 1). Currently only 1 is available.
...	additional arguments passed to plotting functions

Value

No return value; called for the side effect of producing a plot.

plot.contingency_fidelity *Plot method for contingency_fidelity objects*

Description

Plot method for contingency_fidelity objects

Usage

```
## S3 method for class 'contingency_fidelity'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "contingency_fidelity"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = heatmap of pairwise TV distances

Value

No return value; called for the side effect of producing a plot.

---

plot.copula_fidelity *Plot method for copula_fidelity objects*

Description

Plot method for copula_fidelity objects

Usage

```
## S3 method for class 'copula_fidelity'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "copula_fidelity"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = heatmap of pairwise copula distances

Value

No return value; called for the side effect of producing a plot.

---

plot.dcap *Plot method for dcap objects*

Description

Plot method for dcap objects

Usage

```
## S3 method for class 'dcap'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "dcap"
y	not used
...	additional arguments passed to the plot method
which	which plot(s) to show: 1 for CAP histogram, 2 for matches histogram

Value

No return value; called for the side effect of producing a plot.

`plot.dcr` *Plot method for dcr objects*

Description

Plot method for dcr objects

Usage

```
## S3 method for class 'dcr'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "dcr"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	which plot(s) to show: 1 = overlaid histograms, 2 = boxplot comparison, 3 = scatter plot of train vs holdout DCR

Value

No return value; called for the side effect of producing a plot.

`plot.delta_presence` *Plot method for delta_presence objects*

Description

Plot method for delta_presence objects

Usage

```
## S3 method for class 'delta_presence'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "delta_presence"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	integer, which plot(s) to show: 1 = histogram of membership probabilities with delta bounds, 2 = per-QI-combination barplot sorted by probability

Value

No return value; called for the side effect of producing a plot.

`plot.denpca` *Plot method for denpca objects*

Description

This function plots objects of class `denpca`.

Usage

```
## S3 method for class 'denpca'
plot(x, y, ..., which = 1)
```

Arguments

`x`	An object of class `denpca`.
`y`	Not used.
`...`	Additional arguments passed to the plotting functions.
`which`	Which plot to show: 1 for density, 2 for density ratio.

Value

A plot.

`plot.denratio` *Plot method for denratio objects*

Description

This function plots objects of class `denpca`.

Usage

```
## S3 method for class 'denratio'
plot(x, y, ..., which = 1)
```

Arguments

`x`	An object of class `denpca`.
`y`	Not used.
`...`	Additional arguments passed to the plotting functions.
`which`	Which plot to show: 1 for density, 2 for density ratio.

Value

A plot.

`plot.disclosure_report` *Plot method for disclosure_report objects*

Description

Plot method for disclosure_report objects

Usage

```
## S3 method for class 'disclosure_report'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "disclosure_report"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	which plot to show: 1 = summary bar chart, 2 = metric details

Value

No return value; called for the side effect of producing a plot.

`plot.disco` *Plot method for disco objects*

Description

Plot method for disco objects

Usage

```
## S3 method for class 'disco'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "disco"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	which plot(s) to show: 1 = DiSCO vs non-DiSCO bar chart, 2 = comparison with baseline, 3 = comparison of both methods

Value

No return value; called for the side effect of producing a plot.

`plot.domias` *Plot method for domias objects*

Description

Plot method for domias objects

Usage

```
## S3 method for class 'domias'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "domias"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	which plot(s) to show: 1 = density ratio distributions (train vs holdout overlaid histograms), 2 = boxplot comparison of density ratios

Value

No return value; called for the side effect of producing a plot.

`plot.drisk` *Plot method for drisk objects*

Description

Plot method for drisk objects

Usage

```
## S3 method for class 'drisk'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "drisk"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	which plot(s) to show: 1 = per-record risk indicator strip chart, 2 = minimum RMD distance distribution (only available when method includes "rmd")

Value

No return value; called for the side effect of producing a plot.

---

plot.energy_distance *Plot method for energy_distance objects*

Description

Plot method for energy_distance objects

Usage

```
## S3 method for class 'energy_distance'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "energy_distance"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = distance comparison bar chart, 2 = utility gauge

Value

No return value; called for the side effect of producing a plot.

---

plot.epsilon_identifiability *Plot method for epsilon_identifiability objects*

Description

Plot method for epsilon_identifiability objects

Usage

```
## S3 method for class 'epsilon_identifiability'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "epsilon_identifiability"
y	not used
...	additional arguments passed to plotting functions
which	which plot(s) to show: 1 = histogram of minimum distances with epsilon threshold line, 2 = entropy weights barplot

Value

No return value; called for the side effect of producing a plot.

`plot.gower` *Plot method for gower objects*

Description

Visualizes the average Gower distance as a simple bar plot with a reference scale from 0 (identical) to 1 (maximally different).

Usage

```
## S3 method for class 'gower'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class `"gower"`
`y`	not used
`...`	additional arguments passed to `barplot`
`which`	integer, which plot: 1 = bar chart of distance and utility score

Value

No return value; called for the side effect of producing a plot.

`plot.hellinger` *Plot method for hellinger objects*

Description

Plot method for hellinger objects

Usage

```
## S3 method for class 'hellinger'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "hellinger"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	integer, which plot: 1 = bar chart of Hellinger distances, 2 = dotchart sorted by distance

Value

No return value; called for the side effect of producing a plot.

---

plot.hitting_rate *Plot method for hitting_rate objects*

Description

Plot method for hitting_rate objects

Usage

```
## S3 method for class 'hitting_rate'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "hitting_rate"
y	not used
...	additional arguments passed to plotting functions
which	which plot(s) to show: 1 = histogram of min distances with threshold line, 2 = hitting rate vs threshold curve (sweeps thresholds from 0 to max distance)

Value

No return value; called for the side effect of producing a plot.

---

plot.ims *Plot method for ims objects*

Description

Plot method for ims objects

Usage

```
## S3 method for class 'ims'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "ims"
y	not used
...	additional arguments passed to plotting functions
which	which plot to show: 1 = pie chart of identical vs unique, 2 = bar chart of match counts (if any identical records exist)

Value

No return value; called for the side effect of producing a plot.

---

plot.individual_risk *Plot method for individual_risk objects*

Description

Plot method for individual_risk objects

Usage

```
## S3 method for class 'individual_risk'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "individual_risk"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = risk distribution histogram, 2 = risk by equivalence class size

Value

No return value; called for the side effect of producing a plot.

---

plot.kanonymity *Plot method for kanonymity objects*

Description

Plot method for kanonymity objects

Usage

```
## S3 method for class 'kanonymity'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "kanonymity"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = EC size distribution, 2 = cumulative distribution

Value

No return value; called for the side effect of producing a plot.

---

plot.ldiversityRisk *Plot method for ldiversityRisk objects*

Description

Plot method for ldiversityRisk objects

Usage

```
## S3 method for class 'ldiversityRisk'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "ldiversityRisk"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = distinct values distribution, 2 = entropy distribution

Value

No return value; called for the side effect of producing a plot.

---

plot.linkability *Plot method for linkability objects*

Description

Plot method for linkability objects

Usage

```
## S3 method for class 'linkability'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "linkability"
y	not used
...	additional arguments passed to plotting functions
which	which plot(s) to show: 1 = Risk comparison barplot (attack vs control), 2 = Success rate by column group (aux vs secret column counts)

Value

No return value; called for the side effect of producing a plot.

---

plot.mia *Plot method for mia objects*

Description

Plot method for mia objects

Usage

```
## S3 method for class 'mia'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "mia"
y	not used
...	additional arguments (ignored)
which	integer, which plot: 1 = metric barplot (default)

Value

No return value; called for the side effect of producing a plot.

---

plot.missingCompare *Plot method for missingCompare objects*

Description

Plot method for missingCompare objects

Usage

```
## S3 method for class 'missingCompare'
plot(x, ...)
```

Arguments

x	An object of class "missingCompare".
...	Additional arguments (ignored).

Value

No return value; called for the side effect of producing a plot.

plot.mmd *Plot method for mmd objects*

Description

Plot method for mmd objects

Usage

```
## S3 method for class 'mmd'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "mmd"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = permutation null distribution histogram (requires n_perm to have been set)

Value

No return value; called for the side effect of producing a plot.

plot.mqs *Plot method for mqs objects*

Description

Visualizes the model performance comparison between original (X) and synthetic (Y) data.

Usage

```
## S3 method for class 'mqs'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "mqs"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = grouped bar chart of per-model performance, 2 = MQS ratio dot plot

Value

No return value; called for the side effect of producing a plot.

plot.nnaa *Plot method for nnaa objects*

Description

Plot method for nnaa objects

Usage

```
## S3 method for class 'nnaa'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "nnaa"
y	not used
...	additional arguments passed to plotting functions
which	which plot(s) to show: 1 = AA comparison barplot (train vs holdout), 2 = NN distance distributions (overlay histograms of d_TS vs d_TT)

Value

No return value; called for the side effect of producing a plot.

plot.nndr *Plot method for nndr objects*

Description

Plot method for nndr objects

Usage

```
## S3 method for class 'nndr'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "nndr"
y	not used
...	additional arguments passed to plotting functions
which	which plot(s) to show: 1 = overlaid histograms, 2 = boxplot comparison, 3 = cumulative distribution comparison

Value

No return value; called for the side effect of producing a plot.

plot.pMSE *Plot method for pMSE objects*

Description

Plot method for pMSE objects

Usage

```
## S3 method for class 'pMSE'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "pMSE"
y	not used
...	additional arguments passed to plotting functions
which	integer vector, which plots to show: • 1: Propensity score density comparison (default) • 2: Propensity score ECDF comparison • 3: Boxplot of propensity scores by group • 4: Histogram comparison

Value

No return value; called for the side effect of producing a plot.

plot.population_uniqueness *Plot method for population_uniqueness objects*

Description

Plot method for population_uniqueness objects

Usage

```
## S3 method for class 'population_uniqueness'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "population_uniqueness"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot(s) to show: 1 = per-record risk distribution histogram, 2 = method comparison (only when method="all")

Value

No return value; called for the side effect of producing a plot.

---

plot.propscore *Plot method for propscore objects*

Description

Plot method for propscore objects

Usage

```
## S3 method for class 'propscore'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "propscore"
y	not used
...	additional arguments passed to the plot method
which	which plot to show: 1 for density, 2 for density ratio, 3 for proximity structure (ranger only), 4 for variable importance (ranger only)

Value

No return value; called for the side effect of producing a plot.

plot.rapid *Plot method for rapid objects*

Description

Creates visualizations for RAPID disclosure risk assessment results. Six plot types are available via the `which` parameter.

Usage

```
## S3 method for class 'rapid'
plot(
  x,
  y = NULL,
  ...,
  which = 1,
  xlim = "full",
  annotate = TRUE,
  facet_by = NULL,
  bins = 30,
  tau_range = seq(0, 1, by = 0.05),
  epsilon_range = seq(0, 1, by = 0.05),
  formula = NULL,
  top_n = 15
)
```

Arguments

`x`	an object of class "rapid"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	integer indicating which plot to show: • `1`: Risk score distribution histogram (default) • `2`: Prediction scatter plot (numeric targets only) • `3`: Threshold sensitivity curve (RAPID vs threshold range) • `4`: QI variable importance lollipop chart (requires `store_model = TRUE`) • `5`: QI attribution main effects (sum-to-zero logistic regression) • `6`: QI interaction effects (pairwise interactions, requires >= 2 key variables)
`xlim`	character or numeric vector controlling x-axis limits for risk distribution (plots 1 and 2 only): • `"full"` (default): show full data range • `"zero_to_one"`: fixed range from 0 to 1 • `"threshold_to_one"`: range from threshold to 1 (at-risk region only) • numeric vector of length 2: custom limits, e.g., `c(-0.5, 1)`

annotate	logical, if TRUE (default) add annotation showing percentage at risk
facet_by	character vector of variable names to facet by (creates subgroup panels). Must be columns present in the records data frame. Uses ggplot2 for faceted plots.
bins	integer, number of histogram bins (default: 30)
tau_range	numeric vector of tau thresholds for the sensitivity curve (plot 3, categorical targets). Default `seq(0, 1, by = 0.05)`.
epsilon_range	numeric vector of epsilon thresholds for the sensitivity curve (plot 3, numeric targets). Default `seq(0, 1, by = 0.05)`; automatically adjusted to `seq(0, 100, by = 5)` for percentage-based metrics.
formula	optional formula for QI attribution plot (plot 5), e.g., `at_risk ~ age * gender`. If NULL, uses main effects of all key variables.
top_n	integer, maximum number of variables/terms to show in importance (plot 4), attribution (plot 5), and interaction (plot 6) plots. Default 15.

Details

Plot 1 – Risk Distribution: Histogram of risk scores colored by at-risk status. The red dashed line indicates the threshold. For categorical targets, higher scores indicate higher risk. For numeric targets, lower prediction errors indicate higher risk (accurate predictions imply disclosure).

Plot 2 – Prediction Scatter (numeric only): Scatter plot of predicted vs actual values, colored by at-risk status.

Plot 3 – Threshold Sensitivity Curve: Shows how RAPID (proportion at risk) changes across a range of threshold values. The current threshold is marked with a red point. Useful for understanding how sensitive the risk assessment is to threshold choice.

Plot 4 – QI Variable Importance: Lollipop chart of relative variable importance from the attacker model. Requires the model to be stored via `rapid(..., store_model = TRUE)`. Supports random forest, CART, XGBoost, and linear/logistic models.

Plot 5 – QI Attribution (Main Effects): Sum-to-zero contrast logistic regression on at-risk status. Shows which quasi-identifier values contribute most to disclosure risk. Numeric variables are discretized into quartile bins. All factor levels are shown including the implied reference level. Positive coefficients (firebrick) increase risk; negative coefficients (steelblue) decrease risk. Whiskers show 95\

Plot 6 – QI Interaction Effects: Same logistic regression framework as plot 5, but fits all pairwise interactions and displays only the interaction coefficients. Requires at least 2 key variables.

Value

For faceted plots, returns a ggplot2 object (invisibly). For base plots, returns NULL invisibly.

Examples

```
set.seed(42)
n <- 500
original <- data.frame(
  age = sample(20:60, n, replace = TRUE),
  gender = factor(sample(c("M", "F"), n, replace = TRUE)),
```

```
  education = factor(sample(c("High", "Medium", "Low"), n, replace = TRUE))
)
original$health <- factor(ifelse(
  original$age > 40 & original$gender == "M",
  sample(c("Poor", "Fair", "Good"), n, replace = TRUE, prob = c(0.6, 0.3, 0.1)),
  sample(c("Poor", "Fair", "Good"), n, replace = TRUE, prob = c(0.1, 0.3, 0.6))
))
synthetic <- original
synthetic$health <- original$health
idx <- sample(n, n * 0.3)
synthetic$health[idx] <- factor(
  sample(c("Good", "Fair", "Poor"), length(idx), replace = TRUE)
)

result <- rapid(original, synthetic,
                key_vars = c("age", "gender", "education"),
                target_var = "health",
                model_type = "rf",
                cat_tau = 1,
                return_all_records = TRUE,
                store_model = TRUE)

# Risk distribution histogram (default)
plot(result)

# Threshold sensitivity curve
plot(result, which = 3)

# QI variable importance
plot(result, which = 4)

# QI attribution (main effects)
plot(result, which = 5)

# QI interaction effects
plot(result, which = 6)

# Facet by gender
plot(result, facet_by = "gender")
```

`plot.recordLinkageRisk` *Plot method for recordLinkageRisk*

Description

Plot method for recordLinkageRisk

Usage

```
## S3 method for class 'recordLinkageRisk'
plot(x, y = NULL, ..., which = 1, group = NULL, data = NULL)
```

Arguments

x	object of class "recordLinkageRisk".
y	not used.
...	additional arguments passed to plotting functions.
which	integer, which plot(s) to show: 1 = Risk distribution histogram, 2 = Distance/score vs risk scatterplot, 3 = Per-variable importance (all methods), 4 = Propensity score distributions (predictive method only), 5 = Risk band barplot, 6 = True-match rank distribution, 7 = Risk by group barplot (requires group), 8 = Lorenz curve of risk concentration
group	optional grouping vector or column name (with data) for which = 7.
data	optional data frame for extracting group column.

Value

No return value; called for the side effect of producing a plot.

---

plot.regression_fidelity *Plot method for regression_fidelity objects*

Description

Plot method for regression_fidelity objects

Usage

```
## S3 method for class 'regression_fidelity'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "regression_fidelity"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot(s) to produce: 1 = forest plot comparing original and synthetic coefficients with CIs, 2 = CI overlap bar chart per coefficient

Value

No return value; called for the side effect of producing a plot.

`plot.rumap` *Plot method for rumap objects*

Description

Provides multiple visualization approaches for multivariate Risk-Utility evaluation as described in "Beyond the Trade-off Curve" (Thees, Müller, Templ 2026).

Usage

```
## S3 method for class 'rumap'
plot(
  x,
  y = NULL,
  which = 1,
  ...,
  show_pareto = TRUE,
  show_labels = TRUE,
  col_pareto = "steelblue",
  col_dominated = "gray60"
)
```

Arguments

`x`	An object of class "rumap"
`y`	Not used
`which`	Integer vector specifying which plots to produce: • 1: Composite scatterplot (R-U map with Pareto front) • 2: Heatmap (all measures x SDGs) • 3: Dot plot (risk and utility facets) • 4: Parallel coordinates plot • 5: Radial/radar chart • 6: PCA biplot (joint PCA) • 7: Blockwise PCA scatterplot
`...`	Additional arguments passed to plotting functions
`show_pareto`	Logical, whether to highlight Pareto-optimal SDGs. Default TRUE.
`show_labels`	Logical, whether to show SDG labels. Default TRUE.
`col_pareto`	Color for Pareto-optimal points. Default "steelblue".
`col_dominated`	Color for dominated points. Default "gray60".

Details

The visualization approaches are designed to reveal different aspects of the Risk-Utility trade-off:

Plot 1 - Composite Scatterplot: Classic R-U map showing composite risk vs utility scores. Pareto front is highlighted. Good for overall comparison.

Plot 2 - Heatmap: Shows all measures for all SDGs in a matrix format. Useful for identifying patterns across measures.

Plot 3 - Dot Plot: Separates risk and utility measures into facets. Shows individual measure values for detailed comparison.

Plot 4 - Parallel Coordinates: Each SDG is a line crossing all measures. Reveals trade-offs and patterns across dimensions.

Plot 5 - Radial/Radar Chart: Polygonal profile for each SDG. Provides gestalt view of multivariate performance.

Plot 6 - PCA Biplot: Joint PCA of all measures. Shows measure correlations (arrows) and SDG positions simultaneously.

Plot 7 - Blockwise PCA: Separate PCA for risk and utility blocks. X-axis = PC1(utility), Y-axis = PC1(risk).

Value

The `rumap` object `x`, invisibly.

Author(s)

Matthias Templ

See Also

`rumap` for creating rumap objects

Other utility: `chisq_utility()`, `ci_proximity()`, `contingency_fidelity()`, `copula_fidelity()`, `energy_distance()`, `gower()`, `hellinger()`, `mmd()`, `mqs()`, `pMSE()`, `propscore()`, `regression_fidelity()`, `rumap()`, `specks()`, `subgroup_utility()`, `tail_fidelity()`, `tstr()`

`plot.singling_out` *Plot method for singling_out objects*

Description

Plot method for singling_out objects

Usage

```
## S3 method for class 'singling_out'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "singling_out"
y	not used
...	additional arguments passed to plotting functions
which	which plot(s) to show: 1 = Risk comparison barplot (attack vs control), 2 = Match count distribution histogram

Value

No return value; called for the side effect of producing a plot.

---

plot.specks *Plot method for specks objects*

Description

Plot method for specks objects

Usage

```
## S3 method for class 'specks'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "specks"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = density comparison (default), 2 = ECDF comparison, 3 = histogram comparison

Value

No return value; called for the side effect of producing a plot.

plot.subgroup_utility *Plot method for subgroup_utility objects*

Description

Plot method for subgroup_utility objects

Usage

```
## S3 method for class 'subgroup_utility'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "subgroup_utility"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = bar chart of utility scores per subgroup

Value

No return value; called for the side effect of producing a plot.

plot.suda *Plot method for suda objects*

Description

Plot method for suda objects

Usage

```
## S3 method for class 'suda'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "suda"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = SUDA score distribution, 2 = MSU count by size

Value

No return value; called for the side effect of producing a plot.

plot.tail_fidelity *Plot method for tail_fidelity objects*

Description

Plot method for tail_fidelity objects

Usage

```
## S3 method for class 'tail_fidelity'
plot(x, y = NULL, ..., which = 1, max_panels = 9)
```

Arguments

x	an object of class "tail_fidelity"
y	not used
...	additional arguments passed to plotting functions
which	integer, which plot: 1 = tail-focused QQ plot per variable, 2 = per-variable tail divergence lollipop chart
max_panels	integer, maximum number of panels for QQ plots. Default 9.

Value

No return value; called for the side effect of producing a plot.

plot.tcap *Plot method for tcap objects*

Description

Plot method for tcap objects

Usage

```
## S3 method for class 'tcap'
plot(x, y = NULL, ..., which = 1)
```

Arguments

x	an object of class "tcap"
y	not used
...	additional arguments passed to plotting functions
which	which plot(s) to show: 1 = TCAP histogram, 2 = risk categories, 3 = matches distribution

Value

No return value; called for the side effect of producing a plot.

`plot.tcloseness` *Plot method for tcloseness objects*

Description

Plot method for tcloseness objects

Usage

```
## S3 method for class 'tcloseness'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "tcloseness"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	integer, which plot: 1 = EMD distribution histogram, 2 = EC size vs EMD scatter

Value

No return value; called for the side effect of producing a plot.

`plot.weap` *Plot method for weap objects*

Description

Plot method for weap objects

Usage

```
## S3 method for class 'weap'
plot(x, y = NULL, ..., which = 1)
```

Arguments

`x`	an object of class "weap"
`y`	not used
`...`	additional arguments passed to plotting functions
`which`	which plot(s) to show: 1 = WEAP histogram, 2 = equivalence class sizes, 3 = disclosive vs non-disclosive

Value

No return value; called for the side effect of producing a plot.

---

pMSE *Propensity Score Mean Squared Error (pMSE) Utility Measure*

Description

Computes the propensity score mean squared error (pMSE) and related utility statistics for synthetic data. This is a comprehensive implementation of the propensity score approach to measuring synthetic data utility, following Snoke et al. (2018) and Woo et al. (2009).

Usage

```
pMSE(X, ...)

## S3 method for class 'synth_pair'
pMSE(X, ...)

## Default S3 method:
pMSE(
  X,
  Y,
  vars = NULL,
  method = c("logit", "cart", "rf"),
  maxorder = 1,
  tree.method = c("rpart", "ctree"),
  nperms = 20,
  break.vars = NULL,
  ngroups = 5,
  na = c("remove", "indicator", "impute"),
  na.rm = TRUE,
  seed = NULL,
  cp = 1e-08,
  minbucket = 5,
  ...
)
```

Arguments

X	data frame of original data, or a synth_pair object
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data (not needed if X is a synth_pair)
vars	character vector of variable names to use in the propensity model. If NULL, all common variables are used.

`method`	character, propensity score estimation method: • "logit" - logistic regression (default, allows analytical null) • "cart" - CART classification tree (requires rpart) • "rf" - random forest
`maxorder`	integer, maximum order of interactions for logit method (default: 1). Use 0 for main effects only, 1 for two-way interactions, etc.
`tree.method`	character, tree method when method="cart": "rpart" (default) or "ctree"
`nperms`	integer, number of permutations for null distribution when using cart or rf methods (default: 20). Set to 0 to skip permutation test.
`break.vars`	character vector of numeric variables to discretize for modeling. If NULL (default), continuous variables are used as-is for logit/rf, or automatically binned for cart.
`ngroups`	integer, number of groups for discretizing continuous variables (default: 5)
`na`	character, NA handling: "remove" (default), "indicator" (add indicators), or "impute" (impute with median/mode)
`na.rm`	logical, remove records with NA values (default: TRUE). Alternatively, use na="indicator" to add indicator variables for NAs.
`seed`	integer, random seed for reproducibility (default: NULL)
`cp`	numeric, complexity parameter for rpart (default: 1e-8, smaller = more complex tree)
`minbucket`	integer, minimum observations in terminal nodes for rpart (default: 5)

Details

The propensity score approach to utility assessment works by combining original and synthetic data, fitting a model to predict which dataset each record belongs to, and measuring how distinguishable the datasets are.

The pMSE statistic:

$$pMSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - c)^2$$

where $\hat{p}_i$ is the estimated propensity score (probability of being synthetic) and $c = n_Y/(n_X + n_Y)$ is the true proportion of synthetic records.

Null expectation (for logistic regression):

$$E[pMSE] = \frac{(k-1)(1-c)^2 c}{N}$$

where k is the number of parameters and $N = n_X + n_Y$.

Interpretation:

- **pMSE ratio ~ 1**: Correct synthesis - synthetic indistinguishable from original
- **pMSE ratio < 1**: Acceptable (random variation)
- **pMSE ratio > 2**: Poor utility - notable differences

- **pMSE ratio > 5**: Very poor utility - substantial differences
- **S_pMSE ~ 0**: Good utility
- **|S_pMSE| > 2**: Significant difference from null
- **SPECKS < 0.1**: Good utility
- **PO50 ~ 50%**: Good utility (random classification)

Methods:

- **logit**: Analytical null distribution available. Most interpretable. Use maxorder to control model complexity (0=main effects, 1=two-way interactions).
- **cart**: Uses CART trees. Null estimated via permutation. Good for complex relationships but can be unstable.
- **rf**: Random forest. Most flexible but computationally intensive. Null estimated via permutation.

Value

An object of class "pMSE" containing:

- pMSE: the propensity score mean squared error
- pMSE_null: expected pMSE under correct synthesis (NULL hypothesis)
- pMSE_ratio: ratio of observed to expected pMSE (values ~1 are good)
- S_pMSE: standardized pMSE (z-score, values ~0 are good)
- SPECKS: Kolmogorov-Smirnov statistic on propensity scores
- PO50: percent correctly classified at 0.5 threshold (values ~50% are good)
- ks_pvalue: p-value from KS test
- propensity_original: propensity scores for original records
- propensity_synthetic: propensity scores for synthetic records
- c: proportion of synthetic records in combined data
- k: number of parameters in propensity model (for logit)
- method: method used
- utility_interpretation: character, interpretation of results

Author(s)

Matthias Templ

References

Snoke, J., Raab, G.M., Nowok, B., Dibben, C., & Slavkovic, A. (2018). General and specific utility measures for synthetic data. *Journal of the Royal Statistical Society: Series A*, 181(3), 663-688. doi:10.1111/rssa.12358

Woo, M.J., Reiter, J.P., Oganian, A., & Karr, A.F. (2009). Global measures of data utility for microdata masked for disclosure limitation. *Journal of Privacy and Confidentiality*, 1(1), 111-124.

See Also

specks for a simpler SPECKS-focused function, propscore for an alternative implementation

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), gower(), hellinger(), mmd(), mqs(), plot.rumap(), propscore(), regression_fidelity(), rumap(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
# Create example data
set.seed(123)
n <- 500
original <- data.frame(
  age = sample(18:80, n, replace = TRUE),
  gender = factor(sample(c("M", "F"), n, replace = TRUE)),
  region = factor(sample(c("N", "S", "E", "W"), n, replace = TRUE)),
  income = exp(rnorm(n, 10, 1))
)

# Good synthetic data (similar distributions)
synthetic_good <- data.frame(
  age = sample(18:80, n, replace = TRUE),
  gender = factor(sample(c("M", "F"), n, replace = TRUE)),
  region = factor(sample(c("N", "S", "E", "W"), n, replace = TRUE)),
  income = exp(rnorm(n, 10, 1))
)

# Compute pMSE with logistic regression
result <- pMSE(original, synthetic_good, method = "logit", maxorder = 1)
print(result)
summary(result)
plot(result)

# Poor synthetic data
synthetic_poor <- data.frame(
  age = sample(40:60, n, replace = TRUE),
  gender = factor(sample(c("M", "F"), n, replace = TRUE, prob = c(0.9, 0.1))),
 region = factor(sample(c("N", "S", "E", "W"), n, replace = TRUE, prob = c(0.7, 0.1, 0.1, 0.1))),
  income = exp(rnorm(n, 11, 0.5))
)

result_poor <- pMSE(original, synthetic_poor, method = "logit")
print(result_poor)


# Using CART method with permutation test
result_cart <- pMSE(original, synthetic_good, method = "cart", nperms = 10)
print(result_cart)
```

population_uniqueness *Population Uniqueness Risk*

Description

Estimates the probability that records that are unique in the sample are also unique in the population, using super-population models. Three estimation methods are available: Pitman, Zayatz, and Sample-Negative-Binomial (SNB).

Usage

```
population_uniqueness(X, ...)

## S3 method for class 'synth_pair'
population_uniqueness(X, data = c("synthetic", "original"), ...)

## Default S3 method:
population_uniqueness(
  X,
  key_vars,
  sampling_fraction = 0.01,
  method = c("all", "pitman", "zayatz", "snb"),
  na.rm = TRUE,
  ...
)
```

Arguments

X	data frame to assess, or a synth_pair object
...	additional arguments passed to methods (currently unused)
data	character, which dataset to evaluate when using a synth_pair object: "synthetic" (default) or "original". Ignored for the default method.
key_vars	character vector of quasi-identifier variable names
sampling_fraction	numeric, the fraction of the population represented by the sample (default: 0.01)
method	character, one of "pitman", "zayatz", "snb", or "all" (default: "all"). If "all", all three methods are computed and compared.
na.rm	logical, remove records with NA in key variables (default: TRUE)

Details

Population uniqueness estimation asks: given that a record has a unique combination of quasi-identifiers in the sample (f_k = 1), what is the probability that this combination is also unique in the entire population (F_k = 1)?

Three super-population models are supported:

Zayatz (equiprobable form): The simplest model. For each sample unique (f_k = 1):

$$P(F_k = 1 | f_k = 1) = (1 - \pi)^{1/\pi - 1}$$

where π is the sampling fraction. This is the closed form obtained under an equiprobable (Poisson) super-population and assigns a single constant probability to all sample uniques; it is a simplification of Zayatz's (1991) frequency-spectrum estimator. Records with f_k > 1 have risk 0.

Pitman (moment approximation): A simplified estimator inspired by the Pitman partition model, with parameter α estimated by a moment heuristic from the singleton/doubleton frequency counts. For sample uniques:

$$P(F_k = 1 | f_k = 1) \approx \frac{\alpha}{\alpha + n \cdot \pi}$$

where n is the sample size. This is an approximation rather than the full Pitman (2003) maximum-likelihood estimator and should be read as such.

SNB (Bethlehem et al., 1990): Sample-Negative-Binomial model. Fits a negative binomial distribution to the population frequency distribution using method of moments. Parameters r and q are estimated from the sample frequency distribution. For sample uniques:

$$P(F_k = 1 | f_k = 1) = \frac{(1 - \pi q/(1+q))^r}{\text{norm}}$$

Note on estimators: The Zayatz and Pitman options above use simplified closed-form / moment-based approximations rather than the full published estimators; they are intended as fast, comparable heuristics. For authoritative population-uniqueness estimation consider specialised implementations (e.g. the **sdcMicro** package). Results are best compared across methods rather than relied on individually.

Interpretation:

- **Global risk < 0.05**: Low population uniqueness risk
- **Global risk 0.05–0.10**: Moderate risk, caution advised
- **Global risk > 0.10**: Elevated risk, records may be identifiable

Value

An object of class "population_uniqueness" containing:

- risk_per_record: numeric vector of P(F_k=1|f_k=1) per record
- global_risk: mean risk across all records
- n_sample_uniques: count of records with f_k=1
- pct_sample_uniques: percentage of sample uniques
- n_population_uniques_est: estimated number of population uniques
- freq_table: table of QI combination frequencies
- method: method used
- key_vars: key variables used
- sampling_fraction: sampling fraction used
- n_records: number of records assessed
- privacy_pass: logical, global_risk <= 0.1
- comparison: if method="all", list of results per method

Author(s)

Matthias Templ

References

Zayatz, L. (1991). Estimation of the Number of Unique Population Elements Using a Sample. *Proceedings of the Section on Survey Research Methods, American Statistical Association*, 369–373.

Pitman, J. (2003). Poisson-Kingman Partitions. In *Statistics and Science: A Festschrift for Terry Speed*, IMS Lecture Notes Monograph Series, 40, 1–34.

Bethlehem, J.G., Keller, W.J. & Pannekoek, J. (1990). Disclosure Control of Microdata. *Journal of the American Statistical Association*, 85(409), 38–45. doi:10.1080/01621459.1990.10475304

Skinner, C.J. & Shlomo, N. (2008). Assessing Identification Risk in Survey Microdata Using Log-Linear Models. *Journal of the American Statistical Association*, 103(483), 989–1001. doi:10.1198/016214507000001328

Templ, M. (2017). *Statistical Disclosure Control for Microdata: Methods and Applications in R*. Springer. doi:10.1007/9783319502724

See Also

kanonymity for k-anonymity assessment, suda for special uniques detection, individual_risk for individual re-identification risk

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
data <- data.frame(
  age = sample(c("young", "middle", "old"), 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE),
  income = sample(c("low", "medium", "high"), 200, replace = TRUE)
)

# Estimate population uniqueness with all methods
result <- population_uniqueness(data,
                                key_vars = c("age", "gender", "region"),
                                sampling_fraction = 0.01)
print(result)
summary(result)

# Use a specific method
result_z <- population_uniqueness(data,
                                  key_vars = c("age", "gender", "region"),
                                  sampling_fraction = 0.05,
```

```
                                        method = "zayatz")
print(result_z)


# More quasi-identifiers = higher risk of uniqueness
result_more <- population_uniqueness(data,
                                     key_vars = c("age", "gender", "region", "income"),
                                     sampling_fraction = 0.01)
print(result_more)
plot(result_more)
```

`positive_information_disclosure`
Positive Information Disclosure (PID)

Description

Compute the Positive Information Disclosure (PID) metric based on prior and posterior probabilities. This metric measures how much an adversary's posterior probability improves compared to the prior, indicating how much additional information has been gained from an observation.

Usage

```
positive_information_disclosure(prior, posterior)
```

Arguments

`prior`	A named numeric vector of prior probabilities. Names represent possible secrets x^*.
`posterior`	A named numeric vector of posterior probabilities (same names as `prior`).

Details

Positive Information Disclosure (PID) builds on Shannon's perfect secrecy criterion. It quantifies how much an adversary's belief about any secret x^* improves after observing data y. A PID of 0 means no information gain, satisfying perfect secrecy. Larger values indicate greater improvement in belief, which may pose privacy risks.

High PID values suggest privacy leakage—especially if sensitive values receive much higher posterior probabilities.

Value

A single numeric value representing the maximum relative improvement in belief:

$$\max_{x^*} \left(\frac{p(x^*|y) - p(x^*)}{p(x^*)} \right)$$

Author(s)

Matthias Templ

References

Wagner, I., & Eckhoff, D. (2018). Technical Privacy Metrics: A Systematic Survey. ACM Computing Surveys (CSUR), 51(3), Article 57. Section 5.2.11.

See Also

Other information-theory: `Entropy`, `EntropyMeasures`, `information_surprisal()`, `max_info_leakage()`, `mutualInformation()`, `privacy_score()`, `systemAnonymityLevel()`

Examples

```
# Example: Prior vs. posterior beliefs over three secrets
prior <- c("A" = 0.3, "B" = 0.4, "C" = 0.3)
posterior <- c("A" = 0.6, "B" = 0.3, "C" = 0.1)
positive_information_disclosure(prior, posterior)

# Privacy evaluation: Detecting excessive belief improvement for a sensitive attribute
# Suppose after anonymization, the adversary can better guess sensitive conditions
prior <- c("depression" = 0.05, "none" = 0.95)
posterior <- c("depression" = 0.15, "none" = 0.85)
pid <- positive_information_disclosure(prior, posterior)
if (pid > 1) message("Privacy risk detected: PID > 1")
```

`print.attacker_risk` *Print method for attacker_risk objects*

Description

Print method for attacker_risk objects

Usage

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

Arguments

`x`	an object of class "attacker_risk"
`...`	additional arguments (ignored)

Value

The input object, invisibly.

print.chisq_utility *Print method for chisq_utility objects*

Description

Print method for chisq_utility objects

Usage

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

Arguments

x	an object of class "chisq_utility"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.ci_proximity *Print method for ci_proximity objects*

Description

Print method for ci_proximity objects

Usage

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

Arguments

x	an object of class "ci_proximity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.compare_distributions_cont

Print method for compare_distributions_cont objects

Description

Print method for compare_distributions_cont objects

Usage

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

Arguments

x	an object of class "compare_distributions_cont"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.compare_feature_importance

Print method for compare_feature_importance objects

Description

Print method for compare_feature_importance objects

Usage

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

Arguments

x	an object of class "compare_feature_importance"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.contingency_fidelity *Print method for contingency_fidelity objects*

Description

Print method for contingency_fidelity objects

Usage

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

Arguments

x	an object of class "contingency_fidelity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.copula_fidelity *Print method for copula_fidelity objects*

Description

Print method for copula_fidelity objects

Usage

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

Arguments

x	an object of class "copula_fidelity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.dcap *Print method for dcap objects*

Description

Print method for dcap objects

Usage

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

Arguments

x	an object of class "dcap"
...	additional arguments passed to the print method

Value

The input object, invisibly.

print.dcr *Print method for dcr objects*

Description

Print method for dcr objects

Usage

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

Arguments

x	an object of class "dcr"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.delta_presence *Print method for delta_presence objects*

Description

Print method for delta_presence objects

Usage

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

Arguments

x	an object of class "delta_presence"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.denpca *Print method for denpca objects*

Description

Print method for denpca objects

Usage

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

Arguments

x	an object of class "denpca"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.denratio *Print method for denratio objects*

Description

Print method for denratio objects

Usage

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

Arguments

x	an object of class "denratio"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.disclosure_report *Print method for disclosure_report objects*

Description

Print method for disclosure_report objects

Usage

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

Arguments

x	an object of class "disclosure_report"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.disco *Print method for disco objects*

Description

Print method for disco objects

Usage

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

Arguments

x	an object of class "disco"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.domias *Print method for domias objects*

Description

Print method for domias objects

Usage

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

Arguments

x	an object of class "domias"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.drisk *Print method for drisk objects*

Description

Print method for drisk objects

Usage

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

Arguments

x	an object of class "drisk"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.energy_distance *Print method for energy_distance objects*

Description

Print method for energy_distance objects

Usage

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

Arguments

x	an object of class "energy_distance"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.epsilon_identifiability *Print method for epsilon_identifiability objects*

Description

Print method for epsilon_identifiability objects

Usage

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

Arguments

x	an object of class "epsilon_identifiability"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.gower *Print method for gower objects*

Description

Print method for gower objects

Usage

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

Arguments

x	an object of class "gower"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.hellinger *Print method for hellinger objects*

Description

Print method for hellinger objects

Usage

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

Arguments

x	an object of class "hellinger"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.hitting_rate *Print method for hitting_rate objects*

Description

Print method for hitting_rate objects

Usage

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

Arguments

x	an object of class "hitting_rate"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.ims *Print method for ims objects*

Description

Print method for ims objects

Usage

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

Arguments

x	an object of class "ims"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.individual_risk *Print method for individual_risk objects*

Description

Print method for individual_risk objects

Usage

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

Arguments

x	an object of class "individual_risk"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.kanonymity *Print method for kanonymity objects*

Description

Print method for kanonymity objects

Usage

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

Arguments

x	an object of class "kanonymity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.ldiversityRisk *Print method for ldiversityRisk objects*

Description

Print method for ldiversityRisk objects

Usage

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

Arguments

x	an object of class "ldiversityRisk"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.linkability *Print method for linkability objects*

Description

Print method for linkability objects

Usage

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

Arguments

x	an object of class "linkability"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.mia *Print method for mia objects*

Description

Print method for mia objects

Usage

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

Arguments

x	an object of class "mia"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.missingCompare *Print method for missingCompare objects*

Description

Print method for missingCompare objects

Usage

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

Arguments

x	An object of class "missingCompare".
...	Additional arguments (ignored).

Value

The input object, invisibly.

print.mmd *Print method for mmd objects*

Description

Print method for mmd objects

Usage

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

Arguments

x	an object of class "mmd"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.mqs *Print method for mqs objects*

Description

Print method for mqs objects

Usage

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

Arguments

x	an object of class "mqs"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.nnaa *Print method for nnaa objects*

Description

Print method for nnaa objects

Usage

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

Arguments

x	an object of class "nnaa"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.nndr *Print method for nndr objects*

Description

Print method for nndr objects

Usage

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

Arguments

x	an object of class "nndr"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.pMSE *Print method for pMSE objects*

Description

Print method for pMSE objects

Usage

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

Arguments

x	an object of class "pMSE"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.population_uniqueness *Print method for population_uniqueness objects*

Description

Print method for population_uniqueness objects

Usage

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

Arguments

x	an object of class "population_uniqueness"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.propscore *Print method for propscore objects*

Description

Print method for propscore objects

Usage

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

Arguments

x	an object of class "propscore"
...	additional arguments passed to the print method

Value

The input object, invisibly.

print.rapid *Print method for rapid objects*

Description

Print method for rapid objects

Usage

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

Arguments

x	an object of class "rapid"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.recordLinkageRisk *Print method for recordLinkageRisk*

Description

Print method for recordLinkageRisk

Usage

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

Arguments

x	object of class "recordLinkageRisk".
...	ignored.

Value

The input object, invisibly.

print.regression_fidelity *Print method for regression_fidelity objects*

Description

Print method for regression_fidelity objects

Usage

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

Arguments

x	an object of class "regression_fidelity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.rumap *Print method for rumap objects*

Description

Print method for rumap objects

Usage

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

Arguments

x	an object of class "rumap"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.singling_out *Print method for singling_out objects*

Description

Print method for singling_out objects

Usage

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

Arguments

x	an object of class "singling_out"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.specks *Print method for specks objects*

Description

Print method for specks objects

Usage

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

Arguments

x	an object of class "specks"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.subgroup_utility *Print method for subgroup_utility objects*

Description

Print method for subgroup_utility objects

Usage

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

Arguments

x	an object of class "subgroup_utility"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.suda *Print method for suda objects*

Description

Print method for suda objects

Usage

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

Arguments

x	an object of class "suda"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.attacker_risk

Print method for summary.attacker_risk objects

Description

Print method for summary.attacker_risk objects

Usage

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

Arguments

x	an object of class "summary.attacker_risk"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.chisq_utility

Print method for summary.chisq_utility objects

Description

Print method for summary.chisq_utility objects

Usage

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

Arguments

x	an object of class "summary.chisq_utility"
...	additional arguments (ignored)

Value

The input object, invisibly.

`print.summary.ci_proximity` *Print method for summary.ci_proximity objects*

Description

Print method for summary.ci_proximity objects

Usage

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

Arguments

`x`	an object of class "summary.ci_proximity"
`...`	additional arguments (ignored)

Value

The input object, invisibly.

`print.summary.compare_distributions_cont` *Print method for summary.compare_distributions_cont objects*

Description

Print method for summary.compare_distributions_cont objects

Usage

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

Arguments

`x`	an object of class "summary.compare_distributions_cont"
`...`	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.compare_feature_importance

Print method for summary.compare_feature_importance objects

Description

Print method for summary.compare_feature_importance objects

Usage

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

Arguments

x	an object of class "summary.compare_feature_importance"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.contingency_fidelity

Print method for summary.contingency_fidelity objects

Description

Print method for summary.contingency_fidelity objects

Usage

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

Arguments

x	an object of class "summary.contingency_fidelity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.copula_fidelity *Print method for summary.copula_fidelity objects*

Description

Print method for summary.copula_fidelity objects

Usage

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

Arguments

x	an object of class "summary.copula_fidelity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.dcap *Print method for summary.dcap objects*

Description

Print method for summary.dcap objects

Usage

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

Arguments

x	an object of class "summary.dcap"
...	additional arguments passed to the print method

Value

The input object, invisibly.

print.summary.dcr *Print method for summary.dcr objects*

Description

Print method for summary.dcr objects

Usage

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

Arguments

x	an object of class "summary.dcr"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.delta_presence *Print method for summary.delta_presence objects*

Description

Print method for summary.delta_presence objects

Usage

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

Arguments

x	an object of class "summary.delta_presence"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.denpca *Print method for summary.denpca objects*

Description

Print method for summary.denpca objects

Usage

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

Arguments

x	an object of class "summary.denpca"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.denratio *Print method for summary.denratio objects*

Description

Print method for summary.denratio objects

Usage

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

Arguments

x	an object of class "summary.denratio"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.disclosure_report *Print method for summary.disclosure_report objects*

Description

Print method for summary.disclosure_report objects

Usage

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

Arguments

x	an object of class "summary.disclosure_report"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.disco *Print method for summary.disco objects*

Description

Print method for summary.disco objects

Usage

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

Arguments

x	an object of class "summary.disco"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.domias *Print method for summary.domias objects*

Description

Print method for summary.domias objects

Usage

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

Arguments

x	an object of class "summary.domias"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.drisk *Print method for summary.drisk objects*

Description

Print method for summary.drisk objects

Usage

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

Arguments

x	an object of class "summary.drisk"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.energy_distance

Print method for summary.energy_distance objects

Description

Print method for summary.energy_distance objects

Usage

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

Arguments

x	an object of class "summary.energy_distance"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.epsilon_identifiability

Print method for summary.epsilon_identifiability objects

Description

Print method for summary.epsilon_identifiability objects

Usage

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

Arguments

x	an object of class "summary.epsilon_identifiability"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.gower *Print method for summary.gower objects*

Description

Print method for summary.gower objects

Usage

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

Arguments

x	an object of class "summary.gower"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.hellinger *Print method for summary.hellinger objects*

Description

Print method for summary.hellinger objects

Usage

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

Arguments

x	an object of class "summary.hellinger"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.hitting_rate *Print method for summary.hitting_rate objects*

Description

Print method for summary.hitting_rate objects

Usage

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

Arguments

x	an object of class "summary.hitting_rate"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.ims *Print method for summary.ims objects*

Description

Print method for summary.ims objects

Usage

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

Arguments

x	an object of class "summary.ims"
...	additional arguments (ignored)

Value

The input object, invisibly.

`print.summary.individual_risk` *Print method for summary.individual_risk objects*

Description

Print method for summary.individual_risk objects

Usage

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

Arguments

`x`	an object of class "summary.individual_risk"
`...`	additional arguments (ignored)

Value

The input object, invisibly.

`print.summary.kanonymity` *Print method for summary.kanonymity objects*

Description

Print method for summary.kanonymity objects

Usage

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

Arguments

`x`	an object of class "summary.kanonymity"
`...`	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.ldiversityRisk

Print method for summary.ldiversityRisk objects

Description

Print method for summary.ldiversityRisk objects

Usage

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

Arguments

x	an object of class "summary.ldiversityRisk"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.linkability

Print method for summary.linkability objects

Description

Print method for summary.linkability objects

Usage

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

Arguments

x	an object of class "summary.linkability"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.mia *Print method for summary.mia objects*

Description

Print method for summary.mia objects

Usage

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

Arguments

x	an object of class "summary.mia"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.missingCompare *Print method for summary.missingCompare objects*

Description

Print method for summary.missingCompare objects

Usage

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

Arguments

x	An object of class "summary.missingCompare".
...	Additional arguments (ignored).

Value

The input object, invisibly.

print.summary.mmd *Print method for summary.mmd objects*

Description

Print method for summary.mmd objects

Usage

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

Arguments

x	an object of class "summary.mmd"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.mqs *Print method for summary.mqs objects*

Description

Print method for summary.mqs objects

Usage

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

Arguments

x	an object of class "summary.mqs"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.nnaa *Print method for summary.nnaa objects*

Description

Print method for summary.nnaa objects

Usage

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

Arguments

x	an object of class "summary.nnaa"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.nndr *Print method for summary.nndr objects*

Description

Print method for summary.nndr objects

Usage

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

Arguments

x	an object of class "summary.nndr"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.pMSE *Print method for summary.pMSE objects*

Description

Print method for summary.pMSE objects

Usage

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

Arguments

x	an object of class "summary.pMSE"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.population_uniqueness *Print method for summary.population_uniqueness objects*

Description

Print method for summary.population_uniqueness objects

Usage

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

Arguments

x	an object of class "summary.population_uniqueness"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.propscore *Print method for summary.propscore objects*

Description

Print method for summary.propscore objects

Usage

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

Arguments

x	an object of class "summary.propscore"
...	additional arguments passed to the print method

Value

The input object, invisibly.

print.summary.rapid *Print method for summary.rapid objects*

Description

Print method for summary.rapid objects

Usage

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

Arguments

x	an object of class "summary.rapid"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.rapid_cv

Print method for summary.rapid_cv objects

Description

Print method for summary.rapid_cv objects

Usage

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

Arguments

x	an object of class "summary.rapid_cv"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.rapid_test

Print method for summary.rapid_test objects

Description

Print method for summary.rapid_test objects

Usage

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

Arguments

x	an object of class "summary.rapid_test"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.rapid_threshold

Print method for summary.rapid_threshold objects

Description

Print method for summary.rapid_threshold objects

Usage

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

Arguments

x	an object of class "summary.rapid_threshold"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.recordLinkageRisk

Print method for summary.recordLinkageRisk

Description

Print method for summary.recordLinkageRisk

Usage

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

Arguments

x	object of class "summary.recordLinkageRisk".
...	ignored.

Value

The input object, invisibly.

print.summary.regression_fidelity *Print method for summary.regression_fidelity objects*

Description

Print method for summary.regression_fidelity objects

Usage

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

Arguments

x	an object of class "summary.regression_fidelity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.rumap *Print method for summary.rumap objects*

Description

Print method for summary.rumap objects

Usage

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

Arguments

x	an object of class "summary.rumap"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.singling_out *Print method for summary.singling_out objects*

Description

Print method for summary.singling_out objects

Usage

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

Arguments

x	an object of class "summary.singling_out"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.specks *Print method for summary.specks objects*

Description

Print method for summary.specks objects

Usage

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

Arguments

x	an object of class "summary.specks"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.subgroup_utility *Print method for summary.subgroup_utility objects*

Description

Print method for summary.subgroup_utility objects

Usage

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

Arguments

x	an object of class "summary.subgroup_utility"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.suda *Print method for summary.suda objects*

Description

Print method for summary.suda objects

Usage

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

Arguments

x	an object of class "summary.suda"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.synth_pair

Print method for summary.synth_pair objects

Description

Print method for summary.synth_pair objects

Usage

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

Arguments

x	An object of class "summary.synth_pair"
...	Additional arguments (ignored)

Value

The input object, invisibly.

print.summary.tail_fidelity

Print method for summary.tail_fidelity objects

Description

Print method for summary.tail_fidelity objects

Usage

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

Arguments

x	an object of class "summary.tail_fidelity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.tcap *Print method for summary.tcap objects*

Description

Print method for summary.tcap objects

Usage

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

Arguments

x	an object of class "summary.tcap"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.tcloseness *Print method for summary.tcloseness objects*

Description

Print method for summary.tcloseness objects

Usage

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

Arguments

x	an object of class "summary.tcloseness"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.summary.weap *Print method for summary.weap objects*

Description

Print method for summary.weap objects

Usage

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

Arguments

x	an object of class "summary.weap"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.synth_pair *Print method for synth_pair objects*

Description

Print method for synth_pair objects

Usage

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

Arguments

x	An object of class "synth_pair"
...	Additional arguments (ignored)

Value

The input object, invisibly.

print.tail_fidelity *Print method for tail_fidelity objects*

Description

Print method for tail_fidelity objects

Usage

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

Arguments

x	an object of class "tail_fidelity"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.tcap *Print method for tcap objects*

Description

Print method for tcap objects

Usage

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

Arguments

x	an object of class "tcap"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.tcloseness *Print method for tcloseness objects*

Description

Print method for tcloseness objects

Usage

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

Arguments

x	an object of class "tcloseness"
...	additional arguments (ignored)

Value

The input object, invisibly.

print.weap *Print method for weap objects*

Description

Print method for weap objects

Usage

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

Arguments

x	an object of class "weap"
...	additional arguments (ignored)

Value

The input object, invisibly.

`privacy_score` *Privacy Score*

Description

The privacy score quantifies a user's potential privacy risk by considering the sensitivity of disclosed information and its visibility. It is defined as:

$$privPS(u) = \sum_{x^* \in X^*} \omega_{x^*} \cdot Vis(x^*, u)$$

where ω_{x^*} is the sensitivity of item x^* (e.g., how private or revealing it is), and $Vis(x^*, u)$ is the visibility of that item for user u (e.g., the number of people who have access to that item).

Usage

```
privacy_score(sensitivity, visibility, normalize = FALSE)
```

Arguments

`sensitivity`	Named numeric vector of sensitivities for each information item (e.g., c(gender = 0.3, location = 0.8)).
`visibility`	Named numeric vector of visibility values (e.g., number of users who can see each item).
`normalize`	Logical. If TRUE, returns the normalized privacy score (between 0 and 1).

Details

Computes the privacy score of a user based on the sensitivity of information items and their visibility.

The privacy score helps quantify how exposed a user's data is, depending on both what is shared and how widely.

If normalization is enabled, the score is divided by the maximum possible score (sum of sensitivities times max visibility).

Privacy interpretation: A higher privacy score indicates higher exposure and, therefore, higher risk. The score increases with:

- More sensitive items being disclosed
- Items being visible to more people

Value

A single numeric value representing the privacy score for the user.

Author(s)

Matthias Templ

References

Wagner, I., & Eckhoff, D. (2018). Technical Privacy Metrics: A Systematic Survey. *ACM Computing Surveys*, 51(3), 1–38. https://doi.org/10.1145/3168389

See Also

Other information-theory: `Entropy`, `EntropyMeasures`, `information_surprisal()`, `max_info_leakage()`, `mutualInformation()`, `positive_information_disclosure()`, `systemAnonymityLevel()`

Examples

```
# Example 1: Basic usage
sensitivity <- c(gender = 0.2, birthday = 0.8, location = 0.6)
visibility  <- c(gender = 100, birthday = 20, location = 50)
privacy_score(sensitivity, visibility)

# Example 2: Normalized privacy score
privacy_score(sensitivity, visibility, normalize = TRUE)

# Example 3: Privacy evaluation example
# Assume visibility corresponds to number of users
# who can see each attribute (e.g., social network)
psych_profile <- c(stress_score = 0.9, diagnosis = 1.0)
visibility <- c(stress_score = 5, diagnosis = 2)
privacy_score(psych_profile, visibility)
```

---

propscore *Propensity score utility measure*

Description

How well can you predict if an observation is from which data set?

Usage

```
propscore(X, ...)

## S3 method for class 'synth_pair'
propscore(X, form = NULL, ...)

## Default S3 method:
propscore(
  X,
  Y,
  form = NULL,
  method = c("rf", "ranger", "logreg"),
  adjust_size = TRUE,
```

```
  cluster = NULL,
  na = "impute",
  proximity = c("summary", "full", "none"),
  importance = TRUE,
  ...
)
```

Arguments

X	data frame
...	additional arguments passed to methods. For method = "ranger", extra arguments are forwarded to ranger via modifyList.
form	formula. If NULL all variables are used as predictors
Y	data frame with the same structure as X
method	method for propensity score estimation: "rf" (default, uses randomForest), "ranger" (uses ranger with proximity-based structural metrics), or "logreg" (logistic regression).
adjust_size	for very unbalanced sizes of original and synthetic data. Instead of using a constant c, observations are drawn from the smaller data set so that the both data sets has the same size. If a cluster structure is present (e.g. person in housholds), draws are taken from the clusters (e.g. households).
cluster	vector specifiying the cluster structure. Should be NULL if no cluster structure is present in the data.
na	missing value treatment. Either stop, remove or impute (using a kNN from R package VIM).
proximity	character; proximity computation mode for method = "ranger": "summary" (default) returns aggregate within-class and cross-class proximity statistics, "full" also stores the full proximity matrix, "none" skips proximity computation. Ignored for other methods.
importance	logical; whether to compute variable importance for method = "ranger". Default TRUE. Ignored for other methods.

Value

An S3 object of class "propscore" containing:

predictions Data frame of predicted propensity scores for all records.

ps_ratio Propensity score ratio (mean propensity of synthetic / original).

ps_score Propensity score statistic.

cr Classification rate.

mean_ps_x Mean propensity score for original records.

mean_ps_y Mean propensity score for synthetic records.

density_ratio Density ratio of propensity score distributions.

density_ratio_bayes Bayesian density ratio.

kl Kullback-Leibler divergence between propensity distributions.

kl_bayes Bayesian KL divergence.

mean_ratio Mean ratio of propensity densities.

sd_ratio Standard deviation of density ratio.

mean_ratio_bayes Mean Bayesian density ratio.

sd_ratio_bayes Standard deviation of Bayesian density ratio.

points Grid points used for density estimation.

denX Density estimates for original data propensity scores.

denY Density estimates for synthetic data propensity scores.

bayesspace Logical, whether Bayesian space was used.

n_x Number of original records.

n_y Number of synthetic records.

method Method used for propensity score estimation.

oob_error OOB prediction error (ranger method only).

var_importance Named numeric vector of variable importance (ranger method only).

within_orig_prox Mean within-original proximity (ranger with proximity != "none").

within_synth_prox Mean within-synthetic proximity (ranger with proximity != "none").

cross_prox Mean cross-class proximity (ranger with proximity != "none").

structure_ratio Ratio of cross-class to within-class proximity (ranger with proximity != "none").

proximity_matrix Full proximity matrix (ranger with proximity = "full" only).

Author(s)

Matthias Templ

References

Templ, M. Statistical Disclosure Control for Microdata: Methods and Applications in R. *Springer International Publishing*, 287 pages, 2017. ISBN 978-3-319-50272-4. doi:10.1007/978331950272-4

See Also

Other utility: `chisq_utility()`, `ci_proximity()`, `contingency_fidelity()`, `copula_fidelity()`, `energy_distance()`, `gower()`, `hellinger()`, `mmd()`, `mqs()`, `pMSE()`, `plot.rumap()`, `regression_fidelity()`, `rumap()`, `specks()`, `subgroup_utility()`, `tail_fidelity()`, `tstr()`

Examples

```
# Simple example with synthetic data
set.seed(123)
X <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  income = rnorm(100, 50000, 10000)
)
Y <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  income = rnorm(100, 48000, 12000)
)
propscore(X, Y, form = ~ age + gender + income)


# Extended example using simPop and sdcMicro (requires these packages)
if (requireNamespace("simPop", quietly = TRUE) &&
    requireNamespace("sdcMicro", quietly = TRUE)) {
  data(eusilc13puf, package = "simPop")
  eusilc13puf$age <- as.numeric(as.character(eusilc13puf$age))
  keyvars <- c("age", "rb090", "db040", "pl031", "pb220a")
  sdc <- sdcMicro::createSdcObj(eusilc13puf,
                                keyVars = keyvars,
                                numVars = "pgrossIncome",
                                w = "rb050",
                                hhId = "db030")
  sdc <- sdcMicro::localSuppression(sdc)
  sdc <- sdcMicro::microaggregation(sdc)
  eusilc13puf_anon <- sdcMicro::extractManipData(sdc)
  propscore(eusilc13puf, eusilc13puf_anon, na = "remove",
     form = ~ age + rb090 + pl031 + db040 + pb220a + pgrossIncome)
}


# Account for a survey cluster structure (e.g. households) and keep the
# original/synthetic sizes as-is via adjust_size = FALSE
set.seed(123)
Xc <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  income = rnorm(100, 50000, 10000),
  hid = factor(sample(1:25, 100, replace = TRUE))
)
Yc <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  income = rnorm(100, 48000, 12000),
  hid = factor(sample(1:25, 100, replace = TRUE))
)
propscore(Xc, Yc, form = ~ age + gender + income,
          cluster = "hid", adjust_size = FALSE)
```

rapid *RAPID: Risk of Attribute Prediction-Induced Disclosure*

Description

Assesses inferential disclosure risk for a sensitive variable in synthetic data using the RAPID metric. This function trains a predictive model on synthetic data using quasi-identifiers, then evaluates attribute inference risk by scoring the model on the original data.

Usage

```
rapid(X, ...)

## S3 method for class 'synth_pair'
rapid(X, ...)

## Default S3 method:
rapid(
  X,
  Y,
  key_vars,
  target_var,
  model_type = c("rf", "lm", "cart", "gbm", "logit"),
  num_epsilon = 10,
  num_epsilon_type = c("percentage", "absolute"),
  num_error_metric = c("symmetric", "stabilised_relative", "absolute"),
  num_delta = 0.01,
  cat_tau = 1,
  cat_eval_method = c("RCS_conditional", "RCS_marginal", "NCE"),
  na_strategy = c("constant", "drop", "median"),
  na_constant_value = 0,
  return_all_records = TRUE,
  store_model = FALSE,
  seed = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

X	data frame of original data, or a synth_pair object
...	additional arguments passed to model fitting functions
Y	data frame of synthetic data (not needed if X is a synth_pair)
key_vars	character vector of quasi-identifier variable names
target_var	character, name of the sensitive target variable

`model_type` character, model type for inference: "rf" (random forest, default), "lm" (linear model), "cart" (decision tree), "gbm" (gradient boosting), or "logit" (logistic regression, binary only)

`num_epsilon` numeric threshold for continuous attributes. For percentage-based metrics (default), specify as percentage (e.g., 5 for 5\ specify in units of the sensitive attribute.

`num_epsilon_type`
character, threshold type: "percentage" (default) or "absolute"

`num_error_metric`
character, error metric for continuous variables: "symmetric" (default, recommended), "stabilised_relative", or "absolute"

`num_delta` numeric, smoothing constant for percentage-based metrics (default: 0.01)

`cat_tau` numeric threshold for categorical risk. Interpretation depends on method:

- For RCS_conditional: ratio threshold (default 1, typically 1–1.25)
- For RCS_marginal: normalized gain threshold (typically 0.3)
- For NCE: risk score threshold (typically 0.5–0.7)

`cat_eval_method`
character, method for categorical evaluation: "RCS_conditional" (default), "RCS_marginal", or "NCE"

`na_strategy` character, strategy for NA values in sensitive attribute: "constant" (default), "drop", or "median"

`na_constant_value`
numeric, value for constant NA imputation (default: 0)

`return_all_records`
logical, if TRUE (default) return all records with risk status; if FALSE return only at-risk records

`store_model` logical, if TRUE store the fitted model object in the result (needed for `plot(result, which = 4)` variable importance plot). Default FALSE.

`seed` integer, random seed for reproducibility (default: NULL)

`verbose` logical, print diagnostic messages (default: FALSE)

Details

RAPID (Risk of Attribute Prediction-Induced Disclosure) measures disclosure risk by training a predictive model on synthetic data and evaluating how well it can predict the sensitive attribute in the original data. High prediction accuracy indicates potential disclosure risk.

Unlike CAP-based methods (DCAP, TCAP) which use exact or fuzzy matching, RAPID uses machine learning models to capture more complex relationships between quasi-identifiers and sensitive attributes.

For categorical sensitive variables: Three evaluation methods are available:

- `RCS_conditional` (default): Measures if an observation is an outlier within its class using class-conditional baseline.
- `RCS_marginal`: Measures if the attribute can be inferred better than the marginal baseline rate. Uses normalized gain.

- `NCE`: Normalized Cross-Entropy, measures information leakage.

For continuous sensitive variables: Risk is measured by the proportion of records for which prediction errors fall below a threshold. Three error metrics are available:

- `symmetric` (recommended): Symmetric percentage error, treats predicted and true values equally.
- `stabilised_relative`: Stabilised relative error.
- `absolute`: Raw absolute error (use with num_epsilon_type = "absolute").

Value

An object of class "rapid" containing:

- rapid: the confidence rate (proportion of records at risk)
- n_at_risk: number of records at risk
- pct_at_risk: percentage of records at risk
- method: evaluation method used (categorical) or "numeric"
- threshold: threshold used (cat_tau or num_epsilon)
- model_type: model type used
- model_metrics: model performance (accuracy for categorical, MAE/RMSE for numeric)
- model: the fitted model object (only if store_model=TRUE, otherwise NULL)
- formula: the model formula used
- records: data frame of at-risk records (or all records if return_all_records=TRUE)
- key_vars, target_var: input parameters
- call: the function call

Threshold Selection Guidelines

Categorical sensitive variables:

- RCS_conditional: `cat_tau = 1` (default)
- RCS_marginal: `cat_tau = 0.3`
- NCE: `cat_tau = 0.5-0.7`

Continuous sensitive variables:

- symmetric: `num_epsilon = 5-10` (percentage)
- absolute: `num_epsilon` depends on domain/scale

Comparison with CAP Methods

- **RAPID** captures complex non-linear relationships via ML models
- **DCAP/TCAP** use exact/fuzzy matching on key combinations
- Use RAPID when relationships are complex; use CAP when matching is intuitive
- Both approaches are complementary and should ideally be used together

Author(s)

Oscar Thees, Matthias Templ

References

Templ, M. (2024). Beyond the Trade-off Curve: Multivariate Risk-Utility Visualization for Synthetic Data. *Journal of Official Statistics*.

See Also

dcap, tcap for attribution-based metrics, disclosure_report for comprehensive risk assessment

Other rapid: confint.rapid(), rapid_synthesizer_cv(), rapid_test(), rapid_threshold_select()

Examples

```
# Create example data
set.seed(123)
n <- 200
original <- data.frame(
  age = sample(20:70, n, replace = TRUE),
  gender = factor(sample(c("M", "F"), n, replace = TRUE)),
  region = factor(sample(c("N", "S", "E", "W"), n, replace = TRUE)),
  income = round(rnorm(n, 50000, 10000))
)

# Synthetic version with some noise
synthetic <- data.frame(
  age = original$age + sample(-3:3, n, replace = TRUE),
  gender = factor(sample(c("M", "F"), n, replace = TRUE)),
  region = factor(sample(c("N", "S", "E", "W"), n, replace = TRUE)),
  income = round(original$income * runif(n, 0.9, 1.1))
)

# RAPID for continuous sensitive variable

result <- rapid(
  X = original,
  Y = synthetic,
  key_vars = c("age", "gender", "region"),
  target_var = "income",
  model_type = "rf",
  num_epsilon = 10,
  verbose = TRUE
)
print(result)
summary(result)


# RAPID for categorical sensitive variable

original$health <- factor(sample(c("Good", "Fair", "Poor"), n,
```

```
                                    replace = TRUE, prob = c(0.6, 0.3, 0.1)))
synthetic$health <- factor(sample(c("Good", "Fair", "Poor"), n,
                                    replace = TRUE, prob = c(0.55, 0.35, 0.1)))

result_cat <- rapid(
  X = original,
  Y = synthetic,
  key_vars = c("age", "gender", "region"),
  target_var = "health",
  model_type = "rf",
  cat_tau = 1,
  cat_eval_method = "RCS_conditional"
)
print(result_cat)


# ================================================================
# Real data example using SD2011 from synthpop package
# ================================================================

if (requireNamespace("synthpop", quietly = TRUE) &&
    requireNamespace("ranger", quietly = TRUE)) {

  # Load SD2011 Polish Social Diagnosis survey data
  data(SD2011, package = "synthpop")

  # Select relevant variables and remove NAs for cleaner example
  vars <- c("sex", "age", "edu", "marital", "region", "income", "smoke")
  original_sd <- SD2011[, vars]
  original_sd <- original_sd[complete.cases(original_sd), ]

  # Generate synthetic data using synthpop
  synth_obj <- synthpop::syn(original_sd, seed = 123, print.flag = FALSE)
  synthetic_sd <- synth_obj$syn

  # ----------------------------------------------------------------
  # Example 1: Continuous sensitive variable (income)
  # ----------------------------------------------------------------
  # Assess risk of inferring income from demographic quasi-identifiers

  rapid_income <- rapid(
    X = original_sd,
    Y = synthetic_sd,
    key_vars = c("sex", "age", "edu", "marital", "region"),
    target_var = "income",
    model_type = "rf",
    num_epsilon = 10,           # 10% error threshold
    num_epsilon_type = "percentage",
    num_error_metric = "symmetric",
    verbose = TRUE
  )

  print(rapid_income)
```

```
summary(rapid_income)
plot(rapid_income, which = 1:2)

# Interpretation:
# - RAPID score close to 0: low risk (model cannot predict income well)
# - RAPID score > 0.15: elevated risk (sensitive attribute predictable)

# ----------------------------------------------------------------------
# Example 2: Categorical sensitive variable (smoking status)
# ----------------------------------------------------------------------
# Assess risk of inferring smoking behavior from demographics

rapid_smoke <- rapid(
  X = original_sd,
  Y = synthetic_sd,
  key_vars = c("sex", "age", "edu", "marital", "income"),
  target_var = "smoke",
  model_type = "rf",
  cat_tau = 1,                # ratio threshold for RCS_conditional
  cat_eval_method = "RCS_conditional",
  verbose = TRUE
)

print(rapid_smoke)
summary(rapid_smoke)
plot(rapid_smoke)

# Compare with DCAP for the same data
dcap_smoke <- dcap(
  X = original_sd,
  Y = synthetic_sd,
  key_vars = c("sex", "age", "edu", "marital"),
  target_var = "smoke"
)
print(dcap_smoke)

# RAPID may detect risks that DCAP misses when relationships are non-linear
}
```

`rapid_synthesizer_cv` *Cross-Validation of Synthesizer Disclosure Risk*

Description

Evaluates the RAPID disclosure risk of a synthesis method using k-fold cross-validation. In each fold the synthesizer is trained on the training split and RAPID is evaluated on the held-out test split, giving a distribution of risk scores that accounts for variability in both the synthesis and the evaluation.

Usage

```
rapid_synthesizer_cv(
  original_data,
  synthesizer = NULL,
  quasi_identifiers,
  sensitive_attribute,
  k = 5,
  stratified = TRUE,
  return_details = FALSE,
  return_all_records = FALSE,
  seed = 2025,
  verbose = TRUE,
  ...
)

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

Arguments

original_data	A data frame of original (confidential) records.
synthesizer	A function that takes a data frame (and optionally a seed argument) and returns a synthetic data frame. If NULL, synthpop::syn() is used as a default.
quasi_identifiers	Character vector of quasi-identifier column names.
sensitive_attribute	Character string naming the sensitive (target) variable.
k	Number of folds (default 5).
stratified	Logical; use stratified folds for categorical targets? Requires the **caret** package (default TRUE).
return_details	Logical; include per-fold metrics in the result?
return_all_records	Logical; include per-record risk data from every fold?
seed	Optional random seed.
verbose	Logical; print progress messages?
...	additional arguments (ignored)
x	an object of class "rapid_cv"

Value

An object of class "rapid_cv" with components:

cv_summary List with mean, sd, se, median, min, max, ci_lower, ci_upper of the fold-level RAPID scores, plus aggregated model metrics.

cv_details Data frame of per-fold results (if return_details = TRUE).

fold_data Concatenated per-record risk data across folds (if `return_all_records = TRUE`).

settings List of CV settings (`k`, `stratified`, etc.).

eval_method Evaluation method used.

model_type Model type used.

threshold Threshold value used.

Author(s)

Oscar Thees, Matthias Templ

References

Thees, O., Mueller, N., & Templ, M. (2026). Beyond the Trade-off Curve: Multivariate and Advanced Risk-Utility Maps for Evaluating Anonymized and Synthetic Data. *Journal of Official Statistics*.

See Also

`rapid`, `rapid_test`

Other rapid: `confint.rapid()`, `rapid()`, `rapid_test()`, `rapid_threshold_select()`

Examples

```
# Requires the 'synthpop' package for the default-style synthesizer
if (requireNamespace("synthpop", quietly = TRUE)) {
  df <- data.frame(
    age = sample(20:80, 200, replace = TRUE),
    sex = factor(sample(c("m", "f"), 200, replace = TRUE)),
    region = factor(sample(LETTERS[1:4], 200, replace = TRUE)),
    income = rnorm(200, 50000, 10000)
  )
  cv <- rapid_synthesizer_cv(
    original_data = df,
    synthesizer = function(data, seed = NULL) {
      synthpop::syn(data, m = 1, seed = seed, print.flag = FALSE)$syn
    },
    quasi_identifiers = c("age", "sex", "region"),
    sensitive_attribute = "income",
    model_type = "rf", k = 5
  )
  print(cv)
}
```

rapid_test *Permutation Test for RAPID Disclosure Risk*

Description

Tests whether the observed RAPID score is significantly greater than expected under the null hypothesis that the sensitive attribute is independent of the quasi-identifiers. The model is fitted once on the synthetic data and predictions are computed once on the original quasi-identifiers; each permutation only re-evaluates the risk score against shuffled labels, making the test computationally efficient.

Usage

```
rapid_test(
  original_data,
  synthetic_data,
  quasi_identifiers,
  sensitive_attribute,
  model_type = "rf",
  cat_tau = 0.3,
  num_epsilon = 10,
  cat_eval_method = "RCS_marginal",
  num_error_metric = "symmetric",
  num_epsilon_type = "percentage",
  num_delta = 0.01,
  n_permutations = 199,
  alpha = 0.05,
  seed = NULL,
  verbose = FALSE,
  ...
)

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

Arguments

original_data	A data frame of original (confidential) records.
synthetic_data	A data frame of synthetic records used to train the predictive model.
quasi_identifiers	Character vector of column names used as predictors.
sensitive_attribute	Character string naming the sensitive (target) variable.
model_type	Model to train on synthetic data. One of "rf" (default), "lm", "cart", "gbm", or "logit".
cat_tau	Categorical risk threshold; see rapid.

num_epsilon	Numeric error threshold; see rapid.
cat_eval_method	Categorical evaluation method; see rapid.
num_error_metric	Numeric error metric; see rapid.
num_epsilon_type	Numeric threshold type ("percentage" or "absolute"); see rapid.
num_delta	Smoothing constant for percentage metrics (default 0.01).
n_permutations	Number of permutations (default 199).
alpha	Significance level (default 0.05).
seed	Optional random seed.
verbose	Logical; print progress?
...	additional arguments (ignored)
x	an object of class "rapid_test"

Value

An object of class "rapid_test" with components:

statistic Observed RAPID score.

p_value Permutation p-value.

null_distribution Numeric vector of permutation RAPID values.

n_permutations Number of permutations used.

alpha Significance level.

significant Logical; TRUE if p_value <= alpha.

null_mean Mean of the null distribution.

null_sd Standard deviation of the null distribution.

null_quantiles Named vector of null quantiles (5th, 50th, 95th percentile).

observed_rapid The full rapid object from the unpermuted evaluation.

method "permutation".

Author(s)

Oscar Thees, Matthias Templ

References

Thees, O., Mueller, N., & Templ, M. (2026). Beyond the Trade-off Curve: Multivariate and Advanced Risk-Utility Maps for Evaluating Anonymized and Synthetic Data. *Journal of Official Statistics*.

See Also

rapid, confint, rapid_threshold_select

Other rapid: confint.rapid(), rapid(), rapid_synthesizer_cv(), rapid_threshold_select()

Examples

```
# Small runnable example with few permutations
set.seed(42)
X <- data.frame(
  age = sample(20:60, 80, replace = TRUE),
  sex = sample(c("M", "F"), 80, replace = TRUE),
  income = rnorm(80, 50000, 10000)
)
Y <- X
Y$income <- Y$income + rnorm(80, 0, 5000)
res <- rapid_test(X, Y,
                  quasi_identifiers = c("age", "sex"),
                  sensitive_attribute = "income",
                  model_type = "lm", n_permutations = 9)
print(res)


# With more permutations and random forest
res2 <- rapid_test(X, Y,
                   quasi_identifiers = c("age", "sex"),
                   sensitive_attribute = "income",
                   model_type = "rf", n_permutations = 199)
print(res2)
```

`rapid_threshold_select` *Data-Driven Threshold Selection for RAPID*

Description

Selects the largest threshold at which the observed RAPID score is still significantly greater than the permutation null, following the approach described in the RAPID inferential framework. The recommended threshold τ^* (or ε^*) is the maximum threshold where `RAPID_obs > Q_{1-alpha}(null)`.

Usage

```
rapid_threshold_select(
  original_data,
  synthetic_data,
  quasi_identifiers,
  sensitive_attribute,
  model_type = "rf",
  cat_eval_method = "RCS_marginal",
  num_error_metric = "symmetric",
  num_epsilon_type = "percentage",
  num_delta = 0.01,
```

```
  tau_range = seq(0.05, 0.95, by = 0.05),
  epsilon_range = seq(1, 50, by = 2),
  n_permutations = 199,
  alpha = 0.05,
  seed = NULL,
  verbose = FALSE,
  ...
)

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

## S3 method for class 'rapid_threshold'
plot(x, ...)
```

Arguments

original_data	A data frame of original (confidential) records.
synthetic_data	A data frame of synthetic records used to train the predictive model.
quasi_identifiers	Character vector of column names used as predictors.
sensitive_attribute	Character string naming the sensitive (target) variable.
model_type	Model to train on synthetic data. One of "rf" (default), "lm", "cart", "gbm", or "logit".
cat_eval_method	Categorical evaluation method; see rapid.
num_error_metric	Numeric error metric; see rapid.
num_epsilon_type	Numeric threshold type ("percentage" or "absolute"); see rapid.
num_delta	Smoothing constant for percentage metrics (default 0.01).
tau_range	Numeric vector of categorical thresholds to evaluate (used when the sensitive attribute is categorical).
epsilon_range	Numeric vector of numeric thresholds to evaluate (used when the sensitive attribute is continuous).
n_permutations	Number of permutations (default 199).
alpha	Significance level (default 0.05).
seed	Optional random seed.
verbose	Logical; print progress?
...	additional arguments (ignored)
x	an object of class "rapid_threshold"

Value

An object of class "rapid_threshold" with components:

threshold_star Recommended threshold, or NA if no threshold is significant.

results Data frame with columns threshold, rapid_obs, null_quantile, and significant.

alpha Significance level used.

is_categorical Logical; type of sensitive attribute.

observed_rapid Full rapid object at the recommended threshold.

Author(s)

Oscar Thees, Matthias Templ

References

Thees, O., Mueller, N., & Templ, M. (2026). Beyond the Trade-off Curve: Multivariate and Advanced Risk-Utility Maps for Evaluating Anonymized and Synthetic Data. *Journal of Official Statistics*.

See Also

rapid, rapid_test

Other rapid: confint.rapid(), rapid(), rapid_synthesizer_cv(), rapid_test()

Examples

```
# Small runnable example
set.seed(42)
X <- data.frame(
  age = sample(20:60, 80, replace = TRUE),
  sex = sample(c("M", "F"), 80, replace = TRUE),
  income = rnorm(80, 50000, 10000)
)
Y <- X
Y$income <- Y$income + rnorm(80, 0, 5000)
sel <- rapid_threshold_select(X, Y,
  quasi_identifiers = c("age", "sex"),
  sensitive_attribute = "income",
  model_type = "lm",
  epsilon_range = seq(5, 30, by = 5),
  n_permutations = 9)
print(sel)


# With random forest and finer grid
sel2 <- rapid_threshold_select(X, Y,
  quasi_identifiers = c("age", "sex"),
  sensitive_attribute = "income",
  model_type = "rf", n_permutations = 199)
print(sel2)
```

```
plot(sel2)
```

---

recordLinkage *Record Linkage Risk After Perturbation*

Description

Measures targeted re-identification risk by linking each original record to the most similar record(s) in a perturbed dataset using quasi-identifiers. Supports deterministic (distance-based), probabilistic (Fellegi-Sunter), PRAM (transition-matrix), and predictive (propensity-score-based) linkage methods.

Usage

```
recordLinkage(X, ...)

## S3 method for class 'synth_pair'
recordLinkage(X, ...)

## Default S3 method:
recordLinkage(
  X,
  x_anon,
  key,
 method = c("deterministic", "probabilistic", "pram", "predictive", "rf", "rbrl",
    "mahalanobis", "embedding"),
  direction = c("forward", "reverse"),
  risk_weighting = c("uniform", "softmax", "kernel"),
  kappa = NULL,
  bandwidth = NULL,
  kernel = c("gaussian", "epanechnikov", "tricube"),
  truth = c("row", "id"),
  id = NULL,
  type = NULL,
  weights = NULL,
  na_anon = c("ignore", "match", "mismatch"),
 strategy = c("nearest", "threshold", "topk", "topk_threshold", "nearest_threshold"),
  k = NULL,
  threshold = NULL,
  block = NULL,
  m_probs = NULL,
  u_probs = NULL,
  fs_threshold = NULL,
  pram_matrix = NULL,
  pred_model = c("logit", "rf"),
```

```
  pred_se = TRUE,
  return_matches = FALSE,
  matching = c("independent", "bijective", "ot"),
  risk_threshold = 0.1,
  n_trees = 500L,
  rf_global = FALSE,
  robust = TRUE,
  ot_epsilon = NULL,
  ot_max_iter = 100L,
  emb_latent_dim = NULL,
  emb_epochs = 50L,
  emb_global = FALSE,
  compute_baseline = FALSE,
  expected_risk = FALSE,
  Y = NULL,
  key_vars = NULL,
  ...
)
```

Arguments

X	data.frame or synth_pair object. Original microdata.
...	additional arguments passed to methods.
x_anon	data.frame. Perturbed/anonymized microdata.
key	character. Names of quasi-identifier variables used for linkage.
method	character. Linkage method: "deterministic" (default), "probabilistic" (Fellegi-Sunter), "pram" (transition matrix), "predictive" (propensity-score-based), "rf" (random forest proximity-based; requires **ranger**), "rbrl" (rank-based record linkage), "mahalanobis" (Mahalanobis distance with robust covariance), or "embedding" (autoencoder latent-space distance; requires **torch**).
direction	character. Direction of the linkage attack: "forward" (default) loops over original records and searches in the anonymized data, answering "how safe is each original individual?"; "reverse" loops over anonymized records and searches in the original data, answering "how disclosive is each released record?".
risk_weighting	character. How to weight candidates: "uniform" (default, 1/lsetl), "softmax" (exponential distance-weighting), or "kernel" (kernel-weighted donor probabilities).
kappa	numeric or NULL. Temperature parameter for softmax weighting. If NULL (default), auto-calibrated as 2 / range(distances).
bandwidth	numeric or NULL. Bandwidth for kernel weighting. If NULL (default), auto-selected via Silverman's rule on the candidate distances.
kernel	character. Kernel function for kernel weighting: "gaussian" (default), "epanechnikov", or "tricube".
truth	character. How to define the true match for scoring: one of "row" (default) or "id". "row" requires equal row counts and assumes row i in X corresponds to row i in x_anon. When datasets have unequal sizes, use truth = "id" with a shared identifier column.

`id` character or NULL. If `truth="id"`, name of an identifier column present in both `X` and `x_anon` that uniquely defines the true match.

`type` named character vector or NULL. Optional per-key type override: values in `c("numeric","ordinal","nominal")`. If NULL, inferred from `X`.

`weights` named numeric vector or NULL. Optional nonnegative weights for keys. If NULL, equal weights are used.

`na_anon` character. How a missing quasi-identifier value is handled during matching, applied consistently across all methods (deterministic, probabilistic, pram, rbrl, mahalanobis, and the embedding deterministic fallback): `"ignore"` (default), `"match"`, or `"mismatch"`.

- `"ignore"` drops the variable from that pairwise comparison (no contribution to the distance, likelihood ratio, or transition factor; for PRAM this multiplies the factor by 1 instead of collapsing the candidate to zero probability).
- `"match"` treats the missing value as agreement (distance contribution 0; for PRAM, the diagonal transition probability).
- `"mismatch"` treats the missing value as disagreement (distance contribution 1; for PRAM, the candidate is excluded).

For `method = "embedding"` the torch encoder handles NA internally (entity-embedding UNK / numeric pass-through); `na_anon` applies only to its deterministic fallback. The numeric part of `"mahalanobis"` uses a zero-contribution approximation for `"ignore"` (see Details).

`strategy` character. Adversary strategy variant: `"nearest"`, `"threshold"`, `"topk"`, `"topk_threshold"`, or `"nearest_threshold"`.

`k` integer or NULL. Used when `strategy` is `"topk"` or `"topk_threshold"`. Ties at the cut-off distance may yield more than `k` candidates.

`threshold` numeric, list, or NULL. If numeric, absolute distance cutoff in $[0, 1]$. If list, supports `list(type="quantile", p=...)` where `p` is in (0,1). The quantile is computed per record within its candidate set (and within-block if blocking is used).

`block` character or NULL. Optional subset of `key` used for exact blocking. Distances are computed only within blocks defined by these variables.

`m_probs` named numeric vector or NULL. User-supplied m-probabilities for the probabilistic method (probability of agreement given true match).

`u_probs` named numeric vector or NULL. User-supplied u-probabilities for the probabilistic method (probability of agreement given non-match).

`fs_threshold` numeric or NULL. Log-likelihood ratio threshold for the probabilistic method. Records with LR below this are excluded. If NULL, defaults to 0 (include all records with positive evidence of match).

`pram_matrix` named list of matrices. Transition matrices for the PRAM method. Each element is a square matrix where entry (i,j) is P(output=j | input=i). Names must correspond to variables in `key`.

`pred_model` character. Model for the predictive method: `"logit"` (default, logistic regression) or `"rf"` (random forest).

pred_se — logical. For the predictive method with pred_model="logit", use the Fisher-information prediction SE as kernel bandwidth (default TRUE). Ignored for "rf" or when bandwidth is user-supplied.

return_matches — logical. If TRUE, returns candidate indices per record (may be memory-heavy).

matching — character. Matching mode: "independent" (default) scores each record independently (classical DBRL), "bijective" enforces one-to-one assignment via the Hungarian algorithm (GDBRL), and "ot" uses entropy-regularized optimal transport (Sinkhorn) for soft global assignment producing continuous risk. Bijective matching requires the **clue** package. See Herranz, Nin, Rodriguez & Tassa (2016).

risk_threshold — numeric. Threshold for classifying records as "high risk" and for the privacy_pass flag (default 0.1). Records with risk above this threshold are counted in n_high_risk and pct_high_risk. The privacy_pass flag is TRUE when mean_risk <= risk_threshold.

n_trees — integer. Number of trees for method = "rf" (default 500). Must be at least 10.

rf_global — logical. For method = "rf" with blocking: if TRUE, train a single global forest on all data and restrict proximity computation within blocks. If FALSE (default), train a separate forest per block (with automatic global fallback for blocks that are too small).

robust — logical. For method = "mahalanobis": if TRUE (default), use the Minimum Covariance Determinant (MCD) estimator via robustbase::covMcd() for robust covariance estimation, with automatic fallback to MASS::cov.rob() or classical cov() if MCD fails. If FALSE, use classical covariance. Ignored for other methods.

ot_epsilon — numeric or NULL. Regularization strength for OT matching. Smaller values produce sharper (more bijective-like) assignments; larger values produce smoother (more uniform) risk. If NULL (default), auto-calibrated as median(C[C > 0]) * 0.05. Ignored unless matching = "ot".

ot_max_iter — integer. Maximum Sinkhorn iterations for OT matching (default 100). Ignored unless matching = "ot".

emb_latent_dim — integer or NULL. For method = "embedding": bottleneck dimension. If NULL (default), auto-computed as max(2, floor(input_dim / 3)) where input_dim includes entity embedding dimensions for categoricals.

emb_epochs — integer. For method = "embedding": maximum training epochs (default 50). Early stopping with patience 5 may terminate training earlier.

emb_global — logical. For method = "embedding" with blocking: if TRUE, train a single autoencoder on all query data and restrict distance computation within blocks. If FALSE (default), train a separate autoencoder per block (with deterministic fallback for blocks smaller than max(30, 5 * latent_dim)).

compute_baseline — logical. If TRUE, also compute the re-identification risk with NO perturbation by linking X against itself (forward, truth = "row") under the same method and settings, returned in $baseline (including risk_reduction). This gives a reference point for interpreting the perturbed risk. Doubles the computation. Default FALSE.

`expected_risk` logical. For `method = "pram"` (forward direction) only: if `TRUE`, also compute a perturbation-aware *expected* risk that, holding all other records at their realized values, integrates each record's risk over its own PRAM transition distribution $P(x_i \to \cdot)$ (assuming per-variable independence). Returned in `$pram_info$expected_risk`. Default `FALSE`.

`Y` Alias for `x_anon`, for naming consistency with the rest of the package. If `x_anon` is not supplied but `Y` is, `Y` is used.

`key_vars` Alias for `key`, matching the name used by `synth_pair` and `rumap`. If `key` is not supplied but `key_vars` is, `key_vars` is used.

Value

An object of class `"recordLinkageRisk"`: a list with components

per_record data.frame with `n_query` rows and columns: `risk` (numeric, re-identification risk), `cand_n` (integer, candidate set size), `true_in_set` (logical), `d_true` (numeric, distance/score to true match), `d_min` (numeric, minimum distance), `d_rank` (integer, rank of true match among candidates), `risk_band` (factor with levels `"very_low"`, `"low"`, `"moderate"`, `"high"`, `"very_high"`, `"unique_match"`). When `matching = "bijective"`, an additional column `bijective_assigned` gives the search-side row index assigned by the Hungarian algorithm. In bijective mode, `risk` and `bijective_assigned` reflect the global one-to-one assignment, while `d_true`, `d_min`, and `d_rank` retain their per-record independent-scoring values from the main loop (useful for diagnostics). When `matching = "ot"`, `risk` is the continuous transport-weighted risk from the Sinkhorn plan and `d_rank` is the rank of the true match by transport weight. In OT mode, `cand_n`, `true_in_set`, `d_true`, and `d_min` retain their per-record independent-scoring values from the main loop (useful for diagnostics). When `return_matches = TRUE` with OT, the returned `matches` also reflect independent scoring; use `transport_plans` for the OT assignment. When `method` is `"rf"` or `"embedding"`, an additional column `nn_similarity` gives the similarity to the nearest released record (the pre-recast `risk` definition), retained as a diagnostic alongside the true-match `risk`.

overall list with aggregate statistics including `risk_gini` (Gini coefficient of risk concentration).

var_importance named numeric vector of per-variable importance.

direction character, `"forward"` or `"reverse"`.

matches (optional) list of integer vectors with candidate indices.

transport_plans (only when `matching = "ot"`) list of transport plan matrices per block. Each matrix has dimensions n_query_in_block x n_search_in_block, with rows summing to `1/n_query` and columns to `1/n_search`.

baseline (only when `compute_baseline = TRUE`) list with `mean_risk`, `max_risk`, `per_record_risk`, `pct_high_risk`, and `risk_reduction` (baseline minus perturbed mean risk) for the no-perturbation reference.

pram_info (only when `method = "pram"`) list with `variables_used` and `mean_transition_prob`; when `expected_risk = TRUE` it also carries the per-record `expected_risk` and `expected_mean_risk`.

When `direction = "forward"`, `per_record` has one row per original record; when `"reverse"`, one row per anonymized record.

Use `top_at_risk`, `risk_by_group`, `merge_per_record`, and `inspect_record` for post-hoc per-record analysis.

Attacker scenario

Targeted record linkage with membership knowledge and exact quasi-identifier knowledge:

- **Membership knowledge:** the attacker knows the target individual has a record in the released dataset.
- **Exact QI knowledge:** the attacker knows the target's quasi-identifiers exactly for the variables specified in `key`.
- **Linking rule:** the attacker applies a strategy (see `strategy`) based on distances, probabilistic scores, or transition probabilities.

Ground truth for scoring is taken either from row alignment (`truth="row"`) or a shared identifier column (`truth="id"`). Note that `truth` affects only the *evaluation* of success; it is not an attacker capability.

Distance measure

For the deterministic method, record linkage is based on a weighted Gower-type distance over the quasi-identifiers. For an original record i and a candidate record j in the perturbed data, the distance is

$$d(i,j) = \frac{\sum_{v \in \mathcal{V}_{ij}} w_v \, \delta_v(i,j)}{\sum_{v \in \mathcal{V}_{ij}} w_v}, \qquad 0 \le d(i,j) \le 1,$$

where $w_v \ge 0$ are variable weights and $\mathcal{V}_{ij}$ denotes the subset of variables contributing to the comparison (depending on missing-value handling).

Variable-specific dissimilarities are defined as:

- Numeric / ordinal variables:

$$\delta_v(i,j) = \frac{|x_{iv} - x_{jv}^{anon}|}{\max(x_v) - \min(x_v)}$$

  (absolute difference scaled to the unit interval).

- Nominal variables:

$$\delta_v(i,j) = \begin{cases} 0, & \text{if } x_{iv} = x_{jv}^{anon}, \\ 1, & \text{otherwise.} \end{cases}$$

The resulting distance is bounded in $[0,1]$, where 0 indicates complete agreement and 1 maximal disagreement on the contributing quasi-identifiers. If `na_anon = "ignore"`, variables with missing values in either record are excluded from the numerator and denominator for that pairwise comparison.

Probabilistic method

The Fellegi-Sunter probabilistic method estimates match (m) and non-match (u) probabilities for each variable, then computes log-likelihood ratios to score candidate pairs. The m-probabilities represent the probability of agreement among true matched pairs, while u-probabilities represent agreement among random non-matched pairs.

User-supplied `m_probs` and `u_probs` override the supervised estimation. The `fs_threshold` controls the minimum log-likelihood ratio for inclusion in the candidate set.

PRAM method

For data protected via Post-Randomization Method (PRAM), risk is computed directly from transition matrices. For each original record, the probability of producing each anonymized record is computed as the product of per-variable transition probabilities. This avoids distance computation entirely.

Predictive method

When `method = "predictive"`, a propensity model is fit on the stacked original + anonymized data to predict record origin (original = 1, synthetic = 0) from the quasi-identifiers. Records are then linked in the 1D propensity-score space. The propensity score $e(x) = P(\text{original} \mid \text{QIs} = x)$ projects the multivariate QI space into a single dimension that maximally separates the two datasets.

When `pred_se = TRUE` (default) and `pred_model = "logit"`, the bandwidth for kernel weighting is derived from the Fisher information matrix of the logistic regression, giving each record a bandwidth equal to its prediction standard error. This follows the principle from Gaffert et al. (2015): records where the model is uncertain get wider kernels, while confidently classified records get narrow kernels (higher individual risk).

RF method

When `method = "rf"`, a random forest (ranger) is trained on the stacked original + anonymized records to learn a supervised similarity. Terminal-node co-occurrence across trees yields a proximity matrix in $[0, 1]$. Each original record is linked to the anonymized record with highest proximity. Per-block RF is used by default; set `rf_global = TRUE` to train a single global forest and restrict proximity lookups within blocks. Blocks with fewer than 2 records per class fall back to a global RF second pass. Bijective matching is supported via the existing `matching = "bijective"` mechanism with `maximize = TRUE`.

Note: `risk` for the RF method is the re-identification probability of the *true* match within the attacker's candidate set (consistent with the distance/probability methods), obtained by treating the proximity as a similarity and routing it through the shared candidate-set / risk-weighting logic. The nearest-neighbour proximity (the former `risk`) is retained in the `nn_similarity` column of `per_record`. The `d_min` and `d_true` fields still store proximity values (higher = more similar), inverted from the distance convention used by other methods.

RBRL method

When `method = "rbrl"`, rank-based record linkage is used following Muralidhar & Domingo-Ferrer (2016). Each numeric/ordinal variable is replaced by its normalized rank (within each dataset independently), yielding values in $[0, 1]$. Linkage distance is the weighted mean absolute rank difference across variables. Nominal variables use exact matching (0 if equal, 1 if not), consistent with Gower distance.

RBRL is robust to monotone perturbations (additive noise, rounding, top/bottom coding) because rank order is preserved under such transformations. This makes it particularly useful for assessing risk when the anonymization method preserves variable ordering. All `strategy`, `risk_weighting`, `matching`, and `block` parameters work with RBRL.

Mahalanobis method

When `method = "mahalanobis"`, distances are computed using the Mahalanobis metric, which accounts for correlations between variables. Records that appear 'close' in Gower distance may be 'far' in Mahalanobis distance if their difference goes against the data's correlation structure, and vice versa.

By default (`robust = TRUE`), the covariance matrix is estimated using the Minimum Covariance Determinant (MCD) via `robustbase::covMcd()`, with fallback to `MASS::cov.rob()` then classical `cov()` if MCD fails. The covariance is always estimated from the original data regardless of `direction`, modelling an attacker who knows the population covariance structure (Templ & Meindl, 2008).

For mixed data, numeric/ordinal variables use Mahalanobis distance (normalized by a chi-squared threshold) while nominal variables use Gower-style exact matching. The two components are combined as a weighted average, where the weight is the proportion of each variable type. For purely nominal data, use `method = "deterministic"` instead.

Ordinal factors are converted to integer codes (1, 2, 3, ...), assuming approximately equal spacing between levels.

OT matching

When `matching = "ot"`, an entropy-regularized optimal transport plan replaces the hard assignment of bijective matching. The Sinkhorn-Knopp algorithm computes a transport plan P that minimizes

$$\sum_{ij} P_{ij} C_{ij} + \varepsilon \sum_{ij} P_{ij} \log P_{ij}$$

subject to marginal constraints, where C is the cost matrix from the chosen `method` and ε is the regularization strength (`ot_epsilon`).

Risk for record i is defined as $P_{i,\text{true}} \times n_{\text{query}}$, where $P_{i,\text{true}}$ is the transport weight from query record i to its true match. Under uniform random matching this gives $1/n_{\text{search}}$ (the same baseline as independent matching).

OT matching produces *continuous* risk scores in $[0, 1]$, interpolating between bijective matching (hard one-to-one assignment) and uniform random matching. As $\varepsilon \to 0$, the transport plan converges to the Hungarian assignment; as $\varepsilon \to \infty$, it approaches a uniform plan where every record gets equal risk $1/n_{\text{search}}$. If `ot_epsilon = NULL` (default), epsilon is auto-calibrated from the cost matrix as `median(C[C > 0]) * 0.05`. Note that the auto-calibrated epsilon depends on the cost scale, which varies across methods (e.g., Gower distances in $[0, 1]$ vs.\ probabilistic log-likelihood ratios). Typical useful values range from 0.001 (nearly deterministic) to 1 (nearly uniform).

The OT attacker model represents an adversary who performs a global soft assignment, hedging uncertainty across multiple candidate matches via a maximum-entropy prior over assignments. This is appropriate when the attacker is uncertain about the true one-to-one mapping but still optimizes globally. The resulting risk is a *relative risk index* proportional to the attacker's confidence, not a re-identification probability in the frequentist sense.

When to use which matching mode: Use `"independent"` for classical per-record risk (DBRL). Use `"bijective"` when the attacker is known to perform a one-to-one assignment (GDBRL; binary risk). Use `"ot"` for a global assignment that yields continuous risk, or to explore the sensitivity of risk to the assignment model via the `ot_epsilon` parameter.

Softmax risk weighting

When `risk_weighting = "softmax"`, closer candidates receive higher attribution probability via:

$$w_j = \frac{\exp(-\kappa \cdot d_j)}{\sum_k \exp(-\kappa \cdot d_k)}$$

The temperature `kappa` is auto-calibrated from the distance range if not supplied. This replaces the uniform $1/|candidate\_set|$ risk.

Kernel risk weighting

When `risk_weighting = "kernel"`, donor probabilities are computed via kernel-weighted distances. Each candidate j receives weight proportional to a kernel function evaluated at its scaled distance:

$$w_j = K(d_j/h) \Big/ \sum_k K(d_k/h)$$

where h is a bandwidth and K is one of:

- **gaussian** (default): $K(u) = \exp(-u^2/2)$
- **epanechnikov**: $K(u) = \max(0, 1 - u^2) \cdot 3/4$
- **tricube**: $K(u) = \max(0, (1 - |u|^3)^3) \cdot 70/81$

With compact kernels (Epanechnikov, tricube), candidates beyond the bandwidth receive exactly zero weight and are effectively excluded. The bandwidth is auto-selected via Silverman's rule if not supplied.

Direction

By default (`direction = "forward"`) the function loops over original records and finds matches in the anonymized data. This quantifies how easily each individual in the population can be re-identified.

With `direction = "reverse"` the loop runs over anonymized records, searching for matches in the original data. This gives a risk profile of the data to be released: each row in `per_record` corresponds to a released record and its probability of being correctly linked back to the original. Deterministic, probabilistic, and predictive methods are symmetric (distance/agreement does not depend on direction), so only the evaluation perspective changes. The PRAM method is asymmetric: the transition matrix $P(\text{output} \mid \text{input})$ is looked up with swapped roles.

Embedding method

The `method = "embedding"` approach trains an autoencoder with entity embeddings (Guo & Berkhahn, 2016) on the original (query) data, projects both query and search records into a latent space, and measures re-identification risk via Euclidean distance. This captures nonlinear dependencies between quasi-identifiers that Gower and Mahalanobis distances may miss.

As with the RF method, `risk` is the re-identification probability of the *true* match within the attacker's candidate set (the shared candidate-set / risk-weighting logic applied to the latent distance); the nearest-neighbour similarity (`1 -` the smallest latent distance, the former `risk`) is kept in the `nn_similarity` column of `per_record`.

The autoencoder trains only on query data, modeling an attacker who knows the population structure but not the specific anonymization. Distances are normalized to $[0, 1]$ using the 97.5th percentile of within-query pairwise distances. Variable importance is permutation-based: each variable is shuffled and the mean embedding shift is measured.

Requires the **torch** package together with its C++ backend (libtorch), which is downloaded once via `torch::install_torch()`. Entity embeddings handle mixed data naturally; all-numeric and all-categorical datasets are supported.

Author(s)

Matthias Templ and Roman Müller

References

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. Biometrics, 27(4), 857-871.

Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. Journal of the American Statistical Association, 64(328), 1183-1210.

Domingo-Ferrer, J., & Torra, V. (2002). Validating distance-based record linkage with probabilistic record linkage. Lecture Notes in Computer Science, 2504, 207-215. Springer.

Gaffert, P., Meinfelder, F., & Bosch, V. (2015). Towards an MI-proper Predictive Mean Matching. Discussion Paper, University of Bamberg.

Pagliuca, D., & Seri, G. (1999). Some results of individual ranking method on the system of enterprise accounts annual survey (Esprit SDC Project, Deliverable MI-3/D2). Unpublished technical report.

Herranz, J., Nin, J., Rodriguez, P., & Tassa, T. (2016). Revisiting distance-based record linkage for privacy-preserving release of statistical datasets. Data & Knowledge Engineering, 100, 78-93.

Cuturi, M. (2013). Sinkhorn Distances: Lightspeed Computation of Optimal Transport. Advances in Neural Information Processing Systems, 26.

Guo, C., & Berkhahn, F. (2016). Entity Embeddings of Categorical Variables. arXiv preprint arXiv:1604.06737.

See Also

`individual_risk`, `dcr`, `nndr`

Other privacy-models: `attacker_risk()`, `delta_presence()`, `disclosure_report()`, `domias()`, `drisk()`, `epsilon_identifiability()`, `individual_risk()`, `inspect_record()`, `kanonymity()`, `ldiversity()`, `linkability()`, `merge_per_record()`, `mia_classifier()`, `population_uniqueness()`, `risk_by_group()`, `singling_out()`, `suda()`, `tcloseness()`, `top_at_risk()`

Examples

```
set.seed(1)
n <- 100
x <- data.frame(
  age    = sample(18:80, n, TRUE),
  sex    = factor(sample(c("f","m"), n, TRUE)),
```

```
  region = factor(sample(paste0("R",1:5), n, TRUE))
)

# Simple perturbation: swap age within region
x_anon <- x
for (r in levels(x$region)) {
  idx <- which(x$region == r)
  x_anon$age[idx] <- sample(x$age[idx])
}

# Deterministic nearest-neighbor (default)
res1 <- recordLinkage(x, x_anon, key = c("age","sex","region"))
print(res1)
summary(res1)

# Reverse direction: risk per released record
res1r <- recordLinkage(x, x_anon, key = c("age","sex","region"),
                       direction = "reverse")
print(res1r)


# Softmax distance-weighted risk
res2 <- recordLinkage(x, x_anon, key = c("age","sex","region"),
                      risk_weighting = "softmax")
print(res2)

# Kernel-weighted risk (Gaussian kernel, auto bandwidth)
res2b <- recordLinkage(x, x_anon, key = c("age","sex","region"),
                       risk_weighting = "kernel")
print(res2b)

# Kernel weighting with Epanechnikov kernel
res2c <- recordLinkage(x, x_anon, key = c("age","sex","region"),
                       risk_weighting = "kernel",
                       kernel = "epanechnikov")
print(res2c)

# Probabilistic (Fellegi-Sunter) method
res3 <- recordLinkage(x, x_anon, key = c("age","sex","region"),
                      method = "probabilistic")
print(res3)

# With synth_pair
pair <- synth_pair(x, x_anon, key_vars = c("age","sex","region"))
res4 <- recordLinkage(pair)
print(res4)

# Predictive method (propensity-score-based)
res5 <- recordLinkage(x, x_anon, key = c("age","sex","region"),
                      method = "predictive")
print(res5)
plot(res5, which = 4)  # propensity distributions
```

```
# Optimal transport matching (continuous risk via Sinkhorn)
res_ot <- recordLinkage(x, x_anon, key = c("age","sex","region"),
                        matching = "ot")
print(res_ot)

# Plot risk distribution
plot(res1)
plot(res1, which = 2)
```

`regression_fidelity` *Regression Fidelity — Coefficient Comparison*

Description

Fits the same regression model on original and synthetic data, then compares coefficients to assess specific-analysis fidelity. The utility score is the mean confidence interval overlap across all coefficients.

Usage

```
regression_fidelity(X, ...)

## S3 method for class 'synth_pair'
regression_fidelity(X, ...)

## Default S3 method:
regression_fidelity(
  X,
  Y,
  formula = NULL,
  model = c("lm", "glm"),
  family = binomial(),
  conf_level = 0.95,
  na.rm = TRUE,
  ...
)
```

Arguments

`X`	A data.frame containing the original dataset.
`...`	Additional arguments passed to the model fitting function (`lm` or `glm`).
`Y`	A data.frame containing the synthetic/anonymized dataset.
`formula`	An R formula specifying the regression model. Required; an error is raised if not provided. For example, `y ~ x1 + x2`.
`model`	Character, the type of model to fit. One of `"lm"` (default) or `"glm"`.

family	A family object for GLM fitting (e.g., binomial()). Only used when model = "glm". Default binomial().
conf_level	Numeric, the confidence level for confidence intervals. Default 0.95.
na.rm	Logical, whether to remove rows with NA values in the variables used by the formula. Default TRUE.

Details

For each coefficient, the following quantities are computed:

- **Bias**: difference between synthetic and original estimates (synth - orig)
- **Standardized bias**: bias divided by the original standard error (bias / SE_orig)
- **CI overlap**: overlap length divided by the average of the two interval widths, clamped to $[0, 1]$
- **Significance agreement**: whether both models agree on statistical significance at the given confidence level

The CI overlap for two intervals $[lo_1, hi_1]$ and $[lo_2, hi_2]$ is:

$$overlap = \frac{\max(0, \min(hi_1, hi_2) - \max(lo_1, lo_2))}{0.5 \cdot (w_1 + w_2)}$$

where $w_k = hi_k - lo_k$. The result is clamped to $[0, 1]$.

For model = "lm", confidence intervals use the t-distribution with the appropriate residual degrees of freedom. For model = "glm", Wald intervals based on the normal distribution are used.

A utility score of 1 indicates perfect overlap (coefficients are identical), while 0 indicates no overlap at all. The utility_score field provides a standard interface compatible with subgroup_utility and rumap.

Interpretation (heuristic thresholds):

- utility_score > 0.9: EXCELLENT – regression results very well preserved
- utility_score > 0.7: GOOD – reasonably preserved
- utility_score > 0.4: MODERATE – some differences
- utility_score <= 0.4: POOR – significant differences

Value

An object of class "regression_fidelity" containing:

- utility_score: mean CI overlap across coefficients, in $[0, 1]$
- coefficients: data.frame with columns: term, estimate_orig, estimate_synth, se_orig, se_synth, bias, std_bias, ci_overlap, sig_orig, sig_synth, sig_agreement
- mean_ci_overlap: same as utility_score
- mean_abs_std_bias: mean absolute standardized bias
- sig_agreement_rate: proportion of coefficients with matching significance
- formula: the formula used
- model: the model type used

- conf_level: the confidence level used
- n_X: number of observations in original data
- n_Y: number of observations in synthetic data
- n_coef: number of coefficients compared

Author(s)

Matthias Templ

References

Karr, A. F., Kohnen, C. N., Oganian, A., Reiter, J. P., and Sanil, A. P. (2006). A Framework for Evaluating the Utility of Data Altered to Protect Confidentiality. The American Statistician, 60(3), 224-232.

Snoke, J., Raab, G. M., Nowok, B., Dibben, C., and Slavkovic, A. (2018). General and Specific Utility Measures for Synthetic Data. Journal of the Royal Statistical Society: Series A, 181(3), 663-688.

See Also

propscore for propensity score utility, compare_model_performance for predictive performance comparison, contingency_fidelity for categorical dependence comparison, subgroup_utility for stratified utility assessment

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), gower(), hellinger(), mmd(), mqs(), pMSE(), plot.rumap(), propscore(), rumap(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
set.seed(123)
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
y <- 2 + 3 * x1 - 1.5 * x2 + rnorm(n)
X <- data.frame(y = y, x1 = x1, x2 = x2)

# Good synthetic data (similar DGP)
x1s <- rnorm(n)
x2s <- rnorm(n)
ys <- 2 + 3 * x1s - 1.5 * x2s + rnorm(n)
Y_good <- data.frame(y = ys, x1 = x1s, x2 = x2s)

result <- regression_fidelity(X, Y_good, formula = y ~ x1 + x2)
print(result)
summary(result)

# Using synth_pair
pair <- synth_pair(X, Y_good)
result2 <- regression_fidelity(pair, formula = y ~ x1 + x2)
```

```
# Poor synthetic data (wrong coefficients)
ys_bad <- 10 + 0.1 * x1s + 5 * x2s + rnorm(n)
Y_bad <- data.frame(y = ys_bad, x1 = x1s, x2 = x2s)
result_bad <- regression_fidelity(X, Y_bad, formula = y ~ x1 + x2)
print(result_bad)

# GLM example
y_bin <- rbinom(n, 1, plogis(0.5 + 1.5 * x1))
X_bin <- data.frame(y = y_bin, x1 = x1, x2 = x2)
y_bin_s <- rbinom(n, 1, plogis(0.5 + 1.5 * x1s))
Y_bin <- data.frame(y = y_bin_s, x1 = x1s, x2 = x2s)
result_glm <- regression_fidelity(X_bin, Y_bin,
  formula = y ~ x1 + x2, model = "glm")
print(result_glm)
```

`repu` *Replicated Uniques (RepU)*

Description

Alias for `ims`. Computes the share of synthetic records that are identical to unique (singleton) records in the training data. This is particularly concerning as unique records are more identifiable.

Usage

```
repu(X, Y, vars = NULL, uniques_only = TRUE, na.rm = TRUE)
```

Arguments

`X`	data frame of original/training data
`Y`	data frame of synthetic data
`vars`	character vector of variable names to use for matching. If NULL (default), all common variables are used.
`uniques_only`	logical, if TRUE (default), only count matches to records that are unique in the training data. If FALSE, equivalent to `ims`.
`na.rm`	logical, remove records with NA values (default: TRUE)

Details

Replicated Uniques (RepU) is a variant of IMS that focuses specifically on synthetic records that match unique (singleton) training records. Unique records are more identifiable and thus pose a higher privacy risk if copied.

Value

An object of class "ims" (same structure as `ims`)

See Also

ims for identical match share

Other distance-risk: dcr(), hitting_rate(), ims(), nnaa(), nndr(), rf_privacy()

Examples

```
set.seed(123)
X <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  income = sample(c("low", "medium", "high"), 100, replace = TRUE)
)
Y <- X[sample(nrow(X), 50, replace = TRUE), ]

# All identical matches
result_ims <- ims(X, Y)

# Only matches to unique training records
result_repu <- repu(X, Y)
```

---

rf_privacy *RF Proximity Privacy Assessment*

Description

Detects memorization in synthetic data using Random Forest proximity. Trains a supervised RF to discriminate original from synthetic records, then compares proximity of synthetic records to training vs. holdout subsets of the original data.

Usage

```
rf_privacy(X, ...)

## S3 method for class 'synth_pair'
rf_privacy(X, ...)

## Default S3 method:
rf_privacy(
  X,
  Y,
  holdout = NULL,
  holdout_fraction = 0.5,
  vars = NULL,
  na.rm = FALSE,
  seed = NULL,
  progress = FALSE,
  null_test = TRUE,
```

```
  n_null = 100L,
  n_trees = 500L,
  mtry = NULL,
  ...
)

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

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

## S3 method for class 'summary.rf_privacy'
print(x, ...)

## S3 method for class 'rf_privacy'
plot(x, y = NULL, ..., which = 1)
```

Arguments

X	For `rf_privacy.default`: a data.frame of original data. For `rf_privacy.synth_pair`: a `synth_pair` object.
...	additional arguments (not used)
Y	data.frame of synthetic data (for default method)
holdout	data.frame or NULL. If NULL, split from X.
holdout_fraction	numeric in (0, 1), fraction of X to use as holdout
vars	character vector of variable names (NULL = all common)
na.rm	logical. If FALSE (default), ranger handles NAs natively via surrogate splits. If TRUE, remove records with any NA before training.
seed	integer or NULL. Used for holdout splitting (seed) and forest training (seed + 1) to avoid correlation.
progress	logical, show progress bar during proximity computation
null_test	logical, run permutation null test (default TRUE)
n_null	integer, number of permutations for null test
n_trees	integer (>= 10), number of trees
mtry	integer or NULL, number of variables to consider at each split
x	an object of class `"rf_privacy"` or `"summary.rf_privacy"`
object	an object of class `"rf_privacy"`
y	not used
which	integer, which plot: 1 = density, 2 = difference histogram, 3 = null distribution

Details

RF proximity measures how often two records land in the same terminal node across all trees. A proximity of 1 means they always co-terminate; 0 means never. This function trains a supervised RF to discriminate ALL original records from synthetic records, then checks whether synthetic records are more similar (proximate) to training records than to holdout records.

Crucially, the forest is trained on the full original dataset (train + holdout combined) so that both subsets have identical in-sample status, eliminating the bias that would arise if holdout records were out-of-sample.

If no memorization occurred, synthetic records should have roughly equal proximity to training and holdout (both are real data, both in the forest). If synthetic records consistently land in terminal nodes dominated by training records, this signals memorization.

Value

An S3 object of class `"rf_privacy"` with fields:

max_prox_share fraction of synthetic records with higher max proximity to training than holdout (mid-rank tie correction)

max_prox_ratio ratio of mean max-proximities (train/holdout)

max_prox_train per-record max proximity to nearest training record

max_prox_holdout per-record max proximity to nearest holdout record

prox_share fraction with higher mean proximity to training

prox_ratio ratio of mean mean-proximities

prox_train_mean per-record mean proximity to training

prox_holdout_mean per-record mean proximity to holdout

privacy_pass logical, TRUE if no memorization detected

wilcox_test Wilcoxon signed-rank test object (heuristic)

null_distribution list with null stats and p-values (if null_test)

oob_error OOB classification error from the forest

var_importance named numeric vector of variable importances

n_synthetic, n_train, n_holdout, vars dataset metadata

When to use this method

Use `rf_privacy()` when you have mixed data types with complex interactions (20+ variables), or when you want a data-adaptive alternative to `dcr()`. For simple QIs with interpretable risk, `dcr()` with Gower distance is more transparent. For speed on small data (n < 5,000), `dcr()` is faster.

Interpretation

`max_prox_share` is the primary metric. Values near 0.5 indicate no memorization; values above 0.5 suggest synthetic records are systematically closer to training than holdout. The `max_prox_ratio` captures the magnitude: values near 1 = no memorization, > 1 = memorization signal.

Mean-based metrics (`prox_share`, `prox_ratio`) detect aggregate distributional leakage rather than individual memorized records.

Comparison with DCR

`rf_privacy()` is the RF-proximity analog of `dcr()`, using the same holdout design but replacing Gower/Euclidean distance with terminal-node co-occurrence. RF proximity is data-adaptive and handles mixed types natively, but adds forest training overhead.

Limitations

The supervised forest conflates distributional similarity with memorization. When OOB error is near 0.5 (high-quality synthetic data), memorization detection is most reliable. When OOB error is low (poor utility), asymmetry may reflect distributional differences, not individual copying.

The Wilcoxon p-value is anti-conservative because all proximity values share the same forest. Use the permutation null test (`null_test = TRUE`) for principled inference.

When `null_test = FALSE`, a heuristic threshold of 0.55 is used for `max_prox_share`. This is a rough default; the permutation null test provides an adaptive, data-driven threshold.

When an explicit `holdout` is provided that is much smaller or larger than the training set, the max-proximity comparison is biased because the maximum over a larger set is naturally higher. The permutation null test compensates for this (it permutes with the same split sizes), but the Wilcoxon heuristic does not. Prefer roughly equal split sizes (the default `holdout_fraction = 0.5`).

Computational considerations

Expected runtimes (n_trees = 500, modern laptop): n = 1,000: ~5 seconds; n = 5,000: ~30 seconds; n = 10,000: 2-5 minutes; n = 50,000: 30+ minutes.

Author(s)

Matthias Templ

References

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.

Lin, Y. & Jeon, Y. (2006). Random forests and adaptive nearest neighbors. *JASA*, 101(474), 578-590.

See Also

`dcr`, `nndr`, `ims`, `propscore`, `recordLinkage`

Other distance-risk: `dcr()`, `hitting_rate()`, `ims()`, `nnaa()`, `nndr()`, `repu()`

Examples

```
set.seed(1)
X_train <- data.frame(a = rnorm(100), b = rnorm(100), c = rnorm(100))
X_holdout <- data.frame(a = rnorm(100), b = rnorm(100), c = rnorm(100))

# Memorized: synthetic copies training records
Y_mem <- X_train[sample(100, 100, replace = TRUE), ]
Y_mem <- Y_mem + rnorm(300, 0, 0.01)
```

```
res_mem <- rf_privacy(X_train, Y_mem, holdout = X_holdout,
                      seed = 1, n_trees = 200, null_test = FALSE)
print(res_mem)

# Random synthetic data (no memorization)
Y_rand <- data.frame(a = rnorm(100), b = rnorm(100), c = rnorm(100))
res_rand <- rf_privacy(X_train, Y_rand, holdout = X_holdout,
                       seed = 1, n_trees = 200, null_test = FALSE)
print(res_rand)
```

`risk_by_group` *Aggregate Risk by Group*

Description

Computes per-group risk statistics from a `recordLinkageRisk` object.

Usage

```
risk_by_group(x, ...)

## S3 method for class 'recordLinkageRisk'
risk_by_group(x, group, data = NULL, ...)
```

Arguments

`x`	object of class `"recordLinkageRisk"`.
`...`	ignored.
`group`	a vector of group labels (same length as number of records), or a single column name if `data` is provided.
`data`	optional data frame from which to extract the grouping column.

Value

A data frame with columns `mean_risk`, `max_risk`, `n`, `n_high`, `pct_high`, and the grouping variable, sorted by `mean_risk` descending.

Author(s)

Matthias Templ, Oscar Thees

See Also

Other privacy-models: `attacker_risk()`, `delta_presence()`, `disclosure_report()`, `domias()`, `drisk()`, `epsilon_identifiability()`, `individual_risk()`, `inspect_record()`, `kanonymity()`, `ldiversity()`, `linkability()`, `merge_per_record()`, `mia_classifier()`, `population_uniqueness()`, `recordLinkage()`, `singling_out()`, `suda()`, `tcloseness()`, `top_at_risk()`

rumap *Multivariate Risk-Utility Map (RU-Map)*

Description

Computes multiple disclosure risk and data utility measures for synthetic/anonymized data and provides comprehensive visualization tools for multivariate evaluation. This implements the framework from "Beyond the Trade-off Curve" (Thees, Müller, Templ 2026).

Usage

```
rumap(X, ...)

## S3 method for class 'synth_pair'
rumap(X, ...)

## Default S3 method:
rumap(
  X,
  synthetic,
  risk_measures = c("dcap", "tcap", "disco", "rapid", "ims", "repu"),
 utility_measures = c("pmse", "wasserstein", "hellinger", "energy", "ci_proximity"),
  key_vars = NULL,
  target_var = NULL,
  holdout = NULL,
  holdout_fraction = 0.2,
  vars = NULL,
  cat_vars = NULL,
  num_vars = NULL,
  normalize = TRUE,
  seed = NULL,
  na.rm = TRUE,
  Y = NULL,
  ...
)
```

Arguments

X	A synth_pair object, or a data.frame containing the original dataset.
...	additional arguments passed to methods
synthetic	A data.frame or named list of data.frames containing synthetic dataset(s).
risk_measures	Character vector of risk measures to compute. Options: "dcap", "tcap", "disco", "ims", "repu", "dcr", "nndr", "rapid", "rf_privacy". Default includes all except distance-based measures.
utility_measures	Character vector of utility measures to compute. Options: "pmse", "wasserstein", "hellinger", "energy", "ci_proximity". Default includes all.

key_vars	Character vector of quasi-identifier variables (for attribution-based risk).
target_var	Character string naming the sensitive target variable (for CAP metrics).
holdout	Optional data.frame for distance-based risk measures. If NULL and distance-based measures are requested, uses holdout_fraction to split original.
holdout_fraction	Numeric, fraction of original to use as holdout if holdout is not provided. Default 0.2.
vars	Character vector of variables to use for utility measures. If NULL, uses all common variables.
cat_vars	Character vector of categorical variables for Hellinger distance. If NULL, auto-detected.
num_vars	Character vector of numeric variables for energy distance. If NULL, auto-detected.
normalize	Logical, whether to normalize measures to the 0-1 range. Default TRUE.
seed	Integer, random seed for reproducibility.
na.rm	Logical, whether to remove NA values. Default TRUE.
Y	Alias for synthetic, for naming consistency with other measures in the package. If synthetic is not supplied but Y is, Y is used. If a named list, each element is treated as a different synthetic data generator (SDG).

Details

The rumap function provides a comprehensive framework for evaluating synthetic data by computing multiple risk and utility measures simultaneously. It supports:

Risk Measures:

- dcap: Differential Correct Attribution Probability
- tcap: Targeted CAP (mean per-record risk)
- disco: Disclosive in Synthetic Correct Original
- rapid: Risk of Attribute Prediction-Induced Disclosure (ML-based)
- ims: Identical Match Share
- repu: Replicated Uniques
- dcr: Distance to Closest Record ratio
- nndr: Nearest Neighbor Distance Ratio
- rf_privacy: Random forest proximity-based memorization test (requires ranger)

Utility Measures:

- pmse: Propensity Mean Squared Error (from propscore)
- wasserstein: Mean Wasserstein distance across numeric variables
- hellinger: Mean Hellinger distance across categorical variables
- energy: Energy distance for multivariate numeric data
- ci_proximity: Confidence interval proximity score

All measures are normalized to the 0-1 range with consistent direction: higher risk values = higher disclosure risk (bad), higher utility values = higher utility (good).

Pareto-optimal SDGs are identified as those not dominated by any other SDG (cannot improve risk without worsening utility, and vice versa).

Value

An object of class "rumap" containing:

- risk: data.frame of raw risk measures per SDG
- utility: data.frame of raw utility measures per SDG
- normalized: data.frame of normalized measures (if normalize=TRUE)
- composites: data.frame with composite risk/utility scores
- pareto: logical vector indicating Pareto-optimal SDGs
- pareto_sdgs: names of Pareto-optimal SDGs
- n_sdgs: number of synthetic datasets evaluated
- sdg_names: names of SDGs
- risk_measures, utility_measures: measures computed
- metadata: list of parameters used

Author(s)

Matthias Templ

References

Thees, O., Müller, R., & Templ, M. (2026). Beyond the Trade-off Curve: Multivariate and Advanced Risk-Utility Maps for Evaluating Anonymized and Synthetic Data. Journal of Official Statistics.

See Also

plot.rumap for visualization options, dcap, tcap, disco, ims, propscore, hellinger, energy_distance

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), gower(), hellinger(), mmd(), mqs(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), specks(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
set.seed(123)
# Create original data
original <- data.frame(
  age = sample(20:70, 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE),
  income = rnorm(200, 50000, 15000)
)
```

```
# Create synthetic datasets (simulating different SDGs)
synth_good <- data.frame(
  age = sample(20:70, 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE),
  income = rnorm(200, 50000, 15000)
)

synth_poor <- data.frame(
  age = sample(30:50, 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE, prob = c(0.8, 0.2)),
 region = sample(c("N", "S", "E", "W"), 200, replace = TRUE, prob = c(0.7, 0.1, 0.1, 0.1)),
  income = rnorm(200, 60000, 20000)
)

# Compare multiple SDGs
result <- rumap(
  X = original,
  synthetic = list(good_sdg = synth_good, poor_sdg = synth_poor),
  key_vars = c("age", "gender", "region"),
  target_var = "income",
  risk_measures = c("ims", "dcap"),
  utility_measures = c("hellinger", "energy", "ci_proximity"),
  seed = 42
)

print(result)
summary(result)
```

`singling_out` *Singling Out Risk*

Description

Computes the Singling Out Risk for synthetic data, one of three explicit GDPR anonymization failure criteria (Article 29 Working Party). An attacker uses synthetic data to craft logical predicates (conjunctions of column conditions) and tests whether these predicates uniquely identify exactly one record in the original data.

Usage

```
singling_out(X, ...)

## S3 method for class 'synth_pair'
singling_out(X, ...)

## Default S3 method:
```

```
singling_out(
  X,
  Y,
  holdout = NULL,
  holdout_fraction = 0.5,
  n_attacks = 2000,
  n_cols = 3,
  mode = c("multivariate", "univariate"),
  vars = NULL,
  na.rm = TRUE,
  seed = NULL,
  confidence_level = 0.95,
  ...
)
```

Arguments

X	data frame of original/training data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
holdout	data frame of holdout data (optional). If NULL, a holdout set is automatically created by splitting X.
holdout_fraction	numeric, fraction of X to use as holdout if holdout is NULL (default: 0.5)
n_attacks	integer, number of predicate attacks to generate (default: 2000)
n_cols	integer, number of columns to use per predicate in multivariate mode (default: 3)
mode	character, attack mode: "multivariate" (default) builds predicates from random column subsets; "univariate" targets single columns
vars	character vector of variable names to use. If NULL (default), all common variables between X and Y are used.
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for reproducibility (default: NULL)
confidence_level	numeric, confidence level for Wilson score intervals (default: 0.95)

Details

The singling out attack works by sampling random records from the synthetic data and constructing logical predicates from their column values. For each predicate, the algorithm checks how many records in the original data match. A predicate that matches exactly one record constitutes a successful singling out attack.

In multivariate mode (default), each predicate is a conjunction of conditions on `n_cols` randomly selected columns:

- Categorical/factor columns: exact match (`col == val`)

- Numeric columns: inequality based on median (`col >= val` if val > median, `col <= val` otherwise), targeting distribution tails

In univariate mode, each predicate tests a single column condition.

The risk score uses Wilson score intervals for robust estimation:

$$r_{attack} = n_{success}/n_{attacks}$$

$$r_{control} = n_{control\_success}/n_{attacks}$$

$$risk = (r_{attack} - r_{control})/(1 - r_{control})$$

Without holdout data, the absolute attack rate is reported as the risk.

Value

An object of class "singling_out" containing:

- risk: residual risk score (or absolute if no holdout), bounded between 0 and 1
- risk_ci: confidence interval for risk
- risk_attack: attack success rate (fraction singling out in original)
- risk_attack_ci: Wilson CI for attack success rate
- risk_control: control success rate (fraction singling out in holdout, NA if none)
- risk_control_ci: Wilson CI for control success rate (NA if none)
- n_attacks: number of predicates generated
- n_success: number singling out in original
- n_control_success: number singling out in holdout
- match_counts: integer vector of match counts per predicate in original
- match_counts_control: integer vector of match counts per predicate in holdout
- privacy_pass: logical, risk <= 0.1
- n_original: number of original records (training portion)
- n_synthetic: number of synthetic records
- n_holdout: number of holdout records
- mode: attack mode used
- n_cols: number of columns per predicate
- vars: variables used
- confidence_level: confidence level used

Holdout splitting

When no external holdout is provided, the original data is split internally. The holdout serves as a control: predicates that single out records in the holdout (which were not used for synthesis) represent baseline singling out risk due to data structure rather than information leakage. The residual risk subtracts this baseline.

Author(s)

Matthias Templ

References

Giomi, M., Boenisch, F., Wehmeyer, C. & Tasnadi, B. (2023). A Unified Framework for Quantifying Privacy Risk in Synthetic Data. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2023(2), 312–328. doi:10.56553/popets20230055

Cohen, A. & Nissim, K. (2020). Towards Formalizing the GDPR's Notion of Singling Out. *Proceedings of the National Academy of Sciences*, 117(15), 8344–8352. doi:10.1073/pnas.1914598117

Article 29 Data Protection Working Party (2014). Opinion 05/2014 on Anonymisation Techniques. WP216.

See Also

dcap for differential correct attribution probability, tcap for targeted CAP, nnaa for nearest-neighbor adversarial accuracy

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), suda(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
n <- 200
X <- data.frame(
  age = sample(20:70, n, replace = TRUE),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

# Good synthetic data (independent, no memorization)
Y <- data.frame(
  age = sample(20:70, n, replace = TRUE),
  income = rnorm(n, 50000, 15000),
  gender = sample(c("M", "F"), n, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)

result <- singling_out(X, Y, n_attacks = 500, seed = 42)
print(result)
summary(result)


# Memorized data (Y is copy of X) - should show high risk
Y_copy <- X[sample(nrow(X), n, replace = TRUE), ]
result_bad <- singling_out(X, Y_copy, n_attacks = 500, seed = 42)
```

```
print(result_bad)
```

---

specks *SPECKS - Propensity Score Comparison via Kolmogorov-Smirnov Test*

Description

Computes the SPECKS (Synthetic data generation; Propensity score matching; Empirical Comparison via the Kolmogorov-Smirnov distance) utility measure for synthetic data. This metric assesses how distinguishable synthetic data is from original data using propensity scores.

Usage

```
specks(X, ...)

## S3 method for class 'synth_pair'
specks(X, ...)

## Default S3 method:
specks(
  X,
  Y,
  vars = NULL,
  method = c("cart", "logit", "rf"),
  maxorder = 1,
  k = NULL,
  na.rm = TRUE,
  seed = NULL,
  ...
)
```

Arguments

X	data frame of original data, or a synth_pair object
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data (not needed if X is a synth_pair)
vars	character vector of variable names to use. If NULL, all common variables are used.
method	character, method for propensity score estimation: "cart" (classification tree, default), "logit" (logistic regression), or "rf" (random forest)
maxorder	integer, maximum order of interactions for logit method (default: 1)
k	integer, number of percentiles to use for KS calculation (default: NULL uses all unique values)
na.rm	logical, remove records with NA values (default: TRUE)
seed	integer, random seed for reproducibility (default: NULL)

Details

SPECKS (Woo et al. 2009) is a utility measure that assesses how well synthetic data preserves the joint distribution of variables. It works by:

1. Combining original (X) and synthetic (Y) data with labels
2. Fitting a propensity score model to predict the probability of being synthetic
3. Computing the Kolmogorov-Smirnov statistic between the propensity score distributions of original and synthetic records

Interpretation:

- **SPECKS near 0**: Excellent utility - synthetic indistinguishable from original
- **SPECKS < 0.1**: Good utility - minor distributional differences
- **SPECKS < 0.2**: Moderate utility - noticeable differences
- **SPECKS > 0.3**: Poor utility - substantial differences

The pMSE (propensity Mean Squared Error) measures the average squared deviation of propensity scores from the expected value under random assignment:

$$pMSE = \frac{1}{N} \sum_{i=1}^{N} (\hat{p}_i - c)^2$$

where $c = n_Y/(n_X + n_Y)$ is the proportion of synthetic records.

The pMSE ratio compares pMSE to its null expectation, with values near 1 indicating good utility.

Value

An object of class "specks" containing:

- specks: the SPECKS statistic (KS test statistic on propensity scores)
- pMSE: propensity score Mean Squared Error
- pMSE_ratio: ratio of pMSE to null expectation
- ks_pvalue: p-value from the KS test
- propensity_original: propensity scores for original records
- propensity_synthetic: propensity scores for synthetic records
- utility_score: 1 - specks (higher = better utility)
- method: the method used
- n_original, n_synthetic: dataset sizes

Author(s)

Matthias Templ

References

Woo, M. J., Reiter, J. P., Oganian, A., & Karr, A. F. (2009). Global measures of data utility for microdata masked for disclosure limitation. *Journal of Privacy and Confidentiality*, 1(1), 111-124.

Snoke, J., Raab, G. M., Nowok, B., Dibben, C., & Slavkovic, A. (2018). General and specific utility measures for synthetic data. *Journal of the Royal Statistical Society: Series A*, 181(3), 663-688.

See Also

propscore for related propensity score utility, compare_distributions_cont for distributional comparisons

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), gower(), hellinger(), mmd(), mqs(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), rumap(), subgroup_utility(), tail_fidelity(), tstr()

Examples

```
# Create example data
set.seed(123)
n <- 500
original <- data.frame(
  age = sample(18:80, n, replace = TRUE),
  gender = sample(c("M", "F"), n, replace = TRUE),
  income = exp(rnorm(n, 10, 1)),
  education = sample(c("low", "medium", "high"), n, replace = TRUE)
)

# Good synthetic data (similar distributions)
synthetic_good <- data.frame(
  age = sample(18:80, n, replace = TRUE),
  gender = sample(c("M", "F"), n, replace = TRUE),
  income = exp(rnorm(n, 10, 1)),
  education = sample(c("low", "medium", "high"), n, replace = TRUE)
)

result_good <- specks(original, synthetic_good)
print(result_good)
summary(result_good)

# Poor synthetic data (different distributions)
synthetic_poor <- data.frame(
  age = sample(18:80, n, replace = TRUE, prob = c(rep(0.01, 30), rep(0.02, 33))),
  gender = sample(c("M", "F"), n, replace = TRUE, prob = c(0.8, 0.2)),
  income = exp(rnorm(n, 11, 0.5)),
 education = sample(c("low", "medium", "high"), n, replace = TRUE, prob = c(0.1, 0.2, 0.7))
)

result_poor <- specks(original, synthetic_poor)
print(result_poor)

# Compare results
cat("Good synthetic SPECKS:", result_good$specks, "\n")
```

```
cat("Poor synthetic SPECKS:", result_poor$specks, "\n")
```

---

subgroup_utility *Stratified Utility Assessment Across Subgroups*

Description

Evaluates whether synthetic data utility holds across subgroups defined by a grouping variable. Synthetic data can look good overall but perform poorly for minority subgroups. This function applies a utility measure to each subgroup and identifies groups with low utility.

Usage

```
subgroup_utility(X, ...)

## S3 method for class 'synth_pair'
subgroup_utility(X, ...)

## Default S3 method:
subgroup_utility(
  X,
  Y,
  group_var,
  utility_fun = energy_distance,
  threshold = 0.5,
  na.rm = TRUE,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset. For the synth_pair method, a list with $original and $synthetic elements.
...	Additional arguments passed to utility_fun (e.g., seed for functions that accept it).
Y	A data.frame or data.table containing the synthetic/anonymized dataset.
group_var	Character string naming the column to stratify by. Must be present in both datasets. If the column is numeric, it is converted to a factor with a message.
utility_fun	Function to apply per subgroup. Must accept two data.frames as its first two arguments and return an object with a $utility_score element. Default energy_distance.
threshold	Numeric, subgroups with utility below this value are flagged. Default 0.5.
na.rm	Logical, whether to remove rows where group_var is NA. Default TRUE.

Details

The function splits both original and synthetic data by the levels of `group_var`, then applies `utility_fun` to each subgroup pair. Subgroups with fewer than 5 observations in either dataset are skipped (their utility is set to NA) with a warning.

The conservative `utility_score` returned is the minimum across all evaluated subgroups, reflecting a worst-case perspective. The `ratio` of worst to overall utility helps identify whether particular subgroups are disproportionately affected.

Interpretation guidelines:

- Ratio near 1: utility is homogeneous across subgroups
- Ratio below 0.5: substantial disparity; the worst subgroup has less than half the overall utility
- Flagged subgroups (below `threshold`) warrant investigation and potential re-synthesis

Value

An object of class `"subgroup_utility"` containing:

- `overall_score`: utility computed on the full (ungrouped) data
- `utility_score`: worst subgroup score (conservative measure)
- `per_group`: data.frame with columns `group`, `n_orig`, `n_synth`, `utility_score`, `flagged`
- `worst_group`: name of the worst-performing subgroup
- `ratio`: worst-subgroup score / overall score
- `group_var`: grouping variable name
- `threshold`: threshold used for flagging
- `n_groups`: number of subgroups evaluated (excluding skipped)

Author(s)

Matthias Templ

References

Snoke, J., Raab, G. M., Nowok, B., Dibben, C., & Slavkovic, A. (2018). General and specific utility measures for synthetic data. *Journal of the Royal Statistical Society: Series A*, 181(3), 663–688.

See Also

`energy_distance`, `mmd`, `hellinger` for utility functions that can be passed as `utility_fun`; `regression_fidelity` and `contingency_fidelity` for other utility measures

Other utility: `chisq_utility()`, `ci_proximity()`, `contingency_fidelity()`, `copula_fidelity()`, `energy_distance()`, `gower()`, `hellinger()`, `mmd()`, `mqs()`, `pMSE()`, `plot.rumap()`, `propscore()`, `regression_fidelity()`, `rumap()`, `specks()`, `tail_fidelity()`, `tstr()`

Examples

```
set.seed(123)
n <- 200
X <- data.frame(
  group = sample(c("A", "B", "C"), n, replace = TRUE, prob = c(0.6, 0.3, 0.1)),
  x1 = rnorm(n),
  x2 = rnorm(n)
)
# Good synthetic for groups A and B, poor for C
Y <- X
Y$x1 <- Y$x1 + ifelse(Y$group == "C", 3, rnorm(n, 0, 0.1))
Y$x2 <- Y$x2 + ifelse(Y$group == "C", 3, rnorm(n, 0, 0.1))

result <- subgroup_utility(X, Y, group_var = "group", seed = 42)
print(result)
summary(result)

# Using synth_pair interface
pair <- structure(list(original = X, synthetic = Y), class = "synth_pair")
result2 <- subgroup_utility(pair, group_var = "group")


plot(result)
```

---

suda *SUDA - Special Uniques Detection Algorithm*

Description

Implements the Special Uniques Detection Algorithm (SUDA) to identify records that are unique on subsets of key variables. SUDA scores indicate the risk level of individual records based on their uniqueness patterns.

Usage

```
suda(X, ...)

## S3 method for class 'synth_pair'
suda(X, max_msu = NULL, data = c("synthetic", "original"), ...)

## Default S3 method:
suda(X, key_vars, max_msu = NULL, missing = NA, na.rm = TRUE, ...)
```

Arguments

X	data frame to assess, or a synth_pair object
...	additional arguments passed to methods (currently unused)

max_msu	integer, maximum size of Minimal Sample Uniques to consider (default: NULL, uses all variables up to length(key_vars))
data	character, which dataset to assess: "synthetic" (default) or "original". Only used by the synth_pair method.
key_vars	character vector of quasi-identifier variable names
missing	numeric, value to treat as missing (default: NA handling via na.rm)
na.rm	logical, remove records with NA in key variables (default: TRUE)

Details

SUDA (Elliot et al., 2002) identifies records that are unique on subsets of key variables. A Minimal Sample Unique (MSU) is a record that is unique on a set of variables, but not unique on any proper subset of those variables.

SUDA Score: The SUDA score for a record is the sum of contributions from all MSUs it belongs to, where smaller MSUs contribute more (they are more identifying):

$$SUDA_i = \sum_{j \in MSU_i} \frac{1}{2^{|MSU_j|-1}}$$

DIS Score (Data Intrusion Simulation): A normalized version of the SUDA score that rescales by the number of possible variable combinations, making scores more comparable across datasets. Note that this is a custom combinatorial normalisation, *not* the file-level DIS-SUDA intrusion-probability calibration of Elliot (2002), and should not be interpreted as the published DIS-SUDA score.

Interpretation:

- **SUDA = 0**: Record is not unique on any variable subset
- **SUDA > 0**: Record is unique on at least one variable subset
- **Higher SUDA**: More/smaller uniqueness patterns - higher risk

Records with high SUDA scores are at elevated risk of re-identification because they can be identified using fewer variables.

Value

An object of class "suda" containing:

- suda_scores: numeric vector of SUDA scores for each record
- dis_scores: numeric vector of Data Intrusion Simulation (DIS) scores
- msu_counts: number of MSUs each record contributes to
- n_msu: total number of Minimal Sample Uniques found
- msu_by_size: count of MSUs by size
- high_risk_records: indices of records with elevated SUDA scores
- summary_stats: summary statistics of SUDA scores

Author(s)

Matthias Templ

References

Elliot, M.J., Manning, A.M., Mayes, K., Gurd, J., & Bane, M. (2005). SUDA: A Program for Detecting Special Uniques. Joint UNECE/Eurostat Work Session on Statistical Data Confidentiality.

Manning, A.M., Haglin, D.J., & Keane, J.A. (2008). A Recursive Search Algorithm for Statistical Disclosure Assessment. *Data Mining and Knowledge Discovery*, 16(2), 165-196.

See Also

kanonymity for k-anonymity assessment, individual_risk for probabilistic risk measures

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), tcloseness(), top_at_risk()

Examples

```
# Create example data
set.seed(123)
data <- data.frame(
  age = sample(c("young", "middle", "old"), 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 100, replace = TRUE),
  education = sample(c("low", "medium", "high"), 100, replace = TRUE)
)

# Run SUDA analysis
result <- suda(data, key_vars = c("age", "gender", "region", "education"))
print(result)
summary(result)
plot(result)

# Identify high-risk records
high_risk <- which(result$suda_scores > quantile(result$suda_scores, 0.95))
data[high_risk, ]
```

---

summary.attacker_risk *Summary method for attacker_risk objects*

Description

Summary method for attacker_risk objects

Usage

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

Arguments

object	an object of class "attacker_risk"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

---

summary.chisq_utility *Summary method for chisq_utility objects*

Description

Summary method for chisq_utility objects

Usage

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

Arguments

object	an object of class "chisq_utility"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.ci_proximity *Summary method for ci_proximity objects*

Description

Summary method for ci_proximity objects

Usage

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

Arguments

object	an object of class "ci_proximity"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.compare_distributions_cont *Summary method for compare_distributions_cont objects*

Description

Summary method for compare_distributions_cont objects

Usage

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

Arguments

object	an object of class "compare_distributions_cont"
...	additional arguments (ignored)

Value

An object of class "summary.compare_distributions_cont".

summary.compare_feature_importance

Summary method for compare_feature_importance objects

Description

Summary method for compare_feature_importance objects

Usage

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

Arguments

object	an object of class "compare_feature_importance"
...	additional arguments (ignored)

Value

An object of class "summary.compare_feature_importance"

summary.contingency_fidelity

Summary method for contingency_fidelity objects

Description

Summary method for contingency_fidelity objects

Usage

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

Arguments

object	an object of class "contingency_fidelity"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.copula_fidelity *Summary method for copula_fidelity objects*

Description

Summary method for copula_fidelity objects

Usage

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

Arguments

object	an object of class "copula_fidelity"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.dcap *Summary method for dcap objects*

Description

Summary method for dcap objects

Usage

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

Arguments

object	an object of class "dcap"
...	additional arguments passed to the summary method

Value

A list of summary statistics for the corresponding object.

summary.dcr *Summary method for dcr objects*

Description

Summary method for dcr objects

Usage

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

Arguments

object	an object of class "dcr"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.delta_presence *Summary method for delta_presence objects*

Description

Summary method for delta_presence objects

Usage

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

Arguments

object	an object of class "delta_presence"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.denpca *Summary method for denpca objects*

Description

Summary method for denpca objects

Usage

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

Arguments

object	an object of class "denpca"
...	additional arguments (ignored)

Value

An object of class "summary.denpca".

summary.denratio *Summary method for denratio objects*

Description

Summary method for denratio objects

Usage

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

Arguments

object	an object of class "denratio"
...	additional arguments (ignored)

Value

An object of class "summary.denratio".

summary.disclosure_report *Summary method for disclosure_report objects*

Description

Summary method for disclosure_report objects

Usage

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

Arguments

object	an object of class "disclosure_report"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.disco *Summary method for disco objects*

Description

Summary method for disco objects

Usage

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

Arguments

object	an object of class "disco"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.domias *Summary method for domias objects*

Description

Summary method for domias objects

Usage

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

Arguments

object	an object of class "domias"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.drisk *Summary method for drisk objects*

Description

Summary method for drisk objects

Usage

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

Arguments

object	an object of class "drisk"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

`summary.energy_distance`

Summary method for energy_distance objects

Description

Summary method for energy_distance objects

Usage

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

Arguments

`object`	an object of class "energy_distance"
`...`	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

`summary.epsilon_identifiability`

Summary method for epsilon_identifiability objects

Description

Summary method for epsilon_identifiability objects

Usage

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

Arguments

`object`	an object of class "epsilon_identifiability"
`...`	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.gower *Summary method for gower objects*

Description

Summary method for gower objects

Usage

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

Arguments

object	an object of class "gower"
...	additional arguments (ignored)

Value

An object of class "summary.gower"

summary.hellinger *Summary method for hellinger objects*

Description

Summary method for hellinger objects

Usage

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

Arguments

object	an object of class "hellinger"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.hitting_rate *Summary method for hitting_rate objects*

Description

Summary method for hitting_rate objects

Usage

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

Arguments

object	an object of class "hitting_rate"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.ims *Summary method for ims objects*

Description

Summary method for ims objects

Usage

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

Arguments

object	an object of class "ims"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.individual_risk *Summary method for individual_risk objects*

Description

Summary method for individual_risk objects

Usage

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

Arguments

object	an object of class "individual_risk"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.kanonymity *Summary method for kanonymity objects*

Description

Summary method for kanonymity objects

Usage

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

Arguments

object	an object of class "kanonymity"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.ldiversityRisk *Summary method for ldiversityRisk objects*

Description

Summary method for ldiversityRisk objects

Usage

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

Arguments

object	an object of class "ldiversityRisk"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.linkability *Summary method for linkability objects*

Description

Summary method for linkability objects

Usage

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

Arguments

object	an object of class "linkability"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.mia *Summary method for mia objects*

Description

Summary method for mia objects

Usage

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

Arguments

object	an object of class "mia"
...	additional arguments (ignored)

Value

An object of class "summary.mia"

summary.missingCompare *Summary method for missingCompare objects*

Description

Summary method for missingCompare objects

Usage

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

Arguments

object	An object of class "missingCompare".
...	Additional arguments (ignored).

Value

A list of summary statistics for the corresponding object.

summary.mmd *Summary method for mmd objects*

Description

Summary method for mmd objects

Usage

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

Arguments

object	an object of class "mmd"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.mqs *Summary method for mqs objects*

Description

Summary method for mqs objects

Usage

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

Arguments

object	an object of class "mqs"
...	additional arguments (ignored)

Value

An object of class "summary.mqs"

summary.nnaa *Summary method for nnaa objects*

Description

Summary method for nnaa objects

Usage

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

Arguments

object	an object of class "nnaa"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.nndr *Summary method for nndr objects*

Description

Summary method for nndr objects

Usage

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

Arguments

object	an object of class "nndr"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.pMSE *Summary method for pMSE objects*

Description

Summary method for pMSE objects

Usage

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

Arguments

object	an object of class "pMSE"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.population_uniqueness *Summary method for population_uniqueness objects*

Description

Summary method for population_uniqueness objects

Usage

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

Arguments

object	an object of class "population_uniqueness"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

`summary.propscore` *Summary method for propscore objects*

Description

Summary method for propscore objects

Usage

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

Arguments

`object`	an object of class "propscore"
`...`	additional arguments passed to the summary method

Value

An object of class `summary.propscore` with the following components:

ps_score The propensity score statistic (pMSE).

ps_ratio Ratio of predictions below vs above the class ratio.

cr Class ratio (proportion of synthetic records in the combined set).

mean_ps_x Mean predicted propensity for original records.

mean_ps_y Mean predicted propensity for synthetic records.

kl KL divergence between propensity density estimates.

kl_bayes KL divergence in Bayes (log-ratio) space.

mean_ratio Mean density ratio.

sd_ratio Standard deviation of density ratio.

n_x Number of original records.

n_y Number of synthetic records.

method Method used for propensity estimation.

summary.rapid *Summary method for rapid objects*

Description

Summary method for rapid objects

Usage

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

Arguments

object	an object of class "rapid"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.rapid_cv *Summary method for rapid_cv objects*

Description

Summary method for rapid_cv objects

Usage

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

Arguments

object	an object of class "rapid_cv"
...	additional arguments (ignored)

Value

An object of class "summary.rapid_cv"

summary.rapid_test *Summary method for rapid_test objects*

Description

Summary method for rapid_test objects

Usage

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

Arguments

object	an object of class "rapid_test"
...	additional arguments (ignored)

Value

An object of class "summary.rapid_test"

summary.rapid_threshold *Summary method for rapid_threshold objects*

Description

Summary method for rapid_threshold objects

Usage

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

Arguments

object	an object of class "rapid_threshold"
...	additional arguments (ignored)

Value

An object of class "summary.rapid_threshold"

summary.recordLinkageRisk

Summary method for recordLinkageRisk

Description

Summary method for recordLinkageRisk

Usage

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

Arguments

object	object of class "recordLinkageRisk".
...	ignored.

Value

A list of summary statistics for the corresponding object.

summary.regression_fidelity

Summary method for regression_fidelity objects

Description

Summary method for regression_fidelity objects

Usage

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

Arguments

object	an object of class "regression_fidelity"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.rumap *Summary method for rumap objects*

Description

Summary method for rumap objects

Usage

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

Arguments

object	an object of class "rumap"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.singling_out *Summary method for singling_out objects*

Description

Summary method for singling_out objects

Usage

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

Arguments

object	an object of class "singling_out"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.specks *Summary method for specks objects*

Description

Summary method for specks objects

Usage

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

Arguments

object	an object of class "specks"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.subgroup_utility *Summary method for subgroup_utility objects*

Description

Summary method for subgroup_utility objects

Usage

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

Arguments

object	an object of class "subgroup_utility"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.suda *Summary method for suda objects*

Description

Summary method for suda objects

Usage

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

Arguments

object	an object of class "suda"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.synth_pair *Summary method for synth_pair objects*

Description

Summary method for synth_pair objects

Usage

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

Arguments

object	An object of class "synth_pair"
...	Additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.tail_fidelity *Summary method for tail_fidelity objects*

Description

Summary method for tail_fidelity objects

Usage

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

Arguments

object	an object of class "tail_fidelity"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.tcap *Summary method for tcap objects*

Description

Summary method for tcap objects

Usage

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

Arguments

object	an object of class "tcap"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.tcloseness *Summary method for tcloseness objects*

Description

Summary method for tcloseness objects

Usage

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

Arguments

object	an object of class "tcloseness"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

summary.weap *Summary method for weap objects*

Description

Summary method for weap objects

Usage

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

Arguments

object	an object of class "weap"
...	additional arguments (ignored)

Value

A list of summary statistics for the corresponding object.

synth_pair *Create a Synthetic Data Comparison Pair*

Description

Creates a container object that holds both the original and synthetic datasets along with metadata for consistent analysis across multiple risk and utility measures.

Usage

```
synth_pair(
  original,
  synthetic,
  key_vars = NULL,
  target_var = NULL,
  weight_original = NULL,
  weight_synthetic = NULL,
  vars = NULL,
  holdout = NULL,
  source = "custom",
  metadata = list()
)
```

Arguments

original	Original/training data (data.frame or data.table)
synthetic	Synthetic/anonymized data (data.frame or data.table)
key_vars	Character vector of quasi-identifier variables (for attribution-based risk). Required for dcap, tcap, disco, weap.
target_var	Character string naming the sensitive target variable (for CAP metrics). Required for dcap, tcap, disco.
weight_original	Character string naming the weight variable in original data, or NULL.
weight_synthetic	Character string naming the weight variable in synthetic data, or NULL.
vars	Character vector of variables to use for utility comparisons. If NULL (default), all common variables between original and synthetic are used.
holdout	Optional data.frame for distance-based risk measures (dcr, nndr). If NULL, these functions will use internal splitting.
source	Character string describing data source (e.g., "synthpop", "simPop", "custom").
metadata	Named list of additional metadata to store with the pair.

Details

The synth_pair class provides a convenient container for synthetic data evaluation workflows. Instead of repeatedly passing the same parameters to multiple functions, you can create a single object that carries all the context needed for analysis.

Many functions in this package have methods for synth_pair objects, allowing simplified calls like dcap(pair) instead of dcap(X, Y, key_vars = ..., target_var = ...).

For users of synthpop, simPop, or sdcMicro packages, convenience constructors are available:

- from_synthpop: Create from synthpop's synds objects
- from_simPop: Create from simPop's simPopObj objects
- from_sdcMicro: Create from sdcMicro's sdcMicroObj objects

Value

An object of class "synth_pair" containing:

- original: the original dataset
- synthetic: the synthetic dataset
- key_vars, target_var: for attribution-based risk measures
- weight_original, weight_synthetic: weight variable names
- vars: variables for utility comparison
- cat_vars, num_vars: auto-detected categorical and numeric variables
- holdout: optional holdout data for distance-based measures
- source: data source identifier
- metadata: additional metadata
- n_original, n_synthetic: dataset sizes

Author(s)

Matthias Templ

See Also

from_synthpop, from_simPop, from_sdcMicro for package-specific constructors, dcap, tcap, ims, hellinger for functions with synth_pair methods

Examples

```
# Create example data
set.seed(123)
original <- data.frame(
  age = sample(20:70, 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE),
  income = sample(c("low", "medium", "high"), 200, replace = TRUE)
)
```

```
synthetic <- data.frame(
  age = sample(20:70, 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 200, replace = TRUE),
  income = sample(c("low", "medium", "high"), 200, replace = TRUE)
)

# Create a synth_pair object
pair <- synth_pair(
  original = original,
  synthetic = synthetic,
  key_vars = c("age", "gender", "region"),
  target_var = "income"
)

print(pair)

# Now functions can be called directly on the pair
# dcap(pair)
# ims(pair)
# hellinger(pair)
```

`systemAnonymityLevel` *System Anonymity Level*

Description

Computes the system anonymity level based on the entropy of sender-receiver combinations in anonymous communication networks, normalized by the number of users.

Usage

```
systemAnonymityLevel(A)
```

Arguments

`A`	A square binary matrix representing sender-receiver adjacency, where `A[i, j] = 1` indicates that sender `i` can send to receiver `j`.

Details

The system anonymity level quantifies the uncertainty an adversary faces in identifying the correct sender-receiver mapping. It is calculated as:

$$privSAL = \frac{1}{\log_2(|U|!)} H(p(x))$$

, where $H(p(x))$ is the Shannon entropy of the adversary's estimated distribution over sender-receiver combinations and $|U|$ is the number of users.

The probability $p(x)$ is computed as $|E|/per(A)$, where $per(A)$ is the matrix permanent of the adjacency matrix A, and $|E|$ is the size of the equivalence class.

Value

A numeric value between 0 and 1 indicating the system anonymity level. A value of 1 indicates maximum anonymity.

Note

Computing the permanent of a matrix is computationally intensive (NP-hard). This implementation is feasible only for small matrices (typically $<= 6x6$).

Author(s)

Matthias Templ

References

I. Wagner and D. Eckhoff (2018). Technical Privacy Metrics: A Systematic Survey. *ACM Computing Surveys*, 51(3), Article 57. Section 5.2.8.

See Also

Other information-theory: Entropy, EntropyMeasures, information_surprisal(), max_info_leakage(), mutualInformation(), positive_information_disclosure(), privacy_score()

Examples

```
# Example: Anonymity in a simple 3x3 anonymous messaging network
A <- matrix(c(
  1, 1, 0,
  1, 0, 1,
  0, 1, 1
), nrow = 3, byrow = TRUE)
systemAnonymityLevel(A)

# Example for privacy evaluation: Higher values indicate less certainty
# about communication patterns in the system, and thus greater privacy.
A2 <- matrix(1, nrow = 3, ncol = 3) # Fully connected
systemAnonymityLevel(A2) # Should return 1 (maximum uncertainty)
```

tail_fidelity *Tail Fidelity — Tail Preservation Utility Measure*

Description

Assesses how well synthetic data preserves the extreme values (tails) of the original distribution. Combines QQ tail divergence (headline metric), Jensen-Shannon divergence in the tail region, and an optional Hill tail index estimator.

Usage

```
tail_fidelity(X, ...)

## S3 method for class 'synth_pair'
tail_fidelity(X, ...)

## Default S3 method:
tail_fidelity(
  X,
  Y,
  vars = NULL,
  percentile = 95,
  tails = c("both", "upper", "lower"),
  hill = FALSE,
  na.rm = TRUE,
  ...
)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods (currently unused)
Y	A data.frame or data.table containing the synthetic/anonymized dataset.
vars	Character vector of numeric variable names to compare. If NULL (default), all common numeric variables are used; categoricals are skipped with a message.
percentile	Numeric, the percentile threshold defining the tail region. Default 95, meaning the upper 5 percent and lower 5 percent of observations.
tails	Character, which tails to assess. One of "both" (default), "upper", or "lower".
hill	Logical, whether to include the Hill tail index estimator. Default FALSE. Only meaningful for positive-valued data (upper tail).
na.rm	Logical, whether to remove NA values. Default TRUE.

Details

Three complementary metrics are computed per numeric variable:

1. QQ tail divergence (headline metric): Quantiles at 50 evenly-spaced probability levels in the tail region are compared. The mean absolute difference is normalized by the IQR of the original variable, making it scale-invariant. For `tails = "both"`, the upper and lower divergences are averaged.

2. Density ratio in tail region: Observations beyond the percentile threshold are extracted and compared via Jensen-Shannon divergence using `densitydiff_1d_num`. If fewer than 20 observations fall in the tail, `NA` is returned.

3. Hill estimator (optional, `hill = TRUE`): For positive-valued upper-tail data the Hill estimator of the tail index is computed:

$$\hat{\alpha} = \left(\frac{1}{k} \sum_{i=1}^{k} \log \frac{x_{(i)}}{x_{(k+1)}} \right)^{-1}$$

where $x_{(1)} \geq \cdots \geq x_{(n)}$ are order statistics and $k = \lfloor \sqrt{n} \rfloor$. The absolute difference between original and synthetic Hill estimates is reported.

Value

An object of class `"tail_fidelity"` containing:

- `qq_divergence`: mean QQ tail divergence across all variables
- `utility_score`: `exp(-qq_divergence)`, in $[0, 1]$, 1 = identical tails
- `per_variable`: data.frame with columns: variable, qq_tail_div, jsd_tail, and optionally hill_diff
- `percentile`: the percentile threshold used
- `tails`: which tails were assessed
- `n_vars`: number of numeric variables analysed
- `vars`: variable names used
- `n_X`: number of rows in X (after NA removal)
- `n_Y`: number of rows in Y (after NA removal)
- `hill`: whether Hill estimator was computed

Author(s)

Matthias Templ

References

Hill, B. M. (1975). A simple general approach to inference about the tail of a distribution. The Annals of Statistics, 3(5), 1163-1174.

See Also

densitydiff_1d_num for density comparison, compare_wasserstein for Wasserstein distance, energy_distance for multivariate energy distance

Other utility: chisq_utility(), ci_proximity(), contingency_fidelity(), copula_fidelity(), energy_distance(), gower(), hellinger(), mmd(), mqs(), pMSE(), plot.rumap(), propscore(), regression_fidelity(), rumap(), specks(), subgroup_utility(), tstr()

Examples

```
set.seed(42)
# Original data with heavy tails
X <- data.frame(
  income = rlnorm(500, meanlog = 10, sdlog = 1),
  score = rnorm(500, 100, 15)
)

# Good synthetic (same distribution)
Y_good <- data.frame(
  income = rlnorm(500, meanlog = 10, sdlog = 1),
  score = rnorm(500, 100, 15)
)

# Poor synthetic (tails trimmed)
Y_poor <- data.frame(
  income = pmin(pmax(rlnorm(500, meanlog = 10, sdlog = 1), 5000), 80000),
  score = pmin(pmax(rnorm(500, 100, 15), 70), 130)
)

result_good <- tail_fidelity(X, Y_good)
print(result_good)

result_poor <- tail_fidelity(X, Y_poor)
print(result_poor)


# With Hill estimator (for positive data)
result_hill <- tail_fidelity(X, Y_good, hill = TRUE)
summary(result_hill)

# Assess only upper tail
result_upper <- tail_fidelity(X, Y_poor, tails = "upper")
print(result_upper)
```

---

tcap *Targeted Correct Attribution Probability (TCAP)*

Description

Computes the Targeted Correct Attribution Probability for assessing attribute disclosure risk in synthetic data. TCAP measures, for each original record, the probability that an intruder correctly infers a sensitive target variable by matching on quasi-identifier (key) attributes.

Usage

```
tcap(X, ...)

## S3 method for class 'synth_pair'
tcap(X, ...)

## Default S3 method:
tcap(
  X,
  Y,
  key_vars,
  target_var,
  method = c("exact", "gower"),
  gower_threshold = 0.1,
  na.rm = TRUE,
  ...
)
```

Arguments

X	data frame of original data
...	additional arguments passed to methods (currently unused)
Y	data frame of synthetic data
key_vars	character vector of quasi-identifier variable names
target_var	character, name of the sensitive target variable
method	character, matching method: "exact" or "gower" (default: "exact")
gower_threshold	numeric, maximum Gower distance for a match when method="gower" (default: 0.1)
na.rm	logical, remove records with NA in key or target (default: TRUE)

Details

This function computes Correct Attribution Probability (CAP) metrics developed by Elliot (2014) and Taub et al. (2018) to assess attribute disclosure risk in synthetic data.

For each original record i with key values k_i and target value t_i:

$$CAP_i = \frac{|\{j \in Y : keys_j = k_i \wedge target_j = t_i\}|}{|\{j \in Y : keys_j = k_i\}|}$$

The function returns three summary metrics:

- **tcap_mean**: Mean CAP across all matched records. Equivalent to synthpop's CAPd. This is the standard measure for overall disclosure risk.
- **tcap_max**: Maximum CAP value (worst-case individual risk). A value of 1 means at least one record has all its synthetic matches with the correct target value.
- **tcap_certain**: Percentage of matched records where the key uniquely determines the target in BOTH the original AND synthetic data. Equivalent to synthpop's TCAP. These records are at "certain" disclosure risk because the synthetic data perfectly reveals their target value.

Interpretation:

- CAP = 0: No synthetic matches have the correct target (low risk)
- CAP = 1: All synthetic matches have the correct target (high risk)
- CAP close to baseline: No additional risk beyond random guessing

Value

An object of class "tcap" containing:

- tcap_scores: numeric vector of per-record CAP values
- tcap_mean: mean CAP across all matched records (equivalent to synthpop's CAPd)
- tcap_max: maximum CAP (worst-case individual risk)
- tcap_median: median CAP
- tcap_certain: percentage of matched records at "certain" disclosure risk (key uniquely determines target in both original and synthetic data, equivalent to synthpop's TCAP)
- n_certain: count of records at certain disclosure risk
- n_matches: number of synthetic matches per original record
- n_matched: number of original records with at least one match
- n_unmatched: number of original records with no matches
- baseline: baseline probability (modal target frequency in Y)
- key_vars, target_var, method: input parameters

Baseline computation

The baseline is computed as the maximum target category frequency in the synthetic data, representing the best guess without key information. **Note:** This differs from synthpop's approach which uses the original data. See `dcap` for detailed discussion.

Gower threshold selection

When `method = "gower"`, records within `gower_threshold` distance are considered matches. The default of 0.1 works well for most datasets. See `dcap` for guidelines on choosing this value.

Author(s)

Matthias Templ

References

Elliot, M. (2014). Final Report on the Disclosure Risk Associated with the Synthetic Data Produced by the SYLLS Team. Report 2015-2.

Taub, J., Elliot, M., Pampaka, M., & Smith, D. (2018). Differential Correct Attribution Probability for Synthetic Data: An Exploration. *Privacy in Statistical Databases*, 122-137.

See Also

dcap for the differential measure comparing to baseline, weap for within equivalence class attribution probability

Other attribution-risk: dcap(), disco(), weap()

Examples

```
# Create example data
set.seed(42)
X <- data.frame(
  age = sample(20:60, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  region = sample(c("N", "S", "E", "W"), 100, replace = TRUE),
  disease = sample(c("none", "A", "B"), 100, replace = TRUE,
                   prob = c(0.7, 0.2, 0.1))
)
# Synthetic version
Y <- X
Y$disease <- sample(Y$disease)  # Shuffle target

# Compute TCAP
result <- tcap(X, Y,
               key_vars = c("age", "gender", "region"),
               target_var = "disease")
print(result)
summary(result)
plot(result)


# Example 2: Using SD2011 dataset with synthpop-generated synthetic data
# This follows the synthpop::disclosure example pattern
# Requires: synthpop package

if (requireNamespace("synthpop", quietly = TRUE)) {

# Load SD2011 data and select variables (as in ?synthpop::disclosure)
ods <- synthpop::SD2011[, c("sex", "age", "edu", "marital", "income")]

# Convert income to categorical (7 groups), handling special NA code
odsF <- synthpop::numtocat.syn(ods, numtocat = "income",
                               catgroups = 7,
                               cont.na = list(income = -8))
original_data <- odsF$data
```

```
# Generate synthetic data using CART method
synth_obj <- synthpop::syn(original_data,
                           method = "ctree",
                           seed = 75,
                           m = 1,
                           k = 1000,
                           print.flag = FALSE)
synthetic_data <- synth_obj$syn

# Compute TCAP: Can we infer income from demographic keys?
tcap_result <- tcap(original_data, synthetic_data,
                    key_vars = c("sex", "age", "edu", "marital"),
                    target_var = "income")

# View results
print(tcap_result)
summary(tcap_result)

# Compare with synthpop's disclosure function
disc_sp <- synthpop::disclosure(synth_obj, original_data,
                                keys = c("sex", "age", "edu", "marital"),
                                target = "income",
                                print.flag = FALSE)

# Compare with synthpop disclosure metrics
# Note: Core CAP algorithm matches exactly (verified with controlled data).
# Differences with real data are due to NA handling:
# - riskutility: removes NA records, computes mean over matched only
# - synthpop: uses all records with package-specific NA handling
cat("\nComparison with synthpop:\n")
cat("riskutility mean TCAP:", round(tcap_result$tcap_mean * 100, 2), "%\n")
cat("  (over", tcap_result$n_matched, "matched of", tcap_result$n_total, "complete cases)\n")
cat("synthpop CAPd:", round(disc_sp$allCAPs$CAPd, 2), "%\n")

# Plot the TCAP distribution
plot(tcap_result, which = 1:2)
}
```

`tcloseness` *t-Closeness Assessment*

Description

Measures t-closeness of a dataset, which extends l-diversity by requiring that the distribution of a sensitive attribute within each equivalence class is close to the overall distribution, using Earth Mover's Distance (EMD).

Usage

```
tcloseness(X, ...)

## S3 method for class 'synth_pair'
tcloseness(X, t = 0.2, data = c("synthetic", "original"), ...)

## Default S3 method:
tcloseness(X, key_vars, sensitive_var, t = 0.2, na.rm = TRUE, ...)
```

Arguments

X	data frame to assess, or a synth_pair object
...	additional arguments passed to methods (currently unused)
t	numeric, the t-closeness threshold (default: 0.2)
data	character, which dataset to assess: "synthetic" (default) or "original". Only used by the synth_pair method.
key_vars	character vector of quasi-identifier variable names
sensitive_var	character, name of the sensitive attribute
na.rm	logical, remove records with NA values (default: TRUE)

Details

t-Closeness (Li et al., 2007) addresses limitations of l-diversity by requiring that the distribution of a sensitive attribute within each equivalence class is close to the attribute's overall distribution.

The distance is measured using Earth Mover's Distance (EMD), also known as the Wasserstein distance:

Numeric attributes: EMD is computed as the normalized sum of absolute CDF differences over ordered unique values:

$$EMD = \frac{1}{m-1} \sum_{i=1}^{m} |F_{EC}(x_i) - F(x_i)|$$

where m is the number of unique values, F_{EC} is the empirical CDF within the equivalence class, and F is the overall empirical CDF. This CDF-difference form corresponds to the ordered-distance EMD of Li et al. (2007) up to the choice of normalisation (here $1/(m-1)$ over the observed support); it is a close approximation rather than the exact published per-rank-weighted quantity.

Categorical attributes: EMD is the variational distance (half the L1 distance between distributions):

$$EMD = \frac{1}{2} \sum_{i} |p_{EC,i} - p_i|$$

Interpretation:

- **t close to 0**: EC distributions nearly identical to overall
- **t = 0.2**: Common threshold for reasonable protection
- **t close to 1**: Weak protection, distributions may differ greatly

Value

An object of class "tcloseness" containing:

- t_achieved: maximum EMD across all equivalence classes
- t_threshold: user-specified t
- satisfies_t: logical, whether data satisfies t-closeness
- n_violating: number of records in violating equivalence classes
- pct_violating: percentage of records violating t-closeness
- n_records: total number of records
- n_ec: number of equivalence classes
- per_ec: data.frame with key, size, emd, violates_t per equivalence class
- sensitive_type: "numeric" or "categorical"
- key_vars, sensitive_var: input parameters

Author(s)

Matthias Templ

References

Li, N., Li, T. & Venkatasubramanian, S. (2007). t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. *IEEE 23rd International Conference on Data Engineering*, 106–115. doi:10.1109/ICDE.2007.367856

See Also

`kanonymity` for k-anonymity assessment, `ldiversity` for l-diversity assessment

Other privacy-models: `attacker_risk()`, `delta_presence()`, `disclosure_report()`, `domias()`, `drisk()`, `epsilon_identifiability()`, `individual_risk()`, `inspect_record()`, `kanonymity()`, `ldiversity()`, `linkability()`, `merge_per_record()`, `mia_classifier()`, `population_uniqueness()`, `recordLinkage()`, `risk_by_group()`, `singling_out()`, `suda()`, `top_at_risk()`

Examples

```
# Create example data
set.seed(123)
data <- data.frame(
  age = sample(c("young", "middle", "old"), 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S"), 200, replace = TRUE),
  salary = rnorm(200, mean = 50000, sd = 15000)
)

# Check t-closeness with numeric sensitive attribute
result <- tcloseness(data,
                     key_vars = c("age", "gender", "region"),
                     sensitive_var = "salary",
```

```
                      t = 0.2)
print(result)
summary(result)
plot(result)

# Check with categorical sensitive attribute
data$disease <- sample(c("healthy", "cold", "flu", "covid"), 200,
                       replace = TRUE, prob = c(0.5, 0.2, 0.2, 0.1))
result2 <- tcloseness(data,
                      key_vars = c("age", "gender", "region"),
                      sensitive_var = "disease",
                      t = 0.3)
print(result2)
```

`top_at_risk` *Return Highest-Risk Records*

Description

Returns the top-n riskiest records from a `recordLinkageRisk` object with their per-record diagnostics.

Usage

```
top_at_risk(x, ...)

## S3 method for class 'recordLinkageRisk'
top_at_risk(x, n = 10, data = NULL, ...)
```

Arguments

`x`	object of class `"recordLinkageRisk"`.
`...`	ignored.
`n`	integer, number of records to return (default 10).
`data`	optional data frame (original or synthetic depending on direction) whose key-variable columns are appended.

Value

A data frame with per-record diagnostics for the top-n riskiest records, including a `record_id` column.

Author(s)

Matthias Templ, Oscar Thees

See Also

Other privacy-models: attacker_risk(), delta_presence(), disclosure_report(), domias(), drisk(), epsilon_identifiability(), individual_risk(), inspect_record(), kanonymity(), ldiversity(), linkability(), merge_per_record(), mia_classifier(), population_uniqueness(), recordLinkage(), risk_by_group(), singling_out(), suda(), tcloseness()

---

tstr *Train on Synthetic, Test on Real (TSTR) Utility Measure*

Description

Evaluates synthetic data utility by comparing the predictive performance of models trained on synthetic data versus models trained on original data. A TSTR ratio near 1 indicates that the synthetic data preserves the predictive structure of the original data.

Usage

```
tstr(X, ...)

## S3 method for class 'synth_pair'
tstr(X, ...)

## Default S3 method:
tstr(
  X,
  Y,
  target_var = NULL,
  model = c("glm", "rf", "rpart", "gbm"),
  test_fraction = 0.3,
  test_data = NULL,
  na.rm = TRUE,
  seed = NULL,
  ...
)

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

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

## S3 method for class 'summary.tstr'
print(x, ...)

## S3 method for class 'tstr'
plot(x, y = NULL, ..., which = 1)
```

Arguments

X	A data.frame or data.table containing the original dataset.
...	additional arguments passed to methods (currently unused)
Y	A data.frame or data.table containing the synthetic/anonymized dataset.
target_var	Character vector of target variable name(s). Required. If multiple targets are given, TSTR is computed for each and results are averaged.
model	Character, the model type to use. One of "glm" (default), "rf" (random forest via ranger), "rpart" (CART), or "gbm" (gradient boosting). Requires the **caret** package and model-specific packages (**ranger**, **rpart**, **xgboost**).
test_fraction	Numeric, fraction of original data held out for testing. Default 0.3. Ignored if test_data is provided.
test_data	Optional data.frame of pre-split test data. If provided, test_fraction is ignored and the full original data is used for TRTR training.
na.rm	Logical, whether to remove rows with NA values. Default TRUE.
seed	Integer, random seed for reproducibility. Default NULL.
x	an object of class "tstr"
object	an object of class "tstr"
y	not used
which	integer, which plot: 1 = grouped bar chart comparing TRTR vs TSTR performance (per target if multiple)

Details

The TSTR protocol evaluates whether a synthetic dataset preserves the predictive relationships in the original data:

1. Split original data into training (1 - test_fraction) and test (test_fraction) sets.
2. **TRTR** (baseline): Train on original training set, evaluate on test set.
3. **TSTR**: Train on full synthetic data, evaluate on the same test set.
4. Compute tstr_ratio = TSTR_performance / TRTR_performance.

For classification targets (factor/character), the AUC (Area Under the ROC Curve) is computed via the Mann-Whitney U statistic. For regression targets (numeric), R-squared is used.

Models are trained without cross-validation (caret::trainControl(method = "none")) to ensure training uses the full training set.

Value

An object of class "tstr" containing:

- tstr_ratio: ratio of TSTR to TRTR performance (1.0 = perfect)
- utility_score: clamped ratio in $[0, 1]$ (higher = better)
- tstr_performance: model performance trained on synthetic data

- `trtr_performance`: baseline performance trained on original data
- `metric`: performance metric used (`"R2"` or `"AUC"`)
- `model`: model type used
- `target_var`: target variable name(s)
- `n_train_orig`: number of original training observations
- `n_test`: number of test observations
- `n_synth`: number of synthetic observations
- `test_fraction`: fraction used for test split
- `per_target`: list of per-target results (when multiple targets)

Author(s)

Matthias Templ

References

Esteban, C., Hyland, S. L., and Ratsch, G. (2017). Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs. arXiv:1706.02633.

See Also

`propscore` for propensity score utility, `compare_model_performance` for cross-validated model comparison

Other utility: `chisq_utility()`, `ci_proximity()`, `contingency_fidelity()`, `copula_fidelity()`, `energy_distance()`, `gower()`, `hellinger()`, `mmd()`, `mqs()`, `pMSE()`, `plot.rumap()`, `propscore()`, `regression_fidelity()`, `rumap()`, `specks()`, `subgroup_utility()`, `tail_fidelity()`

Examples

```
set.seed(42)
n <- 200
X <- data.frame(
  x1 = rnorm(n),
  x2 = rnorm(n)
)
X$y <- 2 * X$x1 + 0.5 * X$x2 + rnorm(n, sd = 0.5)

# Good synthetic data (same relationship)
Y_good <- data.frame(
  x1 = rnorm(n),
  x2 = rnorm(n)
)
Y_good$y <- 2 * Y_good$x1 + 0.5 * Y_good$x2 + rnorm(n, sd = 0.5)


if (requireNamespace("caret", quietly = TRUE)) {
  result <- tstr(X, Y_good, target_var = "y", seed = 1)
  print(result)
```

```
  summary(result)
}
```

---

weap *Within Equivalence Class Attribution Probability (WEAP)*

Description

Computes the Within Equivalence Class Attribution Probability for synthetic data records. WEAP identifies potentially disclosive synthetic records by measuring the conditional probability of the target value given the key variables within the synthetic dataset.

Usage

```
weap(X, ...)

## S3 method for class 'synth_pair'
weap(X, ...)

## Default S3 method:
weap(X, key_vars, target_var, na.rm = TRUE, ...)
```

Arguments

X	data frame of synthetic data
...	additional arguments passed to methods (currently unused)
key_vars	character vector of quasi-identifier variable names
target_var	character, name of the sensitive target variable
na.rm	logical, remove records with NA in key or target (default: TRUE)

Details

WEAP (Within Equivalence Class Attribution Probability) is computed for each record in the synthetic dataset. An equivalence class is the set of all records sharing the same key variable values.

For each synthetic record j with key values k_j and target value t_j:

$$WEAP_j = \frac{|\{i \in Y : keys_i = k_j \wedge target_i = t_j\}|}{|\{i \in Y : keys_i = k_j\}|}$$

Records with WEAP = 1 are potentially disclosive because the target value is uniquely determined by the key values within the synthetic data.

This measure is used to identify risky synthetic records before release. High WEAP values indicate that if an intruder knows the key values, they can confidently infer the target value.

Value

An object of class "weap" containing:

- weap_scores: numeric vector of per-record WEAP values in Y
- weap_mean: mean WEAP across all records
- equivalence_classes: data frame with equivalence class information
- n_disclosive: number of records with WEAP = 1 (potentially disclosive)
- pct_disclosive: percentage of potentially disclosive records
- n_unique_keys: number of unique key combinations
- key_vars, target_var: input parameters

Author(s)

Matthias Templ

References

Elliot, M. (2014). Final Report on the Disclosure Risk Associated with the Synthetic Data Produced by the SYLLS Team. Report 2015-2.

Taub, J., Elliot, M., Pampaka, M., & Smith, D. (2018). Differential Correct Attribution Probability for Synthetic Data: An Exploration. *Privacy in Statistical Databases*, 122-137.

See Also

`tcap` for targeted correct attribution probability, `dcap` for differential correct attribution probability, `disco` for identifying disclosive records

Other attribution-risk: `dcap()`, `disco()`, `tcap()`

Examples

```
# Create synthetic data
set.seed(123)
X <- data.frame(
  age = sample(c("young", "middle", "old"), 200, replace = TRUE),
  gender = sample(c("M", "F"), 200, replace = TRUE),
  region = sample(c("N", "S"), 200, replace = TRUE),
  income = sample(c("low", "medium", "high"), 200, replace = TRUE)
)

# Compute WEAP
result <- weap(X,
               key_vars = c("age", "gender", "region"),
               target_var = "income")
print(result)
summary(result)
plot(result)

# Identify disclosive records
```

```
disclosive_idx <- which(result$weap_scores == 1)
if (length(disclosive_idx) > 0) {
  head(X[disclosive_idx, ])
}
```

Index