Package 'slideimp'

January 7, 2026

Type Package

Title Numeric Matrices K-NN and PCA Imputation

Version 0.5.4

Description Fast k-nearest neighbors (K-NN) and principal component analysis (PCA) imputation algorithms for missing values in high-dimensional numeric matrices, i.e., epigenetic data. For extremely high-dimensional data with ordered features, a sliding window approach for K-NN or PCA imputation is provided. Additional features include group-wise imputation (e.g., by chromosome), hyperparameter tuning with repeated cross-validation, multi-core parallelization, and optional subset imputation. The K-NN algorithm is described in: Hastie, T., Tibshirani, R., Sherlock, G., Eisen, M., Brown, P. and Botstein, D. (1999) ``Imputing Missing Data for Gene Expression Arrays''. The PCA imputation is an optimized version of the imputePCA() function from the 'missMDA' package described in: Josse, J. and Husson, F. (2016) <doi:10.18637/jss.v070.i01> ``missMDA: A Package for Handling Missing Values in Multivariate Data Analysis''.

License GPL (>= 2)

URL https://github.com/hhp94/slideimp

BugReports https://github.com/hhp94/slideimp/issues

Depends R (>= 4.1.0)

Imports bigmemory, checkmate, collapse, mirai, purrr, Rcpp, stats, tibble

Suggests carrier, FactoMineR, knitr, missMDA, rlang, rmarkdown, testthat (>= 3.0.0)

LinkingTo mlpack, Rcpp, RcppArmadillo, RcppEnsmallen

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

NeedsCompilation yes

Author Hung Pham [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-8271-9355>)

Maintainer Hung Pham <amser.hoanghung@gmail.com>

Repository CRAN

Date/Publication 2026-01-07 09:20:02 UTC

Contents

`col_vars` *Calculate Matrix Column Variance*

Description

Computes the sample variance for each column of a numeric matrix

Usage

```
col_vars(mat, cores = 1)
```

Arguments

`mat`	A numeric matrix.
`cores`	Number of cores to use for parallel computation. Defaults to 1.

Details

Variances for columns with one unique value after dropping `NA` are set to `NA`.

Value

A named numeric vector of column variances

Examples

```
col_vars(t(khanmiss1))
```

`compute_metrics` *Compute Prediction Accuracy Metrics*

Description

Computes prediction accuracy metrics for results from `tune_imp()`.

Usage

```
compute_metrics(results, metrics = c("mae", "rmse", "rsq"))
```

Arguments

`results`	A tibble from `tune_imp()` containing a `result` column with tibbles that have `truth` and `estimate` columns.
`metrics`	A character vector of metric names to compute. Defaults to `c("mae", "rmse", "rsq")`. Also available: `"mape"`, `"bias"`, `"calc_rsq_trad"`.

Details

For alternative or faster metrics, see the {yardstick} package.

Value

A tibble with the original parameters and unnested metrics (`.metric`, `.estimator`, `.estimate`).

Examples

```
data(khanmiss1)
set.seed(1234)
results <- tune_imp(
  obj = t(khanmiss1),
  parameters = data.frame(k = 10),
  .f = "knn_imp",
  rep = 1,
  num_na = 20
)

compute_metrics(results)
```

`group_features` *Group Features for Imputation*

Description

Groups matrix columns (features) based on a provided grouping data.frame, optionally preparing parameters for K-NN or PCA imputation. This function organizes features into groups, handles imputation of only a subset of features, and can pad groups to meet a minimum size.

Usage

```
group_features(
  obj,
  features_df,
  k = NULL,
  ncp = NULL,
  subset = NULL,
  min_group_size = 0,
  seed = NULL
)
```

Arguments

`obj`	A numeric matrix with **samples in rows** and **features in columns**.
`features_df`	A data.frame with exactly two columns: `feature_id` and `group`. Maps feature identifiers to their respective groups. No missing values or duplicate `feature_id` values are allowed.
`k`	Integer or `NULL`. If specified, prepares parameters for K-NN imputation with `k` neighbors. Cannot be used together with `ncp`.
`ncp`	Integer or `NULL`. If specified, prepares parameters for PCA imputation with `ncp` principal components. Cannot be used together with `k`.
`subset`	Character vector of column names or integer vector of column indices specifying which columns to impute.
`min_group_size`	Integer (default 0). Minimum number of features per group. If a group has fewer features, additional features are randomly sampled from remaining columns to meet this threshold.
`seed`	Numeric or `NULL`. Random seed for reproducibility when sampling for `min_group_size` padding.

Value

A `tibble::tibble()` with columns:

- **features**: A list-column containing character vectors of feature column names to impute
- **aux**: A list-column containing character vectors of auxiliary column names used for imputation but not imputed themselves. Omitted if all elements are `NULL`
- **parameters**: A list-column containing group-specific parameters if `k` or `ncp` are specified

See Also

group_imp()

Examples

```
sim_obj <- sim_mat(perc_col_NA = 1)

obj <- t(sim_obj$input)
obj_meta <- sim_obj$group_feature

# group `obj` based on the metadata
head(obj_meta)

# create `group_df` which can then be used for `group_imp`. We can specify
# `k` for K-NN imputation and subset here as well.
group_df <- group_features(
  obj,
  obj_meta,
  subset = sample(obj_meta$feature_id, size = 10),
  k = 10
)
group_df

imputed_obj <- group_imp(obj, group_df)
imputed_obj
```

---

group_imp *Grouped K-NN or PCA Imputation*

Description

K-NN or PCA imputation by groups, such as chromosomes, flanking columns, or clusters identified by column clustering techniques.

Usage

```
group_imp(
  obj,
  group,
  k = NULL,
  colmax = NULL,
  knn_method = NULL,
  post_imp = NULL,
  dist_pow = NULL,
  tree = NULL,
  cores = 1,
  ncp = NULL,
```

```
  scale = NULL,
  pca_method = NULL,
  coeff.ridge = NULL,
  threshold = NULL,
  row.w = NULL,
  seed = NULL,
  nb.init = NULL,
  maxiter = NULL,
  miniter = NULL,
  .progress = TRUE
)
```

Arguments

`obj`	A numeric matrix with **samples in rows** and **features in columns**.
`group`	Preferably created by `group_features()`. A data.frame with columns: • **features**: A list-column containing character vectors of feature column names to impute • **aux**: (Optional) A list-column containing character vectors of auxiliary column names used for imputation but not imputed themselves • **parameters**: (Optional) A list-column containing group-specific parameters
`k`	Number of nearest neighbors for imputation. 10 is a good starting point.
`colmax`	A number from 0 to 1. Threshold of missing data above which K-NN imputation is skipped.
`knn_method`	Either "euclidean" (default) or "manhattan". Distance metric for nearest neighbor calculation.
`post_imp`	Whether to impute remaining missing values (those that failed K-NN imputation) using column means (default = `TRUE`).
`dist_pow`	The amount of penalization for further away nearest neighbors in the weighted average. `dist_pow = 0` (default) is the simple average of the nearest neighbors.
`tree`	Either `NULL` (default, brute-force K-NN), "ball", or "kd" to find nearest neighbors using the `{mlpack}` ball-tree or kd-tree algorithms.
`cores`	Controls the number of cores to parallelize over for K-NN imputation only. To setup parallelization for PCA imputation, use `mirai::daemons()`.
`ncp`	integer corresponding to the number of components used to to predict the missing entries
`scale`	boolean. By default TRUE leading to a same weight for each variable
`pca_method`	"regularized" by default or "EM".
`coeff.ridge`	1 by default to perform the regularized pca_imp (imputePCA) algorithm; useful only if method="Regularized". Other regularization terms can be implemented by setting the value to less than 1 in order to regularized less (to get closer to the results of the EM method) or more than 1 to regularized more (to get closer to the results of the mean imputation)

threshold	the threshold for assessing convergence
row.w	Row weights. Can be one of: • NULL (default): all rows weighted equally. • A numeric vector of length nrow(obj): custom positive weights. • "n_miss": rows with more missing values receive lower weight. Weights are normalized to sum to 1.
seed	integer, by default seed = NULL implies that missing values are initially imputed by the mean of each variable. Other values leads to a random initialization
nb.init	integer corresponding to the number of random initializations; the first initialization is the initialization with the mean imputation
maxiter	integer, maximum number of iteration for the algorithm
miniter	integer, minimum number of iteration for the algorithm
.progress	Show imputation progress (default = FALSE)

Details

This function performs K-NN or PCA imputation on groups of features independently, which significantly reduce imputation time for large datasets.

Specify k and related arguments to use K-NN, ncp and related arguments for PCA imputation. If k and ncp are both NULL, then the group-wise parameters column i.e., group$parameters must be specified and must contains either k or ncp for all groups of group-wise parameters.

Strategies for grouping may include:

- Breaking down search space by chromosomes
- Grouping features with their flanking values/neighbors (e.g., 1000 bp down/up stream of a CpG)
- Using clusters identified by column clustering techniques

Only features in each group (each row of the data.frame) will be imputed, using the search space defined as the union of the features and optional aux columns of that group. Columns that are in aux or in the object but not in any features will be left unchanged.

Value

A numeric matrix of the same dimensions as obj with missing values imputed.

See Also

group_features()

Examples

```
# Generate example data with missing values
set.seed(1234)
to_test <- sim_mat(
  m = 20,
```

```
  n = 50,
  perc_NA = 0.3,
  perc_col_NA = 1,
  nchr = 2
)
# t() to put features in columns
obj <- t(to_test$input)
head(to_test$group_feature) # which group each feature belongs to

# Use group_features() to create the group tibble. By setting `k = 5` in
# group_features(), we are doing K-NN imputation in group_imp(). To make use
# of the `subset` argument in knn_imp(), we specify subset in group_features().
# For demonstration of different group-wise parameters we set `k = 10` for the
# second group.
subset_features <- sample(to_test$group_feature$feature_id, size = 10)
head(subset_features)
knn_df <- group_features(obj, to_test$group_feature, k = 5, subset = subset_features)
knn_df
knn_df$parameters[[2]]$k <- 10
knn_df$parameters

# Run grouped imputation. `k` for K-NN has been specified in `knn_df`.
knn_grouped <- group_imp(obj, group = knn_df, cores = 2)
knn_grouped # only features in subset are imputed

# Specify `ncp` for PCA directly in the group_imp() function (instead of in
# group_features()). We run in parallel with `mirai::daemons(2)`.

mirai::daemons(2) # Set up 2 cores for parallelization
pca_df <- group_features(obj, to_test$group_feature)
pca_grouped <- group_imp(obj, group = pca_df, ncp = 2)
mirai::daemons(0)
pca_grouped
```

---

khanmiss1 *Khan microarray data with random missing values*

Description

A text file containing the Khan microarray data with random missing values introduced for illustrative purposes. Adapted from the impute package.

Usage

```
data(khanmiss1)
```

Format

The data set khanmiss1 consists of genes on rows and samples on column.

Please note that this dataset was derived from the original by introducing some random missing values purely for the purpose of illustration.

Source

Khan, J. and Wei, J.S. and Ringner, M. and Saal, L. and Ladanyi, M. and Westermann, F. and Berthold, F. and Schwab, M. and Antonescu, C. and Peterson, C. and Meltzer, P. (2001) Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural network. Nature Medicine 7, 673-679.

References

Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu (2002). Diagnosis of multiple cancer types by shrunken centroids of gene expression PNAS 99: 6567-6572. Available at www.pnas.org

Examples

```
data(khanmiss1)
```

---

knn_imp *K-Nearest Neighbor Imputation for Numeric Matrices*

Description

Imputes missing values in numeric matrices using full k-nearest neighbor imputation.

Usage

```
knn_imp(
  obj,
  k,
  colmax = 0.9,
  method = c("euclidean", "manhattan"),
  cores = 1,
  post_imp = TRUE,
  subset = NULL,
  dist_pow = 0,
  tree = NULL
)
```

Arguments

obj	A numeric matrix with **samples in rows** and **features in columns**.
k	Number of nearest neighbors for imputation. 10 is a good starting point.
colmax	A number from 0 to 1. Threshold of missing data above which K-NN imputation is skipped.
method	Either "euclidean" (default) or "manhattan". Distance metric for nearest neighbor calculation.

cores	Number of cores to parallelize over.
post_imp	Whether to impute remaining missing values (those that failed K-NN imputation) using column means (default = TRUE).
subset	Character vector of column names or integer vector of column indices specifying which columns to impute.
dist_pow	The amount of penalization for further away nearest neighbors in the weighted average. dist_pow = 0 (default) is the simple average of the nearest neighbors.
tree	Either NULL (default, brute-force K-NN), "ball", or "kd" to find nearest neighbors using the {mlpack} ball-tree or kd-tree algorithms.

Details

This function performs **column-wise** nearest neighbor imputation.

When dist_pow > 0, imputed values are computed as distance-weighted averages where weights are inverse distances raised to the power of dist_pow.

The tree parameter enables faster neighbor search using spatial data structures but requires prefilling missing values with column means, which may introduce bias in high-missingness data. Tree construction overhead may reduce performance for low-dimensional data.

Value

A numeric matrix of the same dimensions as obj with missing values imputed.

Performance Optimization

- **Tree methods**: Only use when imputation runtime becomes prohibitive and missingness is low (<5% missing)
- **Subset imputation**: Use subset parameter for efficiency when only specific columns need imputation (e.g., epigenetic clocks CpGs)

References

Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu (2002). Diagnosis of multiple cancer types by shrunken centroids of gene expression PNAS 99: 6567-6572. Available at www.pnas.org

Examples

```
data(khanmiss1)
sum(is.na(khanmiss1))

# Basic K-NN imputation (khanmiss1 has genes in rows, so transpose)
t_khanmiss1 <- t(khanmiss1)
result <- knn_imp(t_khanmiss1, k = 5)
result
```

mean_imp_col *Column Mean Imputation*

Description

Imputes missing values in a matrix by replacing them with the mean of their respective columns.

Usage

```
mean_imp_col(obj, subset = NULL)
```

Arguments

obj	A numeric matrix with **samples in rows** and **features in columns**.
subset	Character vector of column names or integer vector of column indices specifying which columns to impute.

Details

This function calculates the mean for each column excluding missing values and replaces all missing values in that column with the computed mean.

The subset parameter allows for imputation of only specific columns.

Value

A numeric matrix of the same dimensions as obj with missing values in the specified columns replaced by column means.

Examples

```
# Create example matrix with missing values
mat <- matrix(c(1, 2, NA, 4, 5, 6, NA, 8, 9), nrow = 3)
colnames(mat) <- c("A", "B", "C")
mat

# Impute missing values with column means
imputed_mat <- mean_imp_col(mat)
imputed_mat

# Impute only specific columns by name
imputed_subset <- mean_imp_col(mat, subset = c("A", "C"))
imputed_subset

# Impute only specific columns by index
imputed_idx <- mean_imp_col(mat, subset = c(1, 3))
imputed_idx
```

`pca_imp` *Impute dataset with PCA*

Description

(From the missMDA package on CRAN) Impute the missing values of a dataset with the Principal Components Analysis model. Can be used as a preliminary step before performing a PCA on an completed dataset.

Usage

```
pca_imp(
  obj,
  ncp = 2,
  scale = TRUE,
  method = c("regularized", "EM"),
  coeff.ridge = 1,
  row.w = NULL,
  threshold = 1e-06,
  seed = NULL,
  nb.init = 1,
  maxiter = 1000,
  miniter = 5
)
```

Arguments

`obj`	A numeric matrix with **samples in rows** and **features in columns**.
`ncp`	integer corresponding to the number of components used to to predict the missing entries
`scale`	boolean. By default TRUE leading to a same weight for each variable
`method`	"regularized" by default or "EM"
`coeff.ridge`	1 by default to perform the regularized pca_imp (imputePCA) algorithm; useful only if method="Regularized". Other regularization terms can be implemented by setting the value to less than 1 in order to regularized less (to get closer to the results of the EM method) or more than 1 to regularized more (to get closer to the results of the mean imputation)
`row.w`	Row weights. Can be one of: • `NULL` (default): all rows weighted equally. • A numeric vector of length `nrow(obj)`: custom positive weights. • `"n_miss"`: rows with more missing values receive lower weight. Weights are normalized to sum to 1.
`threshold`	the threshold for assessing convergence

seed	integer, by default seed = NULL implies that missing values are initially imputed by the mean of each variable. Other values leads to a random initialization
nb.init	integer corresponding to the number of random initializations; the first initialization is the initialization with the mean imputation
maxiter	integer, maximum number of iteration for the algorithm
miniter	integer, minimum number of iteration for the algorithm

Details

Impute the missing entries of a mixed data using the iterative PCA algorithm (method="EM") or the regularised iterative PCA algorithm (method="Regularized"). The (regularized) iterative PCA algorithm first consists imputing missing values with initial values such as the mean of the variable. If the argument seed is set to a specific value, a random initialization is performed: the initial values are drawn from a gaussian distribution with mean and standard deviation calculated from the observed values. nb.init different random initialization can be drawn. In such a situation, the solution giving the smallest objective function (the mean square error between the fitted matrix and the observed one) is kept. The second step of the (regularized) iterative PCA algorithm is to perform PCA on the completed dataset. Then, it imputes the missing values with the (regularized) reconstruction formulae of order ncp (the fitted matrix computed with ncp components for the (regularized) scores and loadings). These steps of estimation of the parameters via PCA and imputation of the missing values using the (regularized) fitted matrix are iterate until convergence. The iterative PCA algorithm is also known as the EM-PCA algorithm since it corresponds to an EM algorithm of the fixed effect model where the data are generated as a fixed structure (with a low rank representation) corrupted by noise. The number of components used in the algorithm can be found using cross-validation criteria implemented in the function estim_ncpPCA.
We advice to use the regularized version of the algorithm to avoid the overfitting problems which are very frequent when there are many missing values. In the regularized algorithm, the singular values of the PCA are shrinked.
The output of the algorithm can be used as an input of the PCA function of the FactoMineR package in order to perform PCA on an incomplete dataset.

Value

A numeric matrix of the same dimensions as obj with missing values imputed.

Author(s)

Francois Husson <francois.husson@institut-agro.fr>

Julie Josse <julie.josse@polytechnique.edu>

References

Josse, J & Husson, F. (2013). Handling missing values in exploratory multivariate data analysis methods. Journal de la SFdS. 153 (2), pp. 79-99.

Josse, J. and Husson, F. missMDA (2016). A Package for Handling Missing Values in Multivariate Data Analysis. Journal of Statistical Software, 70 (1), pp 1-31 doi:10.18637/jss.v070.i01.

Examples

```
data("khanmiss1")

# Transpose to put genes on columns. Randomly initialize missing values 5
# times (1st time is mean).
pca_imp(t(khanmiss1), ncp = 2, nb.init = 5)
```

`print.ImputedMatrix` *Print ImputedMatrix*

Description

Print ImputedMatrix

Usage

```
## S3 method for class 'ImputedMatrix'
print(x, n = 5, m = 5, ...)
```

Arguments

`x`	An ImputedMatrix
`n`	Number of rows to print
`m`	Number of cols to print
`...`	Not used

Value

Invisible object of class ImputedMatrix

Examples

```
data(khanmiss1)
t_khanmiss1 <- t(khanmiss1)
result <- knn_imp(t_khanmiss1, k = 5)
print(result, n = 6, m = 6)
```

sim_mat *Simulate Methylation Beta Values with Metadata*

Description

This function generates a matrix of random normal data, scaled between 0 and 1 per column. It also creates corresponding data frames for feature and sample metadata and can optionally introduce NA values into a specified proportion of rows.

Usage

```
sim_mat(
  n = 100,
  m = 100,
  nchr = 2,
  ngrp = 1,
  perc_NA = 0.5,
  perc_col_NA = 0.5,
  beta = TRUE
)
```

Arguments

n	An integer specifying the number of rows (features). Default is 100.
m	An integer specifying the number of columns (samples). Default is 100.
nchr	An integer for the number of chromosome groups to assign to features (e.g., nchr = 22 for human autosomes). Default is 2.
ngrp	An integer for the number of groups to assign to samples. Default is 1.
perc_NA	A numeric value between 0 and 1 indicating the proportion of values to set to NA within each selected row. Default is 0.5.
perc_col_NA	A numeric value between 0 and 1 indicating the proportion of rows to select for NA introduction. Default is 0.5.
beta	If TRUE (default) then simulate beta values by scaling the values between 0 and 1.

Value

A list containing three elements:

- input: The simulated n x m numeric matrix with values between 0 and 1.
- group_feature: A data.frame with feature IDs and their assigned chromosome group.
- group_sample: A data.frame with sample IDs and their assigned group.

Examples

```
set.seed(123)
sim_data <- sim_mat(n = 50, m = 10)

# Metadata of each features
sim_data$group_feature[1:5, ]
sim_data$group_sample[1:5, ]

# View the first few rows and columns of the matrix
sim_data$input[1:5, 1:5]

# Generate a dataset with no missing values
sim_data_complete <- sim_mat(n = 50, m = 10, perc_NA = 0, perc_col_NA = 0)
sum(is.na(sim_data_complete$input))
```

---

slide_imp *Sliding Window K-NN or PCA Imputation*

Description

Performs sliding window K-NN or PCA imputation of large numeric matrices column-wise.

This method assumes that columns are meaningfully sorted.

Usage

```
slide_imp(
  obj,
  n_feat,
  n_overlap,
  k = NULL,
  colmax = 0.9,
  knn_method = c("euclidean", "manhattan"),
  cores = 1,
  post_imp = TRUE,
  dist_pow = 0,
  subset = NULL,
  ncp = NULL,
  scale = TRUE,
  pca_method = c("regularized", "EM"),
  coeff.ridge = 1,
  seed = NULL,
  row.w = NULL,
  nb.init = 1,
  maxiter = 1000,
  miniter = 5,
  .progress = TRUE
)
```

Arguments

obj	A numeric matrix with **samples in rows** and **features in columns**.
n_feat	Number of features in a window.
n_overlap	Number of overlapping features between two windows.
k	Number of nearest neighbors for imputation. 10 is a good starting point.
colmax	A number from 0 to 1. Threshold of missing data above which K-NN imputation is skipped.
knn_method	Either "euclidean" (default) or "manhattan". Distance metric for nearest neighbor calculation.
cores	Number of cores to parallelize over.
post_imp	Whether to impute remaining missing values (those that failed K-NN imputation) using column means (default = TRUE).
dist_pow	The amount of penalization for further away nearest neighbors in the weighted average. dist_pow = 0 (default) is the simple average of the nearest neighbors.
subset	Character vector of column names or integer vector of column indices specifying which columns to impute.
ncp	integer corresponding to the number of components used to to predict the missing entries
scale	boolean. By default TRUE leading to a same weight for each variable
pca_method	"regularized" by default or "EM".
coeff.ridge	1 by default to perform the regularized pca_imp (imputePCA) algorithm; useful only if method="Regularized". Other regularization terms can be implemented by setting the value to less than 1 in order to regularized less (to get closer to the results of the EM method) or more than 1 to regularized more (to get closer to the results of the mean imputation)
seed	integer, by default seed = NULL implies that missing values are initially imputed by the mean of each variable. Other values leads to a random initialization
row.w	Row weights. Can be one of: • NULL (default): all rows weighted equally. • A numeric vector of length nrow(obj): custom positive weights. • "n_miss": rows with more missing values receive lower weight. Weights are normalized to sum to 1.
nb.init	integer corresponding to the number of random initializations; the first initialization is the initialization with the mean imputation
maxiter	integer, maximum number of iteration for the algorithm
miniter	integer, minimum number of iteration for the algorithm
.progress	Show progress bar (default = TRUE).

Details

The sliding window approach divides the input matrix into smaller, overlapping segments and applies imputation to each window independently. Values in overlapping areas are averaged across windows to produce the final imputed result. This approach assumes that features (columns) are sorted meaningfully (e.g., by genomic position, time, etc.).

Specify `k` and related arguments to use K-NN, `ncp` and related arguments for PCA.

Value

A numeric matrix of the same dimensions as `obj` with missing values imputed.

Examples

```
# Generate sample data with missing values with 20 samples and 100 columns
# where the column order is sorted (i.e., by genomic position)
set.seed(1234)
beta_matrix <- t(sim_mat(100, 20)$input)

# Sliding Window K-NN imputation by specifying `k`
imputed_knn <- slide_imp(
  beta_matrix,
  k = 5,
  n_feat = 50,
  n_overlap = 10,
  scale = FALSE # This argument belongs to PCA imputation and will be ignored
)
imputed_knn

# Sliding Window PCA imputation by specifying `ncp`
pca_knn <- slide_imp(
  beta_matrix,
  ncp = 2,
  n_feat = 50,
  n_overlap = 10
)
pca_knn
```

---

tune_imp *Tune Parameters for Imputation Methods*

Description

Tunes hyperparameters for imputation methods such as `slide_imp()`, `knn_imp()`, `pca_imp()`, or user-supplied custom functions by repeated cross-validation.

Usage

```
tune_imp(
  obj,
  parameters,
  .f = NULL,
  rep = 1,
  num_na = 100,
  rowmax = 0.9,
  colmax = 0.9,
  check_sd = FALSE,
  max_iter = 1000,
  .progress = TRUE,
  cores = 1
)
```

Arguments

obj	A numeric matrix with **samples in rows** and **features in columns**.
parameters	A data.frame specifying parameter combinations to tune, where each column represents a parameter accepted by .f (excluding obj). List columns are supported for complex parameters. Duplicate rows are automatically removed. When .f = NULL, the imputation method is inferred from the column names: • k: K-NN imputation • ncp: PCA imputation • k or ncp with n_feat and n_overlap: sliding window imputation
.f	Custom function to tune. Must accept obj as the first argument, accept the arguments in parameters, and return a matrix with the same dimension as obj (default = NULL).
rep	Either an integer specifying the number of repetitions for random NA injection, or a list defining fixed NA positions for each repetition (in which case num_na is ignored). The list elements can be one of the following formats: • A two-column integer matrix. The first column is the row index, the second column is the column index. Each row is an missing value. • A numeric vector specifying linear locations of NAs.
num_na	The number of missing values used to estimate prediction quality.
rowmax	Number between 0 to 1. NA injection cannot create rows with more missing % than this number.
colmax	Number between 0 to 1. NA injection cannot create cols with more missing % than this number.
check_sd	Check if after NA injections zero variance columns are created or not.
max_iter	Maximum number of iterations to attempt finding valid NA positions (default to 1000).
.progress	Show progress bar (default = TRUE).
cores	Controls the number of cores to parallelize over for K-NN and sliding window K-NN imputation only. To setup parallelization for PCA and sliding window PCA imputation, use mirai::daemons().

Details

The function supports tuning for built-in imputation methods ("slide_imp", "knn_imp", "pca_imp") or custom functions provided via `.f`.

When using a custom `.f`, the columns in `parameters` must correspond to the arguments of `.f` (excluding the `obj` argument). The custom function must accept `obj` (a numeric matrix) as its first argument and return a numeric matrix of identical dimensions.

Tuning results can be evaluated using the {yardstick} package or `compute_metrics()`.

Value

A `tibble::tibble()` with columns from `parameters`, plus `param_set` (unique parameter set ID), `rep` (repetition index), and `result` (a nested tibble containing `truth` and `estimate` columns for true and imputed values, respectively).

Examples

```
data(khanmiss1)
obj <- t(khanmiss1)[1:20, sample.int(nrow(khanmiss1), size = 200)]

# Tune full K-NN imputation
parameters <- data.frame(k = c(5, 10))

# With random NA injection
results <- tune_imp(obj, parameters, rep = 1, num_na = 20)

# Compute metrics on results
compute_metrics(results)

# Tune with fixed NA positions (2 repetitions)
# Positions must not be NA in the original `obj`
na_positions <- list(
  matrix(c(1, 2, 3, 1, 1, 1), ncol = 2), # Rows 1-3 in column 1
  matrix(c(2, 3, 4, 2, 2, 2), ncol = 2) # Rows 2-4 in column 2
)
results_fixed <- tune_imp(
  obj,
  data.frame(k = 10),
  rep = na_positions
)

compute_metrics(results_fixed)

# Custom imputation function example, with 2 cores parallelization with `mirai::daemons()`
custom_imp <- function(obj, mean = 0, sd = 1) {
  na_pos <- is.na(obj)
  obj[na_pos] <- rnorm(sum(na_pos), mean = mean, sd = sd)
  obj
}

mirai::daemons(2) # Setup 2 cores for parallelization
parameters_custom <- data.frame(mean = c(0, 0, 1), sd = c(1, 2, 1))
```

```
results_custom <- tune_imp(
  obj,
  parameters_custom,
  .f = custom_imp,
  rep = 2,
  num_na = 20
)
mirai::daemons(0)
compute_metrics(results_custom)
```

Index