Package: RMCLab (via r-universe)

May 25, 2026

Type Package

Title Lab for Matrix Completion and Imputation of Discrete Rating Data

Version 0.1.0

Date 2025-07-25

Description Collection of methods for rating matrix completion, which is a statistical framework for recommender systems. Another relevant application is the imputation of rating-scale survey data in the social and behavioral sciences. Note that matrix completion and imputation are synonymous terms used in different streams of the literature. The main functionality implements robust matrix completion for discrete rating-scale data with a low-rank constraint on a latent continuous matrix (Archimbaud, Alfons, and Wilms (2025) <doi:10.48550/arXiv.2412.20802>). In addition, the package provides wrapper functions for 'softImpute' (Mazumder, Hastie, and Tibshirani, 2010, <https://www.jmlr.org/papers/v11/mazumder10a.html>; Hastie, Mazumder, Lee, Zadeh, 2015, <https://www.jmlr.org/papers/v16/hastie15a.html>) for easy tuning of the regularization parameter, as well as benchmark methods such as median imputation and mode imputation.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.5.0)

Imports Rcpp, softImpute

LinkingTo Rcpp, RcppArmadillo

URL https://github.com/aalfons/RMCLab

BugReports https://github.com/aalfons/RMCLab/issues

Author Andreas Alfons [aut, cre] (<https://orcid.org/0000-0002-2513-3788>), Aurore Archimbaud [aut] (<https://orcid.org/0000-0002-6511-9091>)

Maintainer Andreas Alfons <alfons@ese.eur.nl>

RoxygenNote 7.3.2

LazyData true

Repository https://aalfons.r-universe.dev

Date/Publication 2025-07-25 18:49:19 UTC

RemoteUrl https://github.com/aalfons/rmclab

RemoteRef HEAD

RemoteSha 209194f79812cedf4eb8c9844666560a24d0c169

Contents

`RMCLab-package` *Lab for Matrix Completion and Imputation of Discrete Rating Data*

Description

Collection of methods for rating matrix completion, which is a statistical framework for recommender systems. Another relevant application is the imputation of rating-scale survey data in the social and behavioral sciences. Note that matrix completion and imputation are synonymous terms used in different streams of the literature. The main functionality implements robust matrix completion for discrete rating-scale data with a low-rank constraint on a latent continuous matrix (Archimbaud, Alfons, and Wilms (2025) <doi:10.48550/arXiv.2412.20802>). In addition, the package provides wrapper functions for 'softImpute' (Mazumder, Hastie, and Tibshirani, 2010, <https://www.jmlr.org/papers/v11/mazumder10a.html>; Hastie, Mazumder, Lee, Zadeh, 2015, <https://www.jmlr.org/papers/
for easy tuning of the regularization parameter, as well as benchmark methods such as median imputation and mode imputation.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Andreas Alfons [aut, cre] (<https://orcid.org/0000-0002-2513-3788>), Aurore Archimbaud [aut] (<https://orcid.org/0000-0002-6511-9091>)

Maintainer: Andreas Alfons <alfons@ese.eur.nl>

References

Archimbaud, A., Alfons, A., and Wilms, I. (2025) Robust Matrix Completion for Discrete Rating-Scale Data. arXiv:2412.20802. doi:10.48550/arXiv.2412.20802.

Hastie, T., Mazumder, R., Lee, J. D. and Zadeh, R. (2015) Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares. *Journal of Machine Learning Research*, **16**(104), 3367–3402.

Mazumder, R., Hastie, T. and Tibshirani, R. (2010) Spectral Regularization Algorithms for Learning Large Incomplete Matrices. *Journal of Machine Learning Research*, **11**(80), 2287–2322.

See Also

Useful links:

- https://github.com/aalfons/RMCLab
- Report bugs at https://github.com/aalfons/RMCLab/issues

---

create_splits *Create splits of observed data cells for hyperparameter tuning*

Description

Split the observed cells of a data matrix into training and validation sets for hyperparameter tuning. Methods are available for repeated holdout validation and K-fold cross-validation.

Usage

```
create_splits(indices, control)

holdout(indices, pct = 0.1, R = 10L)

cv_folds(indices, K = 5L)
```

Arguments

indices	an integer vector giving the indices of observed cells in a data matrix.
control	a control object inheriting from class "split_control" as generated by holdout_control() for repeated holdout validation or cv_folds_control() for K-fold cross-validation.
pct	numeric in the interval (0, 1); the percentage of observed cells in the data matrix to be randomly selected into the validation set (defaults to 0.1).
R	an integer giving the number of random splits into training and validation sets (defaults to 10).
K	an integer giving the number of cross-validation folds (defaults to 5).

Details

Functions holdout() and cv_folds() are wrapper functions that first call holdout_control() and cv_folds_control(), respectively, before calling create_splits().

Value

A list of index vectors giving the validation sets of the respective replication or cross-validation fold.

Author(s)

Andreas Alfons

See Also

holdout_control(), cv_folds_control(),

rdmc_tune(), soft_impute_tune()

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# set up validation sets so that methods use same data splits
set.seed(20250723)
observed <- which(!is.na(MovieLensToy))
holdout_splits <- holdout(observed, R = 5)
# robust discrete matrix completion with hyperparameter tuning
fit_RDMC <- rdmc_tune(
  MovieLensToy,
  lambda = fraction_grid(nb_lambda = 6),
  splits = holdout_splits
)
# Soft-Impute with discretization step and hyperparameter tuning
fit_SI <- soft_impute_tune(
  MovieLensToy,
  lambda = fraction_grid(nb_lambda = 6, reverse = TRUE),
  splits = holdout_splits
)
```

```
# extract optimal values of regularization parameter
get_lambda(fit_RDMC)
get_lambda(fit_SI)
```

`get_completed` *Extract the completed (imputed) data matrix*

Description

Extract the completed (i.e., imputed) data matrix from an object returned by a matrix completion algorithm.

Usage

```
get_completed(object, ...)

## S3 method for class 'rdmc_tuned'
get_completed(object, ...)

## S3 method for class 'rdmc'
get_completed(object, which, ...)

## S3 method for class 'soft_impute_tuned'
get_completed(object, discretized = NULL, ...)

## S3 method for class 'soft_impute'
get_completed(object, which, discretized = NULL, ...)

## S3 method for class 'median_impute'
get_completed(object, discretized = NULL, ...)

## S3 method for class 'mode_impute'
get_completed(object, ...)

get_imputed(object, ...)
```

Arguments

`object`	an object returned by a matrix completion algorithm.
`...`	additional arguments to be passed down to methods.
`which`	an integer specifying the index of the regularization parameter for which to extract the completed data matrix.
`discretized`	a logical indicating if the completed data matrix with or without the discretization step should be extracted. The default is `TRUE` if the discretization step was performed and `FALSE` otherwise.

Value

The completed (i.e., imputed) data matrix.

Note

Matrix completion and imputation are synonymous terms used in different streams of the literature, hence `get_imputed()` is an alias for `get_completed()` with the same functionality.

Author(s)

Andreas Alfons and Aurore Archimbaud

See Also

`rdmc_tune()`, `soft_impute_tune()`, `median_impute()`, `mode_impute()`

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# robust discrete matrix completion with hyperparameter tuning
set.seed(20250723)
fit <- rdmc_tune(MovieLensToy,
                 lambda = fraction_grid(nb_lambda = 6),
                 splits = holdout_control(R = 5))
# extract completed matrix with optimal regularization parameter
X_hat <- get_completed(fit)
head(X_hat)

# for more examples, see the help files of other functions for
# matrix completion and imputation methods
```

---

get_lambda *Extract the optimal value of the regularization parameter*

Description

Extract the optimal value of the regularization parameter from an object returned by a matrix completion algorithm.

Usage

```
get_lambda(object, ...)

## S3 method for class 'rdmc_tuned'
get_lambda(object, ...)
```

```
## S3 method for class 'rdmc'
get_lambda(object, relative = TRUE, ...)

## S3 method for class 'soft_impute_tuned'
get_lambda(object, ...)

## S3 method for class 'soft_impute'
get_lambda(object, relative = TRUE, ...)
```

Arguments

object	an object returned by a matrix completion algorithm with a regularization parameter.
...	additional arguments to be passed down to methods.
relative	logical; in case the values of the regularization parameter were given relative to a certain reference value computed from the data at hand, this allows to return the optimal value before or after multiplication with that reference value.

Value

The optimal value of the regularization parameter.

Author(s)

Andreas Alfons

See Also

rdmc_tune(), soft_impute_tune()

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# robust discrete matrix completion with hyperparameter tuning
set.seed(20250723)
fit <- rdmc_tune(MovieLensToy,
                 lambda = fraction_grid(nb_lambda = 6),
                 splits = holdout_control(R = 5))
# extract optimal value of regularization parameter
get_lambda(fit)

# for more examples, see the help files of other functions for
# matrix completion and imputation methods
```

get_nb_iter *Extract the number of iterations*

Description

Extract the number of iterations from an object returned by a matrix completion algorithm.

Usage

```
get_nb_iter(object, ...)

## S3 method for class 'rdmc_tuned'
get_nb_iter(object, ...)

## S3 method for class 'rdmc'
get_nb_iter(object, ...)
```

Arguments

object	an object returned by an iterative matrix completion algorithm.
...	currently ignored.

Value

The number of iterations performed in the iterative algorithm.

Author(s)

Andreas Alfons

See Also

rdmc_tune()

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# robust discrete matrix completion with hyperparameter tuning
set.seed(20250723)
fit <- rdmc_tune(MovieLensToy,
                 lambda = fraction_grid(nb_lambda = 6),
                 splits = holdout_control(R = 5))
# extract number of iterations with optimal regularization parameter
get_nb_iter(fit)
```

lambda_grid *Construct grid of values for the regularization parameter*

Description

Construct a grid of values for the regularization parameter in `rdmc()` or `soft_impute()`.

Usage

```
fraction_grid(
  min = 0.01,
  max = 1,
  nb_lambda = 10L,
  log = TRUE,
  reverse = FALSE
)

mult_grid(min = 0.05, factor = 1.5, nb_lambda = 10L)
```

Arguments

`min`	numeric; the smallest value of the regularization parameter. For `fraction_grid()`, it must be in the interval (0, 1) with the default being 0.01. For `mult_grid()`, it must be larger than 0 with the default being 0.05.
`max`	numeric; the largest value of the regularization parameter. It must be in the interval (`min`, 1] with the default being 1.
`nb_lambda`	a positive integer giving the number of values for the regularization parameter to be generated.
`log`	a logical indicating whether the grid of values should be on a logarithmic scale (defaults to `TRUE`).
`reverse`	a logical indicating whether the grid of values should be in ascending order (`FALSE`, the default) or in descending order (`TRUE`).
`factor`	numeric; multiplication factor larger than 1 to be used to construct the values of the regularization parameter. That is, the second value is obtained by multiplying `min` by `factor`, with this process being iterated further.

Details

Function `fraction_grid()` generates a grid of values in the interval (0, 1], either on a logarithmic or linear scale, which `rdmc()` and `soft_impute()` can relate to a certain reference value computed from the data at hand.

Function `mult_grid()` generates a multiplicative grid in which the each value is obtained by multiplying the previous value with a specified factor.

Value

A numeric vector of values for the regularization parameter.

See Also

rdmc(), rdmc_tune(), soft_impute(), soft_impute_tune()

Examples

```
fraction_grid()
fraction_grid(log = FALSE)
mult_grid(factor = 2)
```

`median_impute` *Median imputation*

Description

Perform median imputation. In case of discrete rating-scale data, a discretization step can be carried out afterwards to make sure that the imputed values are mapped to the rating scale of the observed values (as the median of a given column may lie in between two answer categories in case of an even number of observed values). This is done by randomly sampling from the largest answer category smaller than the median and the smallest answer category larger than the median (for each missing cell).

Usage

```
median_impute(X, discretize = TRUE, values = NULL)
```

Arguments

`X`	a matrix or data frame with missing values.
`discretize`	a logical indicating whether to include a discretization step after median imputation (defaults to `TRUE`). In case of discrete rating-scale data, this can be used to ensure that the imputed values are mapped to the discrete rating scale of the observed values.
`values`	an optional numeric vector giving the possible values of discrete ratings. This is ignored if `discretize` is `FALSE`. Currently, the possible values are assumed to be the same for all columns. If `NULL`, the unique values of the observed parts of `X` are used.

Value

An object of class "median_impute" with the following components:

medians	a numeric vector containing the median of the observed values for each variable.
X	a numeric matrix containing the completed (i.e., imputed) data matrix.
X_discretized	a numeric matrix containing the completed (i.e., imputed) data matrix after the discretization step. This is only returned if requested via discretize = TRUE.

The class structure is still experimental and may change in the future. Use the accessor function get_completed() to extract the completed (i.e., imputed) data matrix.

Author(s)

Andreas Alfons

See Also

mode_impute()

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# median imputation with discretization step
fit <- median_impute(MovieLensToy, values = 1:5)
# extract discretized completed matrix
X_hat <- get_completed(fit)
head(X_hat)
```

---

mode_impute *Mode imputation*

Description

Perform mode imputation for discrete data. In case of multiple modes in a given column, one of them is selected at random for each missing cell.

Usage

```
mode_impute(X, values = NULL)
```

Arguments

X	a matrix or data frame of discrete data with missing values.
values	an optional numeric vector giving the possible values. Currently, the possible values are assumed to be the same for all columns. If NULL, the unique values of the observed parts of X are used.

Value

An object of class `"mode_impute"` with the following components:

`modes`	a list containing the mode(s) of the observed values for each variable.
`X`	a numeric matrix containing the completed (i.e., imputed) data matrix.

The class structure is still experimental and may change in the future. Use the accessor function `get_completed()` to extract the completed (i.e., imputed) data matrix.

Note

The mode is computed as the most frequent value, hence this function is only suitable for discrete data. It does not estimate the mode of a continuous density.

Author(s)

Andreas Alfons

See Also

`median_impute()`

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# mode imputation
fit <- mode_impute(MovieLensToy, values = 1:5)
# extract completed matrix
X_hat <- get_completed(fit)
head(X_hat)
```

MovieLensToy *Toy example derived from the MovieLens 100K dataset*

Description

This dataset is a toy example derived from the MovieLens 100K dataset. The original data were collected via the MovieLens website (movielens.umn.edu) over a seven-month period, from September 19, 1997, to April 22, 1998.

Usage

```
data("MovieLensToy")
```

Format

A matrix with 149 rows and 33 variables. Rows represent users and columns respresent movies. Each cell contains a rating from 1 to 5, or NA if the movie was not rated by the user.

The row names correspond to the user ID and the column names to the movie ID in the original MovieLens 100K dataset.

Details

The original MovieLens 100K dataset contains 100,000 ratings on a scale from 1 to 5 provided by 943 users for 1,682 movies.

This toy version has been curated to be used in instantaneously computable examples of the package documentation. It includes only users who provided at least 200 ratings and movies that were rated at least 300 times, ensuring a denser and more informative subset for testing and demonstration purposes.

Source

GroupLens Research, https://grouplens.org/datasets/movielens/100k/

References

F. Maxwell Harper and Joseph A. Konstan (2015) The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, **5**(4), Article 19. doi:10.1145/2827872.

Examples

```
data("MovieLensToy")
dim(MovieLensToy)
```

---

rdmc *Robust discrete matrix completion*

Description

Perform robust discrete matrix completion with a low-rank constraint on a latent continuous matrix, implemented via an ADMM algorithm.

Usage

```
rdmc(
  X,
  values = NULL,
  lambda = fraction_grid(),
  relative = TRUE,
  loss = c("pseudo_huber", "absolute", "truncated"),
  loss_const = NULL,
  type = "svd",
```

```
  svd_tol = 1e-04,
  rank_max = NULL,
  mu = 0.1,
  delta = 1.05,
  conv_tol = 1e-04,
  max_iter = 100L,
  L = NULL,
  Theta = NULL
)
```

Arguments

X	a matrix or data frame of discrete ratings with missing values.
values	an optional numeric vector giving the possible values of the ratings. Currently, these are assumed to be the same for all columns. If NULL, the unique values of the observed parts of X are used.
lambda	a numeric vector giving values of the regularization parameter. See fraction_grid() for the default values.
relative	a logical indicating whether the values of the regularization parameter should be considered relative to a certain reference value computed from the data at hand. If TRUE (the default), the values of lambda are multiplied with the largest singular value of the median-centered data matrix with missing values replaced by zeros.
loss	a character string specifying the robust loss function for the loss part of the objective function. Possible values are "pseudo_huber" (the default) for the pseudo-Huber loss, "absolute" for the absolute loss, and "truncated" for the truncated absolute loss. See 'Details' for more information.
loss_const	tuning constant for the loss function. For the pseudo-Huber loss, the default value is the average step size between the rating categories in values. For the truncated absolute loss, the default is half the range of the rating categories in values. This is ignored for the absolute loss, which does not have a tuning parameter. See 'Details' for more information.
type	a character string specifying the type of algorithm for the low-rank latent continuous matrix. Currently only "svd" is implemented for a soft-thresholded SVD step.
svd_tol	numeric tolerance for the soft-thresholded SVD step. Only singular values larger than svd_tol are kept to construct the low-rank latent continuous matrix.
rank_max	a positive integer giving a rank constraint in the soft-thresholded SVD step for the latent continuous matrix. The default is to use the minimum of the number of rows and columns.
mu	numeric; penalty parameter for the discrepancy between the discrete rating matrix and the latent low-rank continuous matrix. It is not recommended to change the default value of 0.1.
delta	numeric; update factor for penalty parameter mu applied after each iteration to increase the strength of the penalty. It is not recommended to change the default value of 1.05.

`conv_tol` numeric; convergence tolerance for the relative change in the objective function.

`max_iter` a positive integer specifying the maximum number of iterations. In practice, large gains can often be had in the first few iterations, with subsequent iterations yielding relatively small gains until convergence. Hence the default is to perform at most 10 iterations.

`L`, `Theta` starting values for the algorithm. These are not expected to be set by the user. Instead, it is recommended to call this function with a grid of values for the regularization parameter `lambda` so that the implementation automatically takes advantage of warm starts.

Details

For the loss part of the objective function, the pseudo-Huber loss (`loss = "pseudo_huber"`) is given by

$$\rho(x) = \texttt{loss\_const}^2(\sqrt{1 + (x/\texttt{loss\_const})^2} - 1).$$

The absolute loss (`loss = "absolute"`) is given by

$$\rho(x) = |x|,$$

and the truncated absolute loss (`loss = "truncated"`) is defined as

$$\rho(x) = \min(|x|, \texttt{loss\_const}).$$

Value

An object of class `"rdmc"` with the following components:

`lambda` a numeric vector containing the values of the regularization parameter.

`d_max` a numeric value with which the values of the regularization parameter were multiplied. If `relative = TRUE`, the largest singular value of the median-centered data matrix, otherwise 1.

`L` in case of a single value of `lambda`, a numeric matrix containing the predictions of the median-centered data matrix. Otherwise a list of such matrices.

`Z` in case of a single value of `lambda`, an ancillary continuous matrix used in the optimization algorithm. Otherwise a list of such matrices.

`Theta` in case of a single value of `lambda`, a numeric matrix containing the discrepancy parameter, i.e., the multiplier adjusting for the discrepancy between `L` and `Z` in the optimization algorithm. Otherwise a list of such matrices.

`objective` a numeric vector containing the value of the objective function for each value of the regularization parameter.

`converged` a logical vector indicating whether the algorithm converged for each value of the regularization parameter.

`nb_iter` an integer vector containing the number of iterations for each value of the regularization parameter.

`X` in case of a single value of `lambda`, a numeric matrix containing the completed (i.e., imputed) data matrix. Otherwise a list of such matrices.

The class structure is still experimental and may change in the future. The following accessor functions are available:

- `get_completed()` to extract the completed (i.e., imputed) data matrix for a specified value of the regularization parameter,
- `get_lambda()` to extract the values of the regularization parameter,
- `get_nb_iter()` to extract the number of iterations for each value of the regularization parameter.

Author(s)

Andreas Alfons and Aurore Archimbaud

References

Archimbaud, A., Alfons, A., and Wilms, I. (2025) Robust Matrix Completion for Discrete Rating-Scale Data. arXiv:2412.20802. doi:10.48550/arXiv.2412.20802.

See Also

`rdmc_tune()`, `fraction_grid()`

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# robust discrete matrix completion
fit <- rdmc(MovieLensToy)
# extract completed matrix with fifth value of
# regularization parameter
X_hat <- get_completed(fit, which = 5)
head(X_hat)
```

`rdmc_tune` *Robust discrete matrix completion with hyperparameter tuning*

Description

Perform robust discrete matrix completion with a low-rank constraint on a latent continuous matrix, implemented via an ADMM algorithm. The regularization parameter is thereby selected via repeated holdout validation or cross-validation.

Usage

```
rdmc_tune(
  X,
  values = NULL,
  lambda = fraction_grid(),
  relative = TRUE,
  splits = holdout_control(),
  loss = c("pseudo_huber", "absolute", "truncated"),
  loss_const = NULL,
  ...
)
```

Arguments

X	a matrix or data frame of discrete ratings with missing values.
values	an optional numeric vector giving the possible values of the ratings. Currently, these are assumed to be the same for all columns. If NULL, the unique values of the observed parts of X are used.
lambda	a numeric vector giving values of the regularization parameter. See fraction_grid() for the default values.
relative	a logical indicating whether the values of the regularization parameter should be considered relative to a certain reference value computed from the data at hand. If TRUE (the default), the values of lambda are multiplied with the largest singular value of the median-centered data matrix with missing values replaced by zeros.
splits	an object inheriting from class "split_control", as generated by holdout_control() for repeated holdout validation or cv_folds_control() for K-fold cross-validation, or a list of index vectors giving different validation sets of observed cells as generated by create_splits(). Cells in the validation set will be set to NA for fitting the algorithm with the training set of observed cells.
loss	a character string specifying the robust loss function for the loss part of the objective function. Possible values are "pseudo_huber" (the default) for the pseudo-Huber loss, "absolute" for the absolute loss, and "truncated" for the truncated absolute loss. See 'Details' for more information.
loss_const	tuning constant for the loss function. For the pseudo-Huber loss, the default value is the average step size between the rating categories in values. For the truncated absolute loss, the default is half the range of the rating categories in values. This is ignored for the absolute loss, which does not have a tuning parameter. See 'Details' for more information.
...	additional arguments to be passed down to rdmc().

Details

For the loss part of the objective function, the pseudo-Huber loss (loss = "pseudo_huber") is given by

$$\rho(x) = \texttt{loss\_const}^2(\sqrt{1 + (x/\texttt{loss\_const})^2} - 1).$$

The absolute loss (`loss = "absolute"`) is given by

$$\rho(x) = |x|,$$

and the truncated absolute loss (`loss = "truncated"`) is defined as

$$\rho(x) = \min(|x|, \texttt{loss\_const}).$$

Value

An object of class `"rdmc_tuned"` with the following components:

`lambda`	a numeric vector containing the values of the regularization parameter.
`tuning_loss`	a numeric vector containing the (average) values of the loss function on the validation set(s) for each value of the regularization parameter.
`lambda_opt`	numeric; the optimal value of the regularization parameter.
`fit`	an object of class `"rdmc"` containing the results from the algorithm with the optimal regularization parameter on the full (observed) data matrix.

The class structure is still experimental and may change in the future. The following accessor functions are available:

- `get_completed()` to extract the completed (i.e., imputed) data matrix with the optimal value of the regularization parameter,
- `get_lambda()` to extract the optimal value of the regularization parameter,
- `get_nb_iter()` to extract the number of iterations with the optimal value of the regularization parameter.

Author(s)

Andreas Alfons

References

Archimbaud, A., Alfons, A., and Wilms, I. (2025) Robust Matrix Completion for Discrete Rating-Scale Data. arXiv:2412.20802. doi:10.48550/arXiv.2412.20802.

See Also

`rdmc()`, `fraction_grid()`,
`holdout_control()`, `cv_folds_control()`, `create_splits()`

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# robust discrete matrix completion with hyperparameter tuning
set.seed(20250723)
fit <- rdmc_tune(MovieLensToy,
                 lambda = fraction_grid(nb_lambda = 6),
```

```
                splits = holdout_control(R = 5))
# extract completed matrix with optimal regularization parameter
X_hat <- get_completed(fit)
head(X_hat)
# extract optimal value of regularization parameter
get_lambda(fit)
# extract number of iterations with optimal regularization parameter
get_nb_iter(fit)
```

`soft_impute` *Matrix completion via nuclear-norm regularization*

Description

Convenience wrapper for `softImpute()` that allows to supply a grid of values for the regularization parameter. Other noteworthy differences with the original function are that the columns of the data matrix are centered internally, that some of the default values are different, and that the output is structured differently. Moreover, in case of discrete rating-scale data, the wrapper function allows to include a discretization step after fitting the algorithm to map the imputed values to the rating scale of the observed values.

Usage

```
soft_impute(
  X,
  lambda = fraction_grid(reverse = TRUE),
  relative = TRUE,
  type = c("svd", "als"),
  rank.max = NULL,
  thresh = 1e-05,
  maxit = 100L,
  trace.it = FALSE,
  final.svd = TRUE,
  discretize = TRUE,
  values = NULL
)
```

Arguments

`X`	a matrix or data frame with missing values.
`lambda`	a numeric vector giving values of the regularization parameter. See `fraction_grid()` for the default values.
`relative`	a logical indicating whether the values of the regularization parameter should be considered relative to a certain reference value computed from the data at hand. If `TRUE` (the default), the values of `lambda` are multiplied with the value returned by `lambda0()` (applied to the mean-centered data matrix).

type	a character string specifying the type of algorithm. Possible values are "svd" and "als". See softImpute() for details on the algorithms, but note that the default value here is "svd".
rank.max	a positive integer giving a rank constraint. See softImpute() for more details, but note that the default here is to use the minimum of the number of rows and columns minus 1 if type is "svd", and to use 2 if type is "als".
thresh, maxit, trace.it, final.svd	see softImpute().
discretize	a logical indicating whether to include a discretization step after fitting the algorithm (defaults to TRUE). In case of discrete rating-scale data, this can be used to map the imputed values to the discrete rating scale of the observed values.
values	an optional numeric vector giving the possible values of discrete ratings. This is ignored if discretize is FALSE. Currently, the possible values are assumed to be the same for all columns. If NULL, the unique values of the observed parts of X are used.

Value

An object of class "soft_impute" with the following components:

lambda	a numeric vector containing the values of the regularization parameter.
lambda0	a numeric value with which the values of the regularization parameter were multiplied. If relative = TRUE, the value returned by lambda0() (applied to the mean-centered data matrix), otherwise 1.
svd	in case of a single value of lambda, an object returned by softImpute(). Otherwise a list of such objects.
X	in case of a single value of lambda, a numeric matrix containing the completed (i.e., imputed) data matrix. Otherwise a list of such matrices.
X_discretized	in case of a single value of lambda, a numeric matrix containing the completed (i.e., imputed) data matrix after the discretization step. Otherwise a list of such matrices. This is only returned if requested via discretize = TRUE.

The class structure is still experimental and may change in the future. The following accessor functions are available:

- get_completed() to extract the completed (i.e., imputed) data matrix for a specified value of the regularization parameter,
- get_lambda() to extract the values of the regularization parameter.

Author(s)

Andreas Alfons and Aurore Archimbaud

References

Hastie, T., Mazumder, R., Lee, J. D. and Zadeh, R. (2015) Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares. *Journal of Machine Learning Research*, **16**(104), 3367–3402.

Mazumder, R., Hastie, T. and Tibshirani, R. (2010) Spectral Regularization Algorithms for Learning Large Incomplete Matrices. *Journal of Machine Learning Research*, **11**(80), 2287–2322.

See Also

soft_impute_tune(), fraction_grid()

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# Soft-Impute with discretization step
fit <- soft_impute(MovieLensToy)
# extract discretized completed matrix with fifth value
# of regularization parameter
X_hat <- get_completed(fit, which = 5)
head(X_hat)
```

---

soft_impute_tune *Matrix completion via nuclear-norm regularization with hyperparameter tuning*

Description

Perform matrix completion via nuclear-norm regularization based on softImpute(). The regularization parameter is thereby selected via repeated holdout validation or cross-validation. Note that this uses the convenience wrapper soft_impute(), whose default behavior is different from that of the original function.

Usage

```
soft_impute_tune(
  X,
  lambda = fraction_grid(reverse = TRUE),
  relative = TRUE,
  splits = holdout_control(),
  ...,
  discretize = TRUE,
  values = NULL
)
```

Arguments

X	a matrix or data frame with missing values.
lambda	a numeric vector giving values of the regularization parameter. See fraction_grid() for the default values.
relative	a logical indicating whether the values of the regularization parameter should be considered relative to a certain reference value computed from the data at hand. If TRUE (the default), the values of lambda are multiplied with the value returned by lambda0() (applied to the mean-centered data matrix).

splits	an object inheriting from class "split_control", as generated by holdout_control() for repeated holdout validation or cv_folds_control() for K-fold cross-validation, or a list of index vectors giving different validation sets of observed cells as generated by create_splits(). Cells in the validation set will be set to NA for fitting the algorithm with the training set of observed cells.
...	additional arguments to be passed down to soft_impute().
discretize	a logical indicating whether to include a discretization step after fitting the algorithm (defaults to TRUE). In case of discrete rating-scale data, this can be used to map the imputed values to the discrete rating scale of the observed values.
values	an optional numeric vector giving the possible values of discrete ratings. This is ignored if discretize is FALSE. Currently, the possible values are assumed to be the same for all columns. If NULL, the unique values of the observed parts of X are used.

Value

An object of class "soft_impute_tuned" with the following components:

lambda	a numeric vector containing the values of the regularization parameter.
tuning_loss	a numeric vector containing the (average) values of the loss function on the validation set(s) for each value of the regularization parameter.
lambda_opt	numeric; the optimal value of the regularization parameter.
fit	an object of class "soft_impute" containing the results from the algorithm with the optimal regularization parameter on the full (observed) data matrix.

The class structure is still experimental and may change in the future. The following accessor functions are available:

- get_completed() to extract the imputed data matrix (with the optimal value of the regularization parameter),
- get_lambda() to extract the optimal value of the regularization parameter.

Author(s)

Andreas Alfons

References

Hastie, T., Mazumder, R., Lee, J. D. and Zadeh, R. (2015) Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares. *Journal of Machine Learning Research*, **16**(104), 3367–3402.

Mazumder, R., Hastie, T. and Tibshirani, R. (2010) Spectral Regularization Algorithms for Learning Large Incomplete Matrices. *Journal of Machine Learning Research*, **11**(80), 2287–2322.

See Also

soft_impute(), fraction_grid(),

holdout_control(), cv_folds_control(), create_splits()

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# Soft-Impute with discretization step and hyperparameter tuning
set.seed(20250723)
fit <- soft_impute_tune(MovieLensToy,
                        lambda = fraction_grid(nb_lambda = 6,
                                               reverse = TRUE),
                        splits = holdout_control(R = 5))
# extract discretized completed matrix with optimal
# regularization parameter
X_hat <- get_completed(fit)
head(X_hat)
# extract optimal value of regularization parameter
get_lambda(fit)
```

`validation_control` *Control objects for hyperparameter validation*

Description

Construct control objects for repeated holdout validation or K-fold cross-validation.

Usage

```
holdout_control(pct = 0.1, R = 10L)

cv_folds_control(K = 5L)
```

Arguments

`pct`	numeric in the interval (0, 1); the percentage of observed cells in the data matrix to be randomly selected into the validation set (defaults to 0.1).
`R`	an integer giving the number of random splits into training and validation sets (defaults to 10).
`K`	an integer giving the number of cross-validation folds (defaults to 5).

Value

An object inheriting from class `"split_control"` containing the relevant information for splitting the the observed cells of a data matrix into training and validation sets for hyperparameter tuning.

The subclass `"holdout_control"` returned by `holdout_control()` is a list with components `pct` and `R` containing the corresponding argument values after validity checks.

The subclass `"cv_folds_control"` returned by `cv_folds_control()` is a list with a single component `K` containing the corresponding argument value after validity checks.

See Also

create_splits(),

rdmc_tune(), soft_impute_tune()

Examples

```
# toy example derived from MovieLens 100K dataset
data("MovieLensToy")
# robust discrete matrix completion with hyperparameter tuning
set.seed(20250723)
fit <- rdmc_tune(MovieLensToy,
                 lambda = fraction_grid(nb_lambda = 6),
                 splits = holdout_control(R = 5))
# extract optimal value of regularization parameter
get_lambda(fit)
```

Index