Package ‘sgdGMF’

May 18, 2025

Type Package

Title Estimation of Generalized Matrix Factorization Models via
Stochastic Gradient Descent

Version 1.0.1

Date 2025-05-17

Description

Efficient framework to estimate high-dimensional generalized matrix factorization models using penalized maximum likelihood under a dispersion exponential family specification. Either deterministic and stochastic methods are implemented for the numerical maximization. In particular, the package implements the stochastic gradient descent algorithm with a block-wise mini-batch strategy to speed up the computations and an efficient adaptive learning rate schedule to stabilize the convergence. All the theoretical details can be found in Castiglione et al. (2024, <doi:10.48550/arXiv.2412.20509>). Other methods considered for the optimization are the alternated iterative re-weighted least squares and the quasi-Newton method with diagonal approximation of the Fisher information matrix discussed in Kidzinski et al. (2022, <http://jmlr.org/papers/v23/20-1104.html>).

License MIT + file LICENSE

Imports Rcpp (>= 1.0.10), RcppArmadillo, RSpectra, parallel,
doParallel, foreach, MASS, SuppDists, methods, generics,
reshape2, ggpubr, viridisLite

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 4.0.0), ggplot2

Suggests testthat (>= 3.0.0), Rtsne, dplyr, knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

URL https://github.com/CristianCastiglione/sgdGMF

BugReports https://github.com/CristianCastiglione/sgdGMF/issues

RoxygenNote 7.2.3

VignetteBuilder knitr

NeedsCompilation yes

Author Cristian Castiglione [aut, cre] (ORCID:
<https://orcid.org/0000-0001-5883-4890>),
Davide Risso [ctb] (ORCID: <https://orcid.org/0000-0001-8508-5012>),
Alexandre Segers [ctb] (ORCID: <https://orcid.org/0009-0004-2028-7595>)

Maintainer Cristian Castiglione <cristian_castiglione@libero.it>

Repository CRAN

Date/Publication 2025-05-17 22:00:02 UTC

Contents

`biplot.initgmf` *Biplot of an initialized GMF model*

Description

Plot the observations on a two-dimensional projection determined by the estimated score matrix

Usage

```
## S3 method for class 'initgmf'
biplot(
  x,
  ...,
  choices = 1:2,
  arrange = TRUE,
  byrow = FALSE,
  normalize = FALSE,
  labels = NULL,
  palette = NULL
)
```

Arguments

`x`	an object of class `initgmf`
`...`	further arguments passed to or from other methods
`choices`	a length 2 vector specifying the components to plot
`arrange`	if `TRUE`, return a single plot with two panels
`byrow`	if `TRUE`, the panels are arranged row-wise (if `arrange=TRUE`)
`normalize`	if `TRUE`, orthogonalizes the scores using SVD
`labels`	a vector of labels which should be plotted
`palette`	the color-palette which should be used

Value

If `arrange=TRUE`, a single ggplot object with the selected biplots, otherwise, a list of two ggplot objects showing the row and column latent variables.

See Also

`biplot.sgdgmf`.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Get the biplot of a GMF model
biplot(init) # 1st vs 2nd principal components
biplot(init, choices = 2:3) #2nd vs 3rd principal components
```

---

biplot.sgdgmf *Biplot of a GMF model*

Description

Plot the observations on a two-dimensional projection determined by the estimated score matrix

Usage

```
## S3 method for class 'sgdgmf'
biplot(
  x,
  ...,
  choices = 1:2,
  arrange = TRUE,
  byrow = FALSE,
  normalize = FALSE,
  labels = NULL,
  palette = NULL,
  titles = c(NULL, NULL)
)
```

Arguments

`x`	an object of class `sgdgmf`
`...`	further arguments passed to or from other methods
`choices`	a length 2 vector specifying the components to plot
`arrange`	if `TRUE`, return a single plot with two panels
`byrow`	if `TRUE`, the panels are arranged row-wise (if `arrange=TRUE`)
`normalize`	if `TRUE`, orthogonalizes the scores using SVD
`labels`	a vector of labels which should be plotted
`palette`	the color-palette which should be used
`titles`	a 2-dimensional string vector containing the plot titles

Value

If arrange=TRUE, a single ggplot object with the selected biplots, otherwise, a list of two ggplot objects showing the row and column latent variables.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Get the biplot of a GMF model
biplot(gmf)
```

---

coefficients.initgmf *Extract the coefficient of an initialized GMF model*

Description

Return the initialized coefficients of a GMF model, i.e., the row- and column-specific regression effects, the latent scores and loadings.

Usage

```
## S3 method for class 'initgmf'
coefficients(
  object,
  ...,
  type = c("all", "colreg", "rowreg", "scores", "loadings")
)

## S3 method for class 'initgmf'
coef(object, ..., type = c("all", "colreg", "rowreg", "scores", "loadings"))
```

Arguments

object	an object of class initgmf
...	further arguments passed to or from other methods
type	the type of coefficients which should be returned

Value

If type="all", a list of coefficients containing the fields B, A, U and V. Otherwise, a matrix of coefficients, corresponding to the selected type.

See Also

coefficients.sgdgmf and coef.sgdgmf.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model with 3 latent factors
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Get the estimated coefficients of a GMF model
str(coefficients(init)) # returns all the coefficients
str(coefficients(init, type = "scores")) # returns only the scores, say U
str(coefficients(init, type = "loadings")) # returns only the loadings, say V
```

coefficients.sgdgmf *Extract the coefficient of a GMF model*

Description

Return the estimated coefficients of a GMF model, i.e., the row- and column-specific regression effects, the latent scores and loadings.

Usage

```
## S3 method for class 'sgdgmf'
coefficients(
  object,
  ...,
  type = c("all", "colreg", "rowreg", "scores", "loadings")
)

## S3 method for class 'sgdgmf'
coef(object, ..., type = c("all", "colreg", "rowreg", "scores", "loadings"))
```

Arguments

object	an object of class sgdgmf
...	further arguments passed to or from other methods
type	the type of coefficients which should be returned

Value

If type="all", a list of coefficients containing the fields B, A, U and V. Otherwise, a matrix of coefficients, corresponding to the selected type.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model with 3 latent factors
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Get the estimated coefficients of a GMF model
str(coefficients(gmf)) # returns all the coefficients
str(coefficients(gmf, type = ”scores”)) # returns only the scores, say U
str(coefficients(gmf, type = ”loadings”)) # returns only the loadings, say V
```

deviance.initgmf *Compute deviance, AIC and BIC of an initialized GMF model*

Description

Compute deviance, AIC and BIC of an initialized GMF object

Usage

```
## S3 method for class 'initgmf'
deviance(object, ..., normalize = FALSE)

## S3 method for class 'initgmf'
AIC(object, ..., k = 2)

## S3 method for class 'initgmf'
BIC(object, ...)
```

Arguments

object	an object of class initgmf
...	further arguments passed to or from other methods
normalize	if TRUE, normalize the result using the null-deviance
k	the penalty parameter to be used for AIC; the default is k = 2

Value

The value of the deviance extracted from a `initgmf` object.

See Also

`deviance.sgdgmf`, `AIC.sgdgmf` and `AIC.sgdgmf`.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model with 3 latent factors
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Get the GMF deviance, AIC and BIC
deviance(init)
AIC(init)
BIC(init)
```

deviance.sgdgmf *Compute deviance, AIC and BIC of a GMF model*

Description

Compute deviance, AIC and BIC of a GMF object

Usage

```
## S3 method for class 'sgdgmf'
deviance(object, ..., normalize = FALSE)

## S3 method for class 'sgdgmf'
AIC(object, ..., k = 2)

## S3 method for class 'sgdgmf'
BIC(object, ...)
```

Arguments

`object`	an object of class sgdgmf
`...`	further arguments passed to or from other methods
`normalize`	if `TRUE`, normalize the result using the null-deviance
`k`	the penalty parameter to be used for AIC; the default is `k = 2`

Value

The value of the deviance extracted from a sgdgmf object.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model with 3 latent factors
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Get the GMF deviance, AIC and BIC
deviance(gmf)
AIC(gmf)
BIC(gmf)
```

---

fitted.initgmf *Extract the fitted values of an initialized GMF model*

Description

Computes the fitted values of an initialized GMF model.

Usage

```
## S3 method for class 'initgmf'
fitted(object, ..., type = c("link", "response", "terms"), partial = FALSE)
```

Arguments

object	an object of class initgmf
...	further arguments passed to or from other methods
type	the type of fitted values which should be returned
partial	if TRUE, returns the partial fitted values

Value

If type="terms", a list of fitted values containing the fields XB, AZ and UV. Otherwise, a matrix of fitted values in the link or response scale, depending on the selected type.

See Also

fitted.sgdgmf.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model with 3 latent factors
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Get the fitted values of a GMF model
str(fitted(init)) # returns the overall fitted values in link scale
str(fitted(init, type = "response")) # returns the overall fitted values in response scale
str(fitted(init, partial = TRUE)) # returns the partial fitted values in link scale
```

fitted.sgdgmf *Extract the fitted values of a GMF models*

Description

Computes the fitted values of a GMF model.

Usage

```
## S3 method for class 'sgdgmf'
fitted(object, ..., type = c("link", "response", "terms"), partial = FALSE)
```

Arguments

object	an object of class sgdgmf
...	further arguments passed to or from other methods
type	the type of fitted values which should be returned
partial	if TRUE, returns the partial fitted values

Value

If type="terms", a list of fitted values containing the fields XB, AZ and UV. Otherwise, a matrix of fitted values in the link or response scale, depending on the selected type.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())
```

```
# Fit a GMF model with 3 latent factors
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Get the fitted values of a GMF model
str(fitted(gmf)) # returns the overall fitted values in link scale
str(fitted(gmf, type = "response")) # returns the overall fitted values in response scale
```

`image.initgmf` *Heatmap of an initialized GMF model*

Description

Plots a heatmap of either the data, the fitted values, or the residual values of a GMF model allowing for different types of transformations and normalizations. Moreover, it also permits to plot the latent score and loading matrices.

Usage

```
## S3 method for class 'initgmf'
image(
  x,
  ...,
  type = c("data", "response", "link", "scores", "loadings", "deviance", "pearson",
    "working"),
  resid = FALSE,
  symmetric = FALSE,
  transpose = FALSE,
  limits = NULL,
  palette = NULL
)
```

Arguments

`x`	an object of class `initgmf`
`...`	further arguments passed to or from other methods
`type`	the type of data/predictions/residuals which should be returned
`resid`	if `TRUE`, plots the residual values
`symmetric`	if `TRUE`, symmetrizes the color limits
`transpose`	if `TRUE`, transposes the matrix before plotting it
`limits`	the color limits which should be used
`palette`	the color-palette which should be used

Value

A ggplot object showing the selected heatmap.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Get the heatmap of a GMF model
image(init, type = "data") # original data
image(init, type = "response") # fitted values in response scale
image(init, type = "scores") # estimated score matrix
image(init, type = "loadings") # estimated loading matrix
image(init, type = "deviance", resid = TRUE) # deviance residual matrix
```

image.sgdgmf *Heatmap of a GMF model*

Description

Plots a heatmap of either the data, the fitted values, or the residual values of a GMF model allowing for different types of transformations and normalizations. Moreover, it also permits to plot the latent score and loading matrices.

Usage

```
## S3 method for class 'sgdgmf'
image(
  x,
  ...,
  type = c("data", "response", "link", "scores", "loadings", "deviance", "pearson",
    "working"),
  resid = FALSE,
  symmetric = FALSE,
  transpose = FALSE,
  limits = NULL,
  palette = NULL
)
```

Arguments

x	an object of class sgdgmf
...	further arguments passed to or from other methods
type	the type of data/predictions/residuals which should be returned

resid	if TRUE, plots the residual values
symmetric	if TRUE, symmetrizes the color limits
transpose	if TRUE, transposes the matrix before plotting it
limits	the color limits which should be used
palette	the color-palette which should be used

Value

A ggplot object showing the selected heatmap.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

 # Fit a GMF model
 gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Get the heatmap of a GMF model
image(gmf, type = "data") # original data
image(gmf, type = "response") # fitted values in response scale
image(gmf, type = "scores") # estimated score matrix
image(gmf, type = "loadings") # estimated loading matrix
image(gmf, type = "deviance", resid = TRUE) # deviance residual matrix
```

plot.initgmf *Plot diagnostics for an initialized GMF model*

Description

Plots (one of) six diagnostics to graphically analyze the marginal and conditional distribution of the residuals of a GMF model. Currently, the following plots are available: residuals against observation indices, residuals agains fitted values, absolute square-root residuals against fitted values, histogram of the residuals, residual QQ-plot, residual ECDF-plot.

Usage

```
## S3 method for class 'initgmf'
plot(
  x,
  ...,
  type = c("res-idx", "res-fit", "std-fit", "hist", "qq", "ecdf"),
  resid = c("deviance", "pearson", "working", "response", "link"),
  subsample = FALSE,
  sample.size = 500,
  partial = FALSE,
  normalize = FALSE,
  fillna = FALSE
)
```

Arguments

`x`	an object of class `initgmf`
`...`	further arguments passed to or from other methods
`type`	the type of plot which should be returned
`resid`	the type of residuals which should be used
`subsample`	if `TRUE`, computes the residuals over o small fraction of the data
`sample.size`	the dimension of the sub-sample which should be used
`partial`	if `TRUE`, computes the partial residuals
`normalize`	if `TRUE`, standardizes the residuals column-by-column
`fillna`	if `TRUE`, fills the `NA` values with `0`

Value

A ggplot object showing the selected diagnostic plot.

See Also

`plot.sgdgmf`.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Plot the residual-based GMF diagnostics
plot(init, type = "res-fit") # Residuals vs fitted values
plot(init, type = "std-fit") # Abs-sqrt-transformed residuals vs fitted values
plot(init, type = "qq") # Residual QQ-plot
plot(init, type = "hist") # Residual histogram
```

plot.sgdgmf *Plot diagnostics for a GMF model*

Description

Plots (one of) six diagnostics to graphically analyze the marginal and conditional distribution of the residuals of a GMF model. Currently, the following plots are available: residuals against observation indices, residuals agains fitted values, absolute square-root residuals against fitted values, histogram of the residuals, residual QQ-plot, residual ECDF-plot.

Usage

```
## S3 method for class 'sgdgmf'
plot(
  x,
  ...,
  type = c("res-idx", "res-fit", "std-fit", "hist", "qq", "ecdf"),
  resid = c("deviance", "pearson", "working", "response", "link"),
  subsample = FALSE,
  sample.size = 500,
  partial = FALSE,
  normalize = FALSE,
  fillna = FALSE
)
```

Arguments

`x`	an object of class `sgdgmf`
`...`	further arguments passed to or from other methods
`type`	the type of plot which should be returned
`resid`	the type of residuals which should be used
`subsample`	if `TRUE`, computes the residuals over o small fraction of the data
`sample.size`	the dimension of the sub-sample which should be used
`partial`	if `TRUE`, computes the partial residuals
`normalize`	if `TRUE`, standardizes the residuals column-by-column
`fillna`	if `TRUE`, fills the `NA` values with `0`

Value

A ggplot object showing the selected diagnostic plot.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Plot the residual-based GMF diagnostics
plot(gmf, type = "res-fit") # Residuals vs fitted values
plot(gmf, type = "std-fit") # Abs-sqrt-transformed residuals vs fitted values
plot(gmf, type = "qq") # Residual QQ-plot
plot(gmf, type = "hist") # Residual histogram
```

predict.sgdgmf *Predict method for GMF models*

Description

Computes the predictions of a GMF model. Out-of-sample predictions for a new set of responses and covariates are computed via MLE, by keeping fixed the values of the estimated B and V and maximizing the likelihood with respect to A and U.

Usage

```
## S3 method for class 'sgdgmf'
predict(
  object,
  ...,
  newY = NULL,
  newX = NULL,
  type = c("link", "response", "terms", "coef"),
  parallel = FALSE,
  nthreads = 1
)
```

Arguments

object	an object of class sgdgmf
...	further arguments passed to or from other methods
newY	optionally, a matrix of new response variable
newX	optionally, a matrix of new covariate values
type	the type of prediction which should be returned
parallel	if TRUE, allows for parallel computing using the package foreach
nthreads	number of cores to be used in parallel (only if parallel=TRUE)

Details

If newY and newX are omitted, the predictions are based on the data used for the fit. In that case, the predictions corresponds to the fitted values. If newY and newX are provided, a corresponding set of A and U are estimated via maximum likelihood using the glm.fit function. By doing so, B and V are kept fixed.

Value

If type="link" or typr="response", a matrix of predictions. If type="terms", a list of matrices containing the fields XB, AZ and UV. If type="coef", a list of matrices containing the field B, A, U and V.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 120, m = 20, ncomp = 5, family = poisson())
train = sample(1:120, size = 100)
test = setdiff(1:120, train)

Y = data$Y[train, ]
newY = data$Y[test, ]

# Fit a GMF model with 3 latent factors
gmf = sgdgmf.fit(Y, ncomp = 3, family = poisson())

# Get the fitted values of a GMF model
str(predict(gmf)) # returns the overall fitted values in link scale
str(predict(gmf, type = "response")) # returns the overall fitted values in response scale
str(predict(gmf, type = "terms")) # returns the partial fitted values in link scale
str(predict(gmf, newY = newY)) # returns the predictions for the new set of responses
```

---

print.initgmf *Print the fundamental characteristics of an initialized GMF*

Description

Print some summary information of an initialized GMF model.

Usage

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

Arguments

`x`	an object of class `initgmf`
`...`	further arguments passed to or from other methods

Value

No return value, called only for printing.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model with 3 latent factors
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Print the GMF object
print(init)
```

`print.sgdgmf` *Print the fundamental characteristics of a GMF*

Description

Print some summary information of a GMF model.

Usage

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

Arguments

`x`	an object of class `sgdgmf`
`...`	further arguments passed to or from other methods

Value

No return value, called only for printing.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model with 3 latent factors
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Print the GMF object
print(gmf)
```

`refit.sgdgmf` *Refine the final estimate of a GMF model*

Description

Refine the estimated latent scores of a GMF model via IRWLS

Usage

```
## S3 method for class 'sgdgmf'
refit(
  object,
  ...,
  normalize = TRUE,
  verbose = FALSE,
  parallel = FALSE,
  nthreads = 1
)
```

Arguments

`object`	an object of class `sgdgmf`
`...`	further arguments passed to or from other methods
`normalize`	if `TRUE`, normalize `U` and `V` to uncorrelated Gaussian `U` and upper triangular `V` with positive diagonal
`verbose`	if `TRUE`, print the optimization status
`parallel`	if `TRUE`, use parallel computing using the `foreach` package
`nthreads`	number of cores to be used in the `"glm"` method

Value

An `sgdgmf` object containing the re-fitted model.

See Also

sgdgmf.fit

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model using SGD
gmf_old = sgdgmf.fit(data$Y, ncomp = 3, family = poisson(), method = "sgd")

# Refine the score matrix estimate
gmf_new = refit(gmf_old)

# Get the fitted values in the link and response scales
mu_hat_old = fitted(gmf_old, type = "response")
mu_hat_new = fitted(gmf_new, type = "response")

# Compare the results
oldpar = par(no.readonly = TRUE)
par(mfrow = c(2,2), mar = c(1,1,3,1))
image(data$Y, axes = FALSE, main = expression(Y))
image(data$mu, axes = FALSE, main = expression(mu))
image(mu_hat_old, axes = FALSE, main = expression(hat(mu)[old]))
image(mu_hat_new, axes = FALSE, main = expression(hat(mu)[new]))
par(oldpar)
```

residuals.initgmf *Extract the residuals of an initialized GMF model*

Description

Extract the residuals of an initialized GMF model and, if required, compute the eigenvalues of the residuals covariance/correlation matrix. Moreover, if required, return the partial residual of the model obtained by excluding the matrix decomposition from the linear predictor.

Usage

```
## S3 method for class 'initgmf'
residuals(
  object,
  ...,
  type = c("deviance", "pearson", "working", "response", "link"),
  partial = FALSE,
  normalize = FALSE,
  fillna = FALSE,
  spectrum = FALSE,
  ncomp = 50
)

## S3 method for class 'initgmf'
resid(
  object,
  ...,
  type = c("deviance", "pearson", "working", "response", "link"),
  partial = FALSE,
  normalize = FALSE,
  fillna = FALSE,
  spectrum = FALSE,
  ncomp = 50
)
```

Arguments

`object`	an object of class `initgmf`
`...`	further arguments passed to or from other methods
`type`	the type of residuals which should be returned
`partial`	if `TRUE`, computes the residuals excluding the matrix factorization from the linear predictor
`normalize`	if `TRUE`, standardize the residuals column-by-column
`fillna`	if `TRUE`, fills `NA` values column-by-column
`spectrum`	if `TRUE`, returns the eigenvalues of the residual covariance matrix
`ncomp`	number of eigenvalues to be calculated (only if `spectrum=TRUE`)

Value

If `spectrum=FALSE`, a matrix containing the selected residuals. If `spectrum=TRUE`, a list containing the residuals (`res`), the first `ncomp` eigenvalues of the residual covariance matrix, say (`lambdas`), the variance explained by the first `ncomp` principal component of the residuals (`explained.var`), the variance not explained by the first `ncomp` principal component of the residuals (`residual.var`), the total variance of the residuals (`total.var`).

See Also

`residuals.sgdgmf` and `resid.sgdgmf` for more details on the residual computation.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())
```

```
# Fit a GMF model with 3 latent factors
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Get the deviance residuals of a GMF model
str(residuals(init)) # returns the overall deviance residuals
str(residuals(init, partial = TRUE)) # returns the partial residuals
str(residuals(init, spectrum = TRUE)) # returns the eigenvalues of the residual var-cov matrix
```

residuals.sgdgmf *Extract the residuals of a GMF model*

Description

Extract the residuals of a GMF model and, if required, compute the eigenvalues of the residuals covariance/correlation matrix. Moreover, if required, return the partial residual of the model obtained by excluding the matrix decomposition from the linear predictor.

Usage

```
## S3 method for class 'sgdgmf'
residuals(
  object,
  ...,
  type = c("deviance", "pearson", "working", "response", "link"),
  partial = FALSE,
  normalize = FALSE,
  fillna = FALSE,
  spectrum = FALSE,
  ncomp = 50
)

## S3 method for class 'sgdgmf'
resid(
  object,
  ...,
  type = c("deviance", "pearson", "working", "response", "link"),
  partial = FALSE,
  normalize = FALSE,
  fillna = FALSE,
  spectrum = FALSE,
  ncomp = 50
)
```

Arguments

`object` an object of class `sgdgmf`

`...` further arguments passed to or from other methods

`type` the type of residuals which should be returned

`partial` if `TRUE`, computes the residuals excluding the matrix factorization from the linear predictor

`normalize` if `TRUE`, standardize the residuals column-by-column

`fillna` if `TRUE`, fills `NA` values column-by-column

`spectrum` if `TRUE`, returns the eigenvalues of the residual covariance matrix

`ncomp` number of eigenvalues to be calculated (only if `spectrum=TRUE`)

Details

Let $g(\mu) = \eta = XB^\top + \Gamma Z^\top + UV^\top$ be the linear predictor of a GMF model. Let $R = (r_{ij})$ be the correspondent residual matrix. The following residuals can be considered:

- deviance: $r_{ij}^D = \text{sign}(y_{ij} - \mu_{ij})\sqrt{D(y_{ij}, \mu_{ij})}$;
- Pearson: $r_{ij}^P = (y_{ij} - \mu_{ij})/\sqrt{\nu(\mu_{ij})}$;
- working: $r_{ij}^W = (y_{ij} - \mu_{ij})/\{g'(\mu_{ij})\,\nu(\mu_{ij})\}$;
- response: $r_{ij}^R = y_{ij} - \mu_{ij}$;
- link: $r_{ij}^G = g(y_{ij}) - \eta_{ij}$.

If `partial=TRUE`, mu is computed excluding the latent matrix decomposition from the linear predictor, so as to obtain the partial residuals.

Let Σ be the empirical variance-covariance matrix of R, being $\sigma_{ij} = \text{Cov}(r_{:i}, r_{:j})$. Then, the latent spectrum of the model is the collection of eigenvalues of Σ.

Notice that, in case of Gaussian data, the latent spectrum corresponds to the principal component analysis on the regression residuals, whose eigenvalues can be used to infer the amount of variance explained by each principal component. Similarly, we can use the (partial) latent spectrum in non-Gaussian data settings to infer the correct number of principal components to include into the GMF model or to detect some residual dependence structures not already explained by the model.

Value

If `spectrum=FALSE`, a matrix containing the selected residuals. If `spectrum=TRUE`, a list containing the residuals (`res`), the first `ncomp` eigenvalues of the residual covariance matrix, say (`lambdas`), the variance explained by the first `ncomp` principal component of the residuals (`explained.var`), the variance not explained by the first `ncomp` principal component of the residuals (`residual.var`), the total variance of the residuals (`total.var`).

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model with 3 latent factors
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Get the deviance residuals of a GMF model
str(residuals(gmf)) # returns the overall deviance residuals
str(residuals(gmf, partial = TRUE)) # returns the partial residuals
str(residuals(gmf, spectrum = TRUE)) # returns the eigenvalues of the residual var-cov matrix
```

screeplot.initgmf *Screeplot for the residuals of an initialized GMF model*

Description

Plots the variances of the principal components of the residuals against the number of principal component.

Usage

```
## S3 method for class 'initgmf'
screeplot(
  x,
  ...,
  ncomp = 20,
  type = c("deviance", "pearson", "working", "response", "link"),
  partial = FALSE,
  normalize = FALSE,
  cumulative = FALSE,
  proportion = FALSE
)
```

Arguments

`x`	an object of class `sgdgmf`
`...`	further arguments passed to or from other methods
`ncomp`	number of components to be plotted
`type`	the type of residuals which should be used
`partial`	if `TRUE`, plots the eigenvalues of the partial residuals
`normalize`	if `TRUE`, plots the eigenvalues of the standardized residuals
`cumulative`	if `TRUE`, plots the cumulative sum of the eigenvalues
`proportion`	if `TRUE`, plots the fractions of explained variance

Value

A ggplot object showing the residual screeplot of the model.

See Also

screeplot.sgdgmf.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model
init = sgdgmf.init(data$Y, ncomp = 3, family = poisson())

# Get the partial residual spectrum of a GMF model
screeplot(init) # screeplot of the var-cov matrix of the deviance residuals
screeplot(init, partial = TRUE) # screeplot of the partial residuals
screeplot(init, cumulative = TRUE) # cumulative screeplot
screeplot(init, proportion = TRUE) # proportion of explained residual variance
```

---

screeplot.sgdgmf *Screeplot for the residuals of a GMF model*

Description

Plots the variances of the principal components of the residuals against the number of principal component.

Usage

```
## S3 method for class 'sgdgmf'
screeplot(
  x,
  ...,
  ncomp = 20,
  type = c("deviance", "pearson", "working", "response", "link"),
  partial = FALSE,
  normalize = FALSE,
  cumulative = FALSE,
  proportion = FALSE
)
```

Arguments

`x`	an object of class `sgdgmf`
`...`	further arguments passed to or from other methods
`ncomp`	number of components to be plotted
`type`	the type of residuals which should be used
`partial`	if `TRUE`, plots the eigenvalues of the partial residuals
`normalize`	if `TRUE`, plots the eigenvalues of the standardized residuals
`cumulative`	if `TRUE`, plots the cumulative sum of the eigenvalues
`proportion`	if `TRUE`, plots the fractions of explained variance

Value

A ggplot object showing the residual screeplot of the model.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Get the partial residual spectrum of a GMF model
screeplot(gmf) # screeplot of the var-cov matrix of the deviance residuals
screeplot(gmf, partial = TRUE) # screeplot of the partial residuals
screeplot(gmf, cumulative = TRUE) # cumulative screeplot
screeplot(gmf, proportion = TRUE) # proportion of explained residual variance
```

`set.control.airwls` *Check and set the control parameters for the AIRWLS algorithm*

Description

Check if the input control parameters of the AIRWLS algorithm are allowed and set them to default values if they are not. Returns a list of well-defined control parameters.

Usage

```
set.control.airwls(
  normalize = TRUE,
  maxiter = 100,
  nstep = 1,
  stepsize = 0.1,
  eps = 1e-08,
  nafill = 1,
  tol = 1e-05,
  damping = 0.001,
  verbose = FALSE,
  frequency = 10,
  parallel = FALSE,
  nthreads = 1
)
```

Arguments

`normalize` if `TRUE`, normalize `U` and `V` to uncorrelated Gaussian `U` and upper triangular `V` with positive diagonal

`maxiter` maximum number of iterations

`nstep` number of IRWLS steps in each inner loop of AIRWLS

`stepsize` step-size parameter scaling each IRWLS step

`eps` how much shrinkage has to be introduced on extreme predictions lying outside of the data range

`nafill` how frequently the `NA` values are filled, by default `NA` values are filled at each iteration of the algorithm

`tol` tolerance threshold for the stopping criterion

`damping` regularization parameter which is added to the diagonal of the Hessian to ensure numerical stability

`verbose` if `TRUE`, print the optimization status (default `TRUE`)

`frequency` how often the optimization status is printed (only if `verbose=TRUE`)

`parallel` if `TRUE`, allows for parallel computing using the `C++` library `OpenMP`

`nthreads` number of cores to be used in parallel (only if `parallel=TRUE`)

Value

A `list` of control parameters for the AIRWLS algorithm

Examples

```
library(sgdGMF)

# Empty call
set.control.airwls()
```

```
# Parametrized call
set.control.airwls(maxiter = 100, nstep = 5, stepsize = 0.3)
```

set.control.alg *Check and set the control parameters for the select optimization algorithm*

Description

Check if the input control parameters are allowed and set them to default values if they are not. Returns a list of well-defined control parameters.

Usage

```
set.control.alg(
  method = c("airwls", "newton", "sgd"),
  sampling = c("block", "coord", "rnd-block"),
  control = list()
)
```

Arguments

method	optimization method to use
sampling	sub-sampling method to use
control	list of algorithm-specific control parameters

Details

It is not necessary to provide a complete list of control parameters, one can just specify a list containing the parameters he/she needs to change from the default values. Wrongly specified parameters are ignored or set to default values. For a detailed description of all the algorithm-specific control parameters, please refer to set.control.airwls (method="airwls"), set.control.newton (method="newton"), set.control.block.sgd (method="sgd", sampling="block"). set.control.coord.sgd (method="sgd", sampling="coord"),

Value

A list of control parameters for the selected estimation algorithm

See Also

set.control.init, set.control.cv, sgdgmf.fit

Examples

```
library(sgdGMF)

# Empty call
set.control.alg()

# Parametrized call
set.control.alg(method = "airwls", control = list(maxiter = 200, stepsize = 0.3))
```

---

set.control.block.sgd *Check and set the control parameters for the blockwise-SGD algorithm*

Description

Check if the input control parameters are allowed and set them to default values if they are not. Returns a list of well-defined control parameters.

Usage

```
set.control.block.sgd(
  normalize = TRUE,
  maxiter = 1000,
  eps = 1e-08,
  nafill = 10,
  tol = 1e-08,
  size = c(100, 100),
  burn = 1,
  rate0 = 0.01,
  decay = 0.01,
  damping = 0.001,
  rate1 = 0.1,
  rate2 = 0.01,
  verbose = FALSE,
  frequency = 250,
  progress = FALSE
)
```

Arguments

`normalize`	if `TRUE`, normalize `U` and `V` to uncorrelated Gaussian `U` and upper triangular `V` with positive diagonal
`maxiter`	maximum number of iterations
`eps`	how much shrinkage has to be introduced on extreme predictions lying outside of the data range

`nafill` how frequently the `NA` values are filled, by default `NA` values are filled at each iteration of the algorithm

`tol` tolerance threshold for the stopping criterion

`size` mini-batch size, the first value is for row sub-sample, the second value is for column sub-sample

`burn` percentage of iterations to ignore before performing Polyak averaging

`rate0` initial learning rate

`decay` learning rate decay

`damping` regularization parameter which is added to the Hessian to ensure numerical stability

`rate1` exponential decay rate for the moment estimate of the gradient

`rate2` exponential decay rate for the moment estimate of the Hessian

`verbose` if `TRUE`, print the optimization status

`frequency` how often the optimization status is printed (only if `verbose=TRUE`)

`progress` if `TRUE`, print a compact progress-bar instead of a full-report of the optimization status (only if `verbose=TRUE`)

Value

A `list` of control parameters for the adaptive SGD algorithm with block-wise sub-sampling

Examples

```
library(sgdGMF)

# Empty call
set.control.block.sgd()

# Parametrized call
set.control.block.sgd(maxiter = 2000, rate0 = 0.01, decay = 0.01)
```

`set.control.coord.sgd` *Check and set the control parameters for the coordinate-SGD algorithm*

Description

Check if the input control parameters are allowed and set them to default values if they are not. Returns a list of well-defined control parameters.

Usage

```
set.control.coord.sgd(
  normalize = TRUE,
  maxiter = 1000,
  eps = 1e-08,
  nafill = 10,
  tol = 1e-08,
  size = c(100, 100),
  burn = 1,
  rate0 = 0.01,
  decay = 0.01,
  damping = 0.001,
  rate1 = 0.1,
  rate2 = 0.01,
  verbose = FALSE,
  frequency = 250,
  progress = FALSE
)
```

Arguments

`normalize`	if `TRUE`, normalize `U` and `V` to uncorrelated Gaussian `U` and upper triangular `V` with positive diagonal
`maxiter`	maximum number of iterations
`eps`	how much shrinkage has to be introduced on extreme predictions lying outside of the data range
`nafill`	how frequently the `NA` values are filled, by default `NA` values are filled at each iteration of the algorithm
`tol`	tolerance threshold for the stopping criterion
`size`	mini-batch size, the first value is for row sub-sample, the second value is for column sub-sample
`burn`	percentage of iterations to ignore before performing Polyak averaging
`rate0`	initial learning rate
`decay`	learning rate decay
`damping`	regularization parameter which is added to the Hessian to ensure numerical stability
`rate1`	exponential decay rate for the moment estimate of the gradient
`rate2`	exponential decay rate for the moment estimate of the Hessian
`verbose`	if `TRUE`, print the optimization status
`frequency`	how often the optimization status is printed (only if `verbose=TRUE`)
`progress`	if `TRUE`, print a compact progress-bar instead of a full-report of the optimization status (only if `verbose=TRUE`)

Value

A `list` of control parameters for the adaptive SGD algorithm with coordinate-wise sub-sampling

Examples

```
library(sgdGMF)

# Empty call
set.control.coord.sgd()

# Parametrized call
set.control.coord.sgd(maxiter = 2000, rate0 = 0.01, decay = 0.01)
```

set.control.cv *Check and set the cross-validation parameters*

Description

Check if the input cross-validation parameters are allowed and set them to default values if they are not. Returns a list of well-defined cross-validation parameters.

Usage

```
set.control.cv(
  criterion = c("dev", "mae", "mse", "aic", "bic"),
  refit = TRUE,
  nfolds = 5,
  proportion = 0.3,
  init = c("common", "separate"),
  verbose = FALSE,
  parallel = FALSE,
  nthreads = 1
)
```

Arguments

`criterion`	information criterion to minimize for selecting the matrix rank
`refit`	if `TRUE`, refit the model with the selected rank and return the fitted model
`nfolds`	number of cross-validation folds
`proportion`	proportion of the data to be used as test set in each fold
`init`	initialization approach to use
`verbose`	if `TRUE`, print the cross-validation status
`parallel`	if `TRUE`, allows for parallel computing
`nthreads`	number of cores to use in parallel (only if `parallel=TRUE`)

Value

A list of control parameters for the cross-validation algorithm

See Also

set.control.init, set.control.alg, sgdgmf.cv

Examples

```
library(sgdGMF)

# Empty call
set.control.cv()

# Parametrized call
set.control.cv(criterion = "bic", proportion = 0.2)
```

set.control.init *Check and set the initialization parameters for a GMF model*

Description

Check if the input initialization parameters are allowed and set them to default values if they are not. Returns a list of well-defined options which specify how to initialize a GMF model. See sgdgmf.init for more details upon the methods used for initialisation.

Usage

```
set.control.init(
  method = c("ols", "glm", "random", "values"),
  type = c("deviance", "pearson", "working", "link"),
  values = list(),
  niter = 5,
  normalize = TRUE,
  verbose = FALSE,
  parallel = FALSE,
  nthreads = 1
)
```

Arguments

method	initialization method (see sgdgmf.init for more details upon the initialization methods used)
type	residual type to be decomposed (see sgdgmf.init for more details upon the residuals used)
values	list of custom initialization parameters fixed by the user

`niter` number if refinement iterations in the "`svd`" method

`normalize` if `TRUE`, normalize `U` and `V` to orthogonal `U` and lower triangular `V`

`verbose` if `TRUE`, print the initialization state

`parallel` if `TRUE`, use parallel computing for the "`glm`" method

`nthreads` number of cores to be used in the "`glm`" method

Value

A `list` of control parameters for the initialization

See Also

`set.control.alg`, `set.control.cv`, `sgdgmf.init`

Examples

```
library(sgdGMF)

# Empty call
set.control.init()

# Parametrized call
set.control.init(method = "glm", type = "deviance", niter = 10)
```

`set.control.newton` *Check and set the control parameters for the Newton algorithm*

Description

Check if the input control parameters of the quasi-Newton algorithm are allowed and set them to default values if they are not. Returns a list of well-defined control parameters.

Usage

```
set.control.newton(
  normalize = TRUE,
  maxiter = 500,
  stepsize = 0.01,
  eps = 1e-08,
  nafill = 1,
  tol = 1e-05,
  damping = 0.001,
  verbose = FALSE,
  frequency = 50,
  parallel = FALSE,
  nthreads = 1
)
```

Arguments

normalize — if TRUE, normalize U and V to uncorrelated Gaussian U and upper triangular V with positive diagonal

maxiter — maximum number of iterations

stepsize — step-size parameter scaling each IRWLS step

eps — how much shrinkage has to be introduced on extreme predictions lying outside of the data range

nafill — how frequently the NA values are filled, by default NA values are filled at each iteration of the algorithm

tol — tolerance threshold for the stopping criterion

damping — regularization parameter which is added to the Hessian to ensure numerical stability

verbose — if TRUE, print the optimization status

frequency — how often the optimization status is printed (only if verbose=TRUE

parallel — if TRUE, allows for parallel computing using the C++ library OpenMP

nthreads — number of cores to be used in parallel (only if parallel=TTUE)

Value

A list of control parameters for the quasi-Newton algorithm

Examples

```
library(sgdGMF)

# Empty call
set.control.newton()

# Parametrized call
set.control.newton(maxiter = 1000, stepsize = 0.01, tol = 1e-04)
```

sgdgmf.cv — *Model selection via cross-validation for generalized matrix factorization models*

Description

K-fold cross-validation for generalized matrix factorization (GMF) models.

Usage

```
sgdgmf.cv(
  Y,
  X = NULL,
  Z = NULL,
  family = gaussian(),
  ncomps = seq(from = 1, to = 10, by = 1),
  weights = NULL,
  offset = NULL,
  method = c("airwls", "newton", "sgd"),
  sampling = c("block", "coord", "rnd-block"),
  penalty = list(),
  control.init = list(),
  control.alg = list(),
  control.cv = list()
)
```

Arguments

`Y` matrix of responses ($n \times m$)

`X` matrix of row fixed effects ($n \times p$)

`Z` matrix of column fixed effects ($q \times m$)

`family` a `glm` family (see `family` for more details)

`ncomps` ranks of the latent matrix factorization used in cross-validation (default 1 to 10)

`weights` an optional matrix of weights ($n \times m$)

`offset` an optional matrix of offset values ($n \times m$), that specify a known component to be included in the linear predictor.

`method` estimation method to minimize the negative penalized log-likelihood

`sampling` sub-sampling strategy to use if `method = "sgd"`

`penalty` list of penalty parameters (see `set.penalty` for more details)

`control.init` list of control parameters for the initialization (see `set.control.init` for more details)

`control.alg` list of control parameters for the optimization (see `set.control.alg` for more details)

`control.cv` list of control parameters for the cross-validation (see `set.control.cv` for more details)

Details

Cross-validation is performed by minimizing the estimated out-of-sample error, which can be measured in terms of averaged deviance, AIC or BIC calculated on fold-specific test sets. Within each fold, the test set is defined as a fixed proportion of entries in the response matrix which are held out from the estimation process. To this end, the test set entries are hidden by `NA` values when training the model. Then, the predicted, i.e. imputed, values are used to compute the fold-specific out-of-sample error.

Value

If refit = FALSE (see set.control.cv), the function returns a list containing control.init, control.alg, control.cv and summary.cv. The latter is a matrix collecting the cross-validation results for each combination of fold and latent dimension.

If refit = TRUE (see set.control.cv), the function returns an object of class sgdgmf, obtained by refitting the model on the whole data matrix using the latent dimension selected via cross-validation. The returned object also contains the summary.cv information along with the other standard output of the sgdgmf.fit function.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Set the data dimensions
n = 100; m = 20; d = 5

# Generate data using Poisson, Binomial and Gamma models
data_pois = sim.gmf.data(n = n, m = m, ncomp = d, family = poisson())
data_bin = sim.gmf.data(n = n, m = m, ncomp = d, family = binomial())
data_gam = sim.gmf.data(n = n, m = m, ncomp = d, family = Gamma(link = "log"), dispersion = 0.25)

# Set RUN = TRUE to run the example, it may take some time. To speed up
# the computation it is possible to run CV in parallel specifying
# control.cv = list(parallel = TRUE, nthreads = <number_of_workers>)
# as an argument of sgdgmf.cv()
RUN = FALSE
if (RUN) {
  # Initialize the GMF parameters assuming 3 latent factors
  gmf_pois = sgdgmf.cv(data_pois$Y, ncomp = 1:10, family = poisson())
  gmf_bin = sgdgmf.cv(data_bin$Y, ncomp = 3, family = binomial())
  gmf_gam = sgdgmf.cv(data_gam$Y, ncomp = 3, family = Gamma(link = "log"))

  # Get the fitted values in the link and response scales
  mu_hat_pois = fitted(gmf_pois, type = "response")
  mu_hat_bin = fitted(gmf_bin, type = "response")
  mu_hat_gam = fitted(gmf_gam, type = "response")

  # Compare the results
  oldpar = par(no.readonly = TRUE)
  par(mfrow = c(1,3), mar = c(1,1,3,1))
  image(data_pois$Y, axes = FALSE, main = expression(Y[Pois]))
  image(data_pois$mu, axes = FALSE, main = expression(mu[Pois]))
  image(mu_hat_pois, axes = FALSE, main = expression(hat(mu)[Pois]))
  image(data_bin$Y, axes = FALSE, main = expression(Y[Bin]))
  image(data_bin$mu, axes = FALSE, main = expression(mu[Bin]))
  image(mu_hat_bin, axes = FALSE, main = expression(hat(mu)[Bin]))
  image(data_gam$Y, axes = FALSE, main = expression(Y[Gam]))
  image(data_gam$mu, axes = FALSE, main = expression(mu[Gam]))
  image(mu_hat_gam, axes = FALSE, main = expression(hat(mu)[Gam]))
  par(oldpar)
}
```

sgdgmf.fit *Factorize a matrix of non-Gaussian observations using GMF*

Description

Fit a generalized matrix factorization (GMF) model for non-Gaussian data using either deterministic or stochastic optimization methods. It is an alternative to PCA when the observed data are binary, counts, and positive scores or, more generally, when the conditional distribution of the observations can be appropriately described using a dispersion exponential family or a quasi-likelihood model. Some examples are Gaussian, Gamma, Binomial and Poisson probability laws.

The dependence among the observations and the variables in the sample can be taken into account through appropriate row- and column-specific regression effects. The residual variability is then modeled through a low-rank matrix factorization.

For the estimation, the package implements two deterministic optimization methods, (AIRWLS and Newton) and two stochastic optimization algorithms (adaptive SGD with coordinate-wise and block-wise sub-sampling).

Usage

```
sgdgmf.fit(
  Y,
  X = NULL,
  Z = NULL,
  family = gaussian(),
  ncomp = 2,
  weights = NULL,
  offset = NULL,
  method = c(”airwls”, ”newton”, ”sgd”),
  sampling = c(”block”, ”coord”, ”rnd-block”),
  penalty = list(),
  control.init = list(),
  control.alg = list()
)
```

Arguments

`Y`	matrix of responses ($n \times m$)
`X`	matrix of row fixed effects ($n \times p$)
`Z`	matrix of column fixed effects ($q \times m$)
`family`	a `glm` family (see `family` for more details)
`ncomp`	rank of the latent matrix factorization (default 2)
`weights`	an optional matrix of weights ($n \times m$)

`offset` an optional matrix of offset values ($n \times m$), that specify a known component to be included in the linear predictor

`method` estimation method to minimize the negative penalized log-likelihood

`sampling` sub-sampling strategy to use if `method = "sgd"`

`penalty` list of penalty parameters (see `set.penalty` for more details)

`control.init` list of control parameters for the initialization (see `set.control.init` for more details)

`control.alg` list of control parameters for the optimization (see `set.control.alg` for more details)

Details

Model specification

The model we consider is defined as follows. Let $Y = (y_{ij})$ be a matrix of observed data of dimension $n \times m$. We assume for the (i, j)th observation in the matrix a dispersion exponential family law $(y_{ij} \mid \theta_{ij}) \sim EF(\theta_{ij}, \phi)$, where θ_{ij} is the natural parameter and ϕ is the dispersion parameter. Recall that the conditional probability density function of y_{ij} is given by

$$f(y_{ij}; \psi) = \exp\left[w_{ij}\{(y_{ij}\theta_{ij} - b(\theta_{ij})\}/\phi - c(y_{ij}, \phi/w_{ij})\right],$$

where ψ is the vector of unknown parameters to be estimated, $b(\cdot)$ is a convex twice differentiable log-partition function, and $c(\cdot, \cdot)$ is the cumulant function of the family.

The conditional mean of y_{ij}, say μ_{ij}, is then modeled as

$$g(\mu_{ij}) = \eta_{ij} = x_i^\top \beta_j + \alpha_i^\top z_j + u_i^\top v_j,$$

where $g(\cdot)$ is a bijective twice differentiable link function, η_{ij} is a linear predictor, $x_i \in \mathbb{R}^p$ and $z_j \in \mathbb{R}^q$ are observed covariate vectors, $\beta_j \in \mathbb{R}^p$ and $\alpha_j \in \mathbb{R}^q$ are unknown regression parameters and, finally, $u_i \in \mathbb{R}^d$ and $v_j \in \mathbb{R}^d$ are latent vector explaining the residual varibility not captured by the regression effects. Equivalently, in matrix form, we have $g(\mu) = \eta = XB^\top + AZ^\top + UV^\top$.

The natural parameter θ_{ij} is linked to the conditional mean of y_{ij} through the equation $E(y_{ij}) = \mu_{ij} = b'(\theta_{ij})$. Similarly, the variance of y_{ij} is given by $\text{Var}(y_{ij}) = (\phi/w_{ij})\, \nu(\mu_{ij}) = (\phi/w_{ij})\, b''(\mu_{ij})$, where $\nu(\cdot)$ is the so-called variance function of the family. Finally, we denote by $D_\phi(y, \mu)$ the deviance function of the family, which is defined as $D_\phi(y, \mu) = -2\log\{f(y, \psi)/f_0(y)\}$, where $f_0(y)$ is the likelihood of the saturated model.

The estimation of the model parameters is performed by minimizing the penalized deviance function

$$\ell_\lambda(\psi; y) = -\sum_{i=1}^{n}\sum_{j=1}^{m} D_\phi(y_{ij}, \mu_{ij}) + \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2,$$

where $\lambda_U > 0$ and $\lambda_V > 0$ are regularization parameters and $\|\cdot\|_F$ is the Frobenious norm. Additional ℓ_2 penalization terms can be introduced to regularize B and A. Quasi-likelihood models can be considered as well, where $D_\phi(y, \mu)$ is substituted by $Q_\phi(y, \mu) = -\log(\phi/w) - (w/\phi)\int_y^\mu \{(y - t)/\nu(t)\}\, dt$, under an appropriate specification of mean, variance and link functions.

Identifiability constraints

The GMF model is not identifiable being invariant with respect to rotation, scaling and sign-flip transformations of U and V. To enforce the uniqueness of the solution, we impose the following identifiability constraints:

- $\text{Cov}(U) = U^\top (I_n - 1_n 1_n^\top / n) U / n = I_d$,
- V is lower triangular, with positive diagonal entries,

where I_n and 1_n are, respectively, the n-dimensional identity matrix and unitary vector.

Alternative identifiability constraints on U and V can be easily obtained by post processing. For instance, a PCA-like solution, say $U^\top U$ is diagonal and $V^\top V = I_d$, can by obtained by applying the truncated SVD decomposition $UV^\top = \tilde{U}\tilde{D}\tilde{V}^\top$, and setting $U = \tilde{U}\tilde{D}$ and $V = \tilde{V}$.

Estimation algorithms

To obtain the penalized maximum likelihood estimate, we here employs four different algorithms

- AIRWLS: alternated iterative re-weighted least squares (`method="airwls"`);
- Newton: quasi-Newton algorithm with diagonal Hessian (`method="newton"`);
- C-SGD: adaptive stochastic gradient descent with coordinate-wise sub-sampling (`method="sgd", sampling="coord"`);
- B-SGD: adaptive stochastic gradient descent with block-wise sub-sampling (`method="sgd", sampling="block"`);
- RB-SGD: as B-SGD but with an alternative rule to scan randomly the minibatch blocks (`method="sgd", sampling="rnd-block"`).

Likelihood families

Currently, all standard `glm` families are supported, including `neg.bin` and `negative.binomial` families from the `MASS` package. In such a case, the deviance function we consider takes the form $D_\phi(y, \mu) = 2w\big[y \log(y/\mu) - (y + \phi) \log\{(y + \phi)/(\mu + \phi)\}\big]$. This corresponds to a Negative Binomial model with variance function $\nu(\mu) = \mu + \mu^2/\phi$. Then, for $\phi \to \infty$, the Negative Binomial likelihood converges to a Poisson likelihood, having linear variance function, say $\nu(\mu) = \mu$. Notice that the over-dispersion parameter, that here is denoted as ϕ, in the `MASS` package is referred to as θ. If the Negative Binomial family is selected, a global over-dispersion parameter ϕ is estimated from the data using the method of moments.

Parallelization

Parallel execution is implemented in `C++` using `OpenMP`. When installing and compiling the `sgdGMF` package, the compiler check whether `OpenMP` is installed in the system. If it is not, the package is compiled excluding all the `OpenMP` functionalities and no parallel execution is allowed at `C++` level.

Notice that `OpenMP` is not compatible with `R` parallel computing packages, such as `parallel` and `foreach`. Therefore, when `parallel=TRUE`, it is not possible to run the `sgdgmf.fit` function within `R` level parallel functions, e.g., `foreach` loop.

Value

An `sgdgmf` object, namely a list, containing the estimated parameters of the GMF model. In particular, the returned object collects the following information:

- `method`: the selected estimation method
- `family`: the model family
- `ncomp`: rank of the latent matrix factorization
- `npar`: number of unknown parameters to be estimated

- `control.init`: list of control parameters used for the initialization
- `control.alg`: list of control parameters used for the optimization
- `control.cv`: list of control parameters used for the cross.validation
- `Y`: response matrix
- `X`: row-specific covariate matrix
- `Z`: column-specific covariate matrix
- `B`: the estimated col-specific coefficient matrix
- `A`: the estimated row-specific coefficient matrix
- `U`: the estimated factor matrix
- `V`: the estimated loading matrix
- `weights`: weighting matrix
- `offset`: offset matrix
- `eta`: the estimated linear predictor
- `mu`: the estimated mean matrix
- `var`: the estimated variance matrix
- `phi`: the estimated dispersion parameter
- `penalty`: the penalty value at the end of the optimization
- `deviance`: the deviance value at the end of the optimization
- `objective`: the penalized objective function at the end of the optimization
- `aic`: Akaike information criterion
- `bic`: Bayesian information criterion
- `names`: list of row and column names for all the output matrices
- `exe.time`: the total execution time in seconds
- `trace`: a trace matrix recording the optimization history
- `summary.cv`:

References

Kidzinnski, L., Hui, F.K.C., Warton, D.I. and Hastie, J.H. (2022). *Generalized Matrix Factorization: efficient algorithms for fitting generalized linear latent variable models to large data arrays.* Journal of Machine Learning Research, 23: 1-29.

Wang, L. and Carvalho, L. (2023). *Deviance matrix factorization.* Electronic Journal of Statistics, 17(2): 3762-3810.

Castiglione, C., Segers, A., Clement, L, Risso, D. (2024). *Stochastic gradient descent estimation of generalized matrix factorization models with application to single-cell RNA sequencing data.* arXiv preprint: arXiv:2412.20509.

See Also

`set.control.init`, `set.control.alg`, `sgdgmf.init`, `sgdgmf.rank`, `refit.sgdgmf`, `coef.sgdgmf`, `resid.sgdgmf`, `fitted.sgdgmf`, `predict.sgdgmf`, `plot.sgdgmf`, `screeplot.sgdgmf`, `biplot.sgdgmf`, `image.sgdgmf`

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Set the data dimensions
n = 100; m = 20; d = 5

# Generate data using Poisson, Binomial and Gamma models
data = sim.gmf.data(n = n, m = m, ncomp = d, family = poisson())

# Estimate the GMF parameters assuming 3 latent factors
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson(), method = "airwls")

# Get the fitted values in the link and response scales
mu_hat = fitted(gmf, type = "response")

# Compare the results
oldpar = par(no.readonly = TRUE)
par(mfrow = c(1,3), mar = c(1,1,3,1))
image(data$Y, axes = FALSE, main = expression(Y))
image(data$mu, axes = FALSE, main = expression(mu))
image(mu_hat, axes = FALSE, main = expression(hat(mu)))
par(oldpar)
```

---

sgdgmf.rank *Rank selection via eigenvalue-gap methods*

Description

Select the number of significant principal components of a GMF model via exploitation of eigenvalue-gap methods

Usage

```
sgdgmf.rank(
  Y,
  X = NULL,
  Z = NULL,
  maxcomp = ncol(Y),
  family = gaussian(),
  weights = NULL,
  offset = NULL,
  method = c("evr", "onatski", "act", "oht"),
  type.reg = c("ols", "glm"),
  type.res = c("deviance", "pearson", "working", "link"),
  normalize = FALSE,
  maxiter = 10,
  parallel = FALSE,
  nthreads = 1,
  return.eta = FALSE,
  return.mu = FALSE,
  return.res = FALSE,
  return.cov = FALSE
)
```

Arguments

`Y`	matrix of responses ($n \times m$)
`X`	matrix of row-specific fixed effects ($n \times p$)
`Z`	matrix of column-specific fixed effects ($q \times m$)
`maxcomp`	maximum number of eigenvalues to compute
`family`	a family as in the `glm` interface (default `gaussian()`)
`weights`	matrix of optional weights ($n \times m$)
`offset`	matrix of optional offsets ($n \times m$)
`method`	rank selection method
`type.reg`	regression method to be used to profile out the covariate effects
`type.res`	residual type to be decomposed
`normalize`	if `TRUE`, standardize column-by-column the residual matrix
`maxiter`	maximum number of iterations
`parallel`	if `TRUE`, allows for parallel computing using `foreach`
`nthreads`	number of cores to be used in parallel (only if `parallel=TRUE`)
`return.eta`	if `TRUE`, return the linear predictor martix
`return.mu`	if `TRUE`, return the fitted value martix
`return.res`	if `TRUE`, return the residual matrix
`return.cov`	if `TRUE`, return the covariance matrix of the residuals

Value

A list containing the `method`, the selected latent rank `ncomp`, and the eigenvalues used to select the latent rank `lambdas`. Additionally, if required, in the output list will also provide the linear predictor `eta`, the predicted mean matrix `mu`, the residual matrix `res`, and the implied residual covariance matrix `covmat`.

References

Onatski, A. (2010). *Determining the number of factors from empirical distribution of eigenvalues.* Review of Economics and Statistics, 92(4): 1004-1016

Ahn, S.C., Horenstein, A.R. (2013). *Eigenvalue ratio test for the number of factors.* Econometrica, 81, 1203-1227

Gavish, M., Donoho, D.L. (2014) *The optimal hard thresholding for singular values is 4/sqrt(3).* IEEE Transactions on Information Theory, 60(8): 5040–5053

Fan, J., Guo, J. and Zheng, S. (2020). *Estimating number of factors by adjusted eigenvalues thresholding.* Journal of the American Statistical Association, 117(538): 852–861

Wang, L. and Carvalho, L. (2023). *Deviance matrix factorization.* Electronic Journal of Statistics, 17(2): 3762-3810

Examples

```
library(sgdGMF)

# Set the data dimensions
n = 100; m = 20; d = 5

# Generate data using Poisson, Binomial and Gamma models
data_pois = sim.gmf.data(n = n, m = m, ncomp = d, family = poisson())
data_bin = sim.gmf.data(n = n, m = m, ncomp = d, family = binomial())
data_gam = sim.gmf.data(n = n, m = m, ncomp = d, family = Gamma(link = "log"), dispersion = 0.25)

# Initialize the GMF parameters assuming 3 latent factors
ncomp_pois = sgdgmf.rank(data_pois$Y, family = poisson(), normalize = TRUE)
ncomp_bin = sgdgmf.rank(data_bin$Y, family = binomial(), normalize = TRUE)
ncomp_gam = sgdgmf.rank(data_gam$Y, family = Gamma(link = "log"), normalize = TRUE)

# Get the selected number of components
print(paste("Poisson:", ncomp_pois$ncomp))
print(paste("Binomial:", ncomp_bin$ncomp))
print(paste("Gamma:", ncomp_gam$ncomp))

# Plot the screeplot used for the component determination
oldpar = par(no.readonly = TRUE)
par(mfrow = c(3,1))
barplot(ncomp_pois$lambdas, main = "Poisson screeplot")
barplot(ncomp_bin$lambdas, main = "Binomial screeplot")
barplot(ncomp_gam$lambdas, main = "Gamma screeplot")
par(oldpar)
```

sim.gmf.data *Simulate non-Gaussian data from a GMF model*

Description

Simulate synthetic non-Gaussian data from a generalized matrix factorization (GMF) model.

Usage

```
sim.gmf.data(n = 100, m = 20, ncomp = 5, family = gaussian(), dispersion = 1)
```

Arguments

n	number of observations
m	number of variables
ncomp	rank of the latent matrix factorization
family	a glm family (see family for more details)
dispersion	a positive dispersion parameter

Details

The loadings, V, are independently sampled from a standard normal distribution. The scores, U, are simulated according to sinusoidal signals evaluated at different phases, frequencies and amplitudes. These parameters are randomly sampled from independent uniform distributions.

Value

A list containing the following objects:

- Y: simulated response matrix
- U: simulated factor matrix
- V: simulated loading matrix
- eta: linear predictor matrix
- mu: conditional mean matrix
- phi: scalar dispersion parameter
- family: model family
- ncomp: rank of the latent matrix factorization
- param: a list containing time, phase, frequency and amplitude vectors used to generate U

Examples

```
library(sgdGMF)

# Set the data dimensions
n = 100; m = 20; d = 5

# Generate data using Poisson, Binomial and Gamma models
data_pois = sim.gmf.data(n = n, m = m, ncomp = d, family = poisson())
data_bin = sim.gmf.data(n = n, m = m, ncomp = d, family = binomial())
data_gam = sim.gmf.data(n = n, m = m, ncomp = d, family = Gamma(link = "log"), dispersion = 0.25)

# Compare the results
oldpar = par(no.readonly = TRUE)
par(mfrow = c(3,3), mar = c(1,1,3,1))
image(data_pois$Y, axes = FALSE, main = expression(Y[Pois]))
image(data_pois$mu, axes = FALSE, main = expression(mu[Pois]))
image(data_pois$U, axes = FALSE, main = expression(U[Pois]))
image(data_bin$Y, axes = FALSE, main = expression(Y[Bin]))
image(data_bin$mu, axes = FALSE, main = expression(mu[Bin]))
image(data_bin$U, axes = FALSE, main = expression(U[Bin]))
image(data_gam$Y, axes = FALSE, main = expression(Y[Gam]))
image(data_gam$mu, axes = FALSE, main = expression(mu[Gam]))
image(data_gam$U, axes = FALSE, main = expression(U[Gam]))
par(oldpar)
```

simulate *Simulate new data*

Description

Generic function to simulate new data from a statistical model

Usage

```
simulate(object, ...)
```

Arguments

object	an object from which simulate new data
...	additional arguments passed to or from other methods

Value

An array containing the simulated data.

simulate.sgdgmf *Simulate method for GMF models*

Description

Simulate new data from a fitted generalized matrix factorization models

Usage

```
## S3 method for class 'sgdgmf'
simulate(object, ..., nsim = 1)
```

Arguments

object	an object of class sgdgmf
...	further arguments passed to or from other methods
nsim	number of samples

Value

An 3-fold array containing the simulated data.

Examples

```
# Load the sgdGMF package
library(sgdGMF)

# Generate data from a Poisson model
data = sim.gmf.data(n = 100, m = 20, ncomp = 5, family = poisson())

# Fit a GMF model
gmf = sgdgmf.fit(data$Y, ncomp = 3, family = poisson())

# Simulate new data from a GMF model
str(simulate(gmf))
```

Index