Package 'MOFA2'

December 20, 2024

Type Package

Title Multi-Omics Factor Analysis v2

Version 1.17.0

Maintainer Ricard Argelaguet <ricard.argelaguet@gmail.com>

Date 2023-01-12

License file LICENSE

Description The MOFA2 package contains a collection of tools for training and analysing multi-omic factor analysis (MOFA). MOFA is a probabilistic factor model that aims to identify principal axes of variation from data sets that can comprise multiple omic layers and/or groups of samples. Additional time or space information on the samples can be incorporated using the MEFISTO framework, which is part of MOFA2. Downstream analysis functions to inspect molecular features underlying each factor, vizualisation, imputation etc are available.

Encoding UTF-8

Depends R (>= 4.0)

Imports rhdf5, dplyr, tidyr, reshape2, pheatmap, ggplot2, methods, RColorBrewer, cowplot, ggrepel, reticulate, HDF5Array, grDevices, stats, magrittr, forcats, utils, corrplot, DelayedArray, Rtsne, uwot, basilisk, stringi

Suggests knitr, testthat, Seurat, SeuratObject, ggpubr, foreach, psych, MultiAssayExperiment, SummarizedExperiment, SingleCellExperiment, ggrastr, mvtnorm, GGally, rmarkdown, data.table, tidyverse, BiocStyle, Matrix, markdown

biocViews DimensionReduction, Bayesian, Visualization

URL https://biofam.github.io/MOFA2/index.html

BugReports https://github.com/bioFAM/MOFA2

VignetteBuilder knitr

LazyData false

StagedInstall no

NeedsCompilation yes

RoxygenNote 7.2.3

SystemRequirements Python (>=3), numpy, pandas, h5py, scipy, argparse, sklearn, mofapy2

git_url https://git.bioconductor.org/packages/MOFA2

git_branch devel

git_last_commit 12bde39

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2024-12-20

Author Ricard Argelaguet [aut, cre] (ORCID: <https://orcid.org/0000-0003-3199-3722>), Damien Arnol [aut] (ORCID: <https://orcid.org/0000-0003-2462-534X>), Danila Bredikhin [aut] (ORCID: <https://orcid.org/0000-0001-8089-6983>), Britta Velten [aut] (ORCID: <https://orcid.org/0000-0002-8397-3515>)

Contents

`add_mofa_factors_to_seurat` *Function to add the MOFA representation onto a Seurat object*

Description

Function to add the MOFA latent representation to a Seurat object

Usage

```
add_mofa_factors_to_seurat(
  mofa_object,
  seurat_object,
  views = "all",
  factors = "all"
)
```

Arguments

`mofa_object`	a trained `MOFA` object.
`seurat_object`	a Seurat object
`views`	character vector with the view names, or numeric vector with view indexes. Default is 'all'
`factors`	character vector with the factor names, or numeric vector with the factor indexes. Default is 'all'

Details

This function calls the `CreateDimReducObject` function from Seurat to store the MOFA factors.

Value

Returns a Seurat object with the 'reductions' slot filled with the MOFA factors. Also adds, if calculated, the UMAP/TSNE obtained with the MOFA factors.

Examples

```
# Generate a simulated data set
MOFAexample <- make_example_data()
```

`calculate_contribution_scores` *Calculate contribution scores for each view in each sample*

Description

This function calculates, *for each sample* how much each view contributes to its location in the latent manifold, what we call *contribution scores*

Usage

```
calculate_contribution_scores(
  object,
  views = "all",
  groups = "all",
  factors = "all",
  scale = TRUE
)
```

Arguments

`object`	a trained `MOFA` object.
`views`	character vector with the view names, or numeric vector with view indexes. Default is 'all'
`groups`	character vector with the group names, or numeric vector with group indexes. Default is 'all'
`factors`	character vector with the factor names, or numeric vector with the factor indexes. Default is 'all'
`scale`	logical indicating whether to scale the sample-wise variance explained values by the total amount of variance explained per view. This effectively normalises each view by its total variance explained. It is important when different amounts of variance is explained for each view (check with `plot_variance_explained(..., plot_total=TRUE)`)

Details

Contribution scores are calculated in three steps:

- Step 1calculate variance explained for each cell i and each view m (R_{im}), using all factors
- Step 2(optional) scale values by the total variance explained for each view
- Step 3calculate contribution score (C_{im}) for cell i and view m as:

$$C_{im} = \frac{R2_{im}}{\sum_m R2_{im}}$$

Note that contribution scores can be calculated using any number of data modalities, but it is easier to interpret when you specify two.
Please note that this functionality is still experimental, contact the authors if you have questions.

Value

adds the contribution scores to the metadata slot (`samples_metadata(MOFAobject)`) and to the `MOFAobject@cache` slot

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
model <- calculate_contribution_scores(model)
```

`calculate_variance_explained`

Calculate variance explained by the model

Description

This function takes a trained MOFA model as input and calculates the proportion of variance explained (i.e. the coefficient of determinations (R^2)) by the MOFA factors across the different views.

Usage

```
calculate_variance_explained(
  object,
  views = "all",
  groups = "all",
  factors = "all"
)
```

Arguments

`object`	a `MOFA` object.
`views`	character vector with the view names, or numeric vector with view indexes. Default is 'all'
`groups`	character vector with the group names, or numeric vector with group indexes. Default is 'all'
`factors`	character vector with the factor names, or numeric vector with the factor indexes. Default is 'all'

Value

a list with matrices with the amount of variation explained per factor and view.

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Calculate variance explained (R2)
r2 <- calculate_variance_explained(model)

# Plot variance explained values (view as x-axis, and factor as y-axis)
plot_variance_explained(model, x="view", y="factor")

# Plot variance explained values (view as x-axis, and group as y-axis)
plot_variance_explained(model, x="view", y="group")

# Plot variance explained values for factors 1 to 3
plot_variance_explained(model, x="view", y="group", factors=1:3)

# Scale R2 values
plot_variance_explained(model, max_r2 = 0.25)
```

`calculate_variance_explained_per_sample` *Calculate variance explained by the MOFA factors for each sample*

Description

This function takes a trained MOFA model as input and calculates, **for each sample** the proportion of variance explained (i.e. the coefficient of determinations (R^2)) by the MOFA factors across the different views.

Usage

```
calculate_variance_explained_per_sample(
  object,
  views = "all",
  groups = "all",
  factors = "all"
)
```

Arguments

`object`	a MOFA object.
`views`	character vector with the view names, or numeric vector with view indexes. Default is 'all'
`groups`	character vector with the group names, or numeric vector with group indexes. Default is 'all'
`factors`	character vector with the factor names, or numeric vector with the factor indexes. Default is 'all'

Value

a list with matrices with the amount of variation explained per sample and view.

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Calculate variance explained (R2)
r2 <- calculate_variance_explained_per_sample(model)
```

`cluster_samples` *K-means clustering on samples based on latent factors*

Description

MOFA factors are continuous in nature but they can be used to predict discrete clusters of samples. The clustering can be performed in a single factor, which is equivalent to setting a manual threshold. More interestingly, it can be done using multiple factors, where multiple sources of variation are aggregated.
Importantly, this type of clustering is not weighted and does not take into account the different importance of the latent factors.

Usage

```
cluster_samples(object, k, factors = "all", ...)
```

Arguments

`object`	a trained `MOFA` object.
`k`	number of clusters (integer).
`factors`	character vector with the factor name(s), or numeric vector with the index of the factor(s) to use. Default is 'all'
`...`	extra arguments passed to `kmeans`

Details

In some cases, due to model technicalities, samples can have missing values in the latent factor space. In such a case, these samples are currently ignored in the clustering procedure.

Value

output from `kmeans` function

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Cluster samples in the factor space using factors 1 to 3 and K=2 clusters
clusters <- cluster_samples(model, k=2, factors=1:3)
```

`compare_elbo` *Compare different trained MOFA objects in terms of the final value of the ELBO statistics and number of inferred factors*

Description

Different objects of MOFA are compared in terms of the final value of the ELBO statistics. For model selection the model with the highest ELBO value is selected.

Usage

```
compare_elbo(models, log = FALSE, return_data = FALSE)
```

Arguments

`models`	a list containing MOFA objects.
`log`	logical indicating whether to plot the log of the ELBO.
`return_data`	logical indicating whether to return a data.frame with the ELBO values per model

Value

A ggplot object or the underlying data.frame if return_data is TRUE

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model1 <- load_model(file)
model2 <- load_model(file)

# Compare ELBO between models
## Not run: compare_elbo(list(model1,model2))
```

compare_factors *Plot the correlation of factors between different models*

Description

Different MOFA objects are compared in terms of correlation between their factors.

Usage

```
compare_factors(models, ...)
```

Arguments

models	a list with MOFA objects.
...	extra arguments passed to pheatmap

Details

If assessing model robustness across trials, the output should look like a block diagonal matrix, suggesting that all factors are robustly detected in all model instances.

Value

Plots a heatmap of the Pearson correlation between latent factors across all input models.

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model1 <- load_model(file)
model2 <- load_model(file)

# Compare factors between models
compare_factors(list(model1,model2))
```

correlate_factors_with_covariates *Plot correlation of factors with external covariates*

Description

Function to correlate factor values with external covariates.

Usage

```
correlate_factors_with_covariates(
  object,
  covariates,
  factors = "all",
  groups = "all",
  abs = FALSE,
  plot = c("log_pval", "r"),
  alpha = 0.05,
  return_data = FALSE,
  transpose = FALSE,
  ...
)
```

Arguments

object	a trained MOFA object.
covariates	• **data.frame**:a data.frame where the samples are stored in the rows and the covariates are stored in the columns. Use row names for sample names and column names for covariate names. Columns values must be numeric. • **character vector**:character vector with names of columns that are present in the sample metadata (samples_metadata(model)
factors	character vector with the factor name(s), or numeric vector with the index of the factor(s) to use. Default is 'all'.
groups	character vector with the groups names, or numeric vector with the indices of the groups of samples to use, or "all" to use samples from all groups.
abs	logical indicating whether to take the absolute value of the correlation coefficient (default is TRUE).
plot	character indicating whether to plot Pearson correlation coefficiens (plot="r") or log10 adjusted p-values (plot="log_pval").
alpha	p-value threshold
return_data	logical indicating whether to return the correlation results instead of plotting
transpose	logical indicating whether to transpose the plot
...	extra arguments passed to corrplot (if plot=="r") or pheatmap (if plot=="log_pval").

Value

A corrplot (if plot=="r") or pheatmap (if plot=="log_pval") or the underlying data.frame if return_data is TRUE

covariates_names *covariates_names: set and retrieve covariate names*

Description

covariates_names: set and retrieve covariate names

Usage

```
covariates_names(object)

covariates_names(object) <- value

## S4 method for signature 'MOFA'
covariates_names(object)

## S4 replacement method for signature 'MOFA,vector'
covariates_names(object) <- value
```

Arguments

object	a MOFA object.
value	a character vector of covariate names

Value

character vector with the covariate names

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
covariates_names(model)
```

create_mofa *create a MOFA object*

Description

Method to create a MOFA object. Depending on the input data format, this method calls one of the following functions:

- **long data.frame**: create_mofa_from_df
- **List of matrices**: create_mofa_from_matrix

- **MultiAssayExperiment**: create_mofa_from_MultiAssayExperiment
- **Seurat**: create_mofa_from_Seurat
- **SingleCellExperiment**: create_mofa_from_SingleCellExperiment

Please read the documentation of the corresponding function for more details on your specific data format.

Usage

```
create_mofa(data, groups = NULL, extract_metadata = TRUE, ...)
```

Arguments

data	one of the formats above
groups	group information, only relevant when using the multi-group framework.
extract_metadata	logical indicating whether to incorporate the sample metadata from the input object into the MOFA object (not relevant when the input is a list of matrices). Default is TRUE.
...	further arguments that can be passed to the function depending on the inout data format. See the dpcumentation of above functions for details.

Value

Returns an untrained MOFA object

Examples

```
# Using an existing simulated data with two groups and two views
file <- system.file("extdata", "test_data.RData", package = "MOFA2")

# Load data (in long data.frame format)
load(file)
MOFAmodel <- create_mofa(dt)
```

---

create_mofa_from_df *create a MOFA object from a data.frame object*

Description

Method to create a MOFA object from a data.frame object

Usage

```
create_mofa_from_df(df, extract_metadata = TRUE)
```

Arguments

df	data.frame object with at most 5 columns: sample, group, feature, view, value. The group column (optional) indicates the group of each sample when using the multi-group framework. The view column (optional) indicates the view of each feature when having multi-view data.
extract_metadata	logical indicating whether to incorporate the extra columns as sample metadata into the MOFA object

Value

Returns an untrained MOFA object

Examples

```
# Using an existing simulated data with two groups and two views
file <- system.file("extdata", "test_data.RData", package = "MOFA2")

# Load data (in long data.frame format)
load(file)
MOFAmodel <- create_mofa_from_df(dt)
```

---

create_mofa_from_matrix

create a MOFA object from a a list of matrices

Description

Method to create a MOFA object from a list of matrices

Usage

```
create_mofa_from_matrix(data, groups = NULL)
```

Arguments

data	A list of matrices, where each entry corresponds to one view. Samples are stored in columns and features in rows. Missing values must be filled in prior to creating the MOFA object (see for example the CLL tutorial)
groups	A character vector with group assignment for every sample. Default is NULL, no group structure.

Value

Returns an untrained MOFA object

Examples

```
m <- make_example_data()
create_mofa_from_matrix(m$data)
```

`create_mofa_from_MultiAssayExperiment`
create a MOFA object from a MultiAssayExperiment object

Description

Method to create a MOFA object from a MultiAssayExperiment object

Usage

```
create_mofa_from_MultiAssayExperiment(
  mae,
  groups = NULL,
  extract_metadata = FALSE
)
```

Arguments

`mae`	a MultiAssayExperiment object
`groups`	a string specifying column name of the colData to use it as a group variable. Alternatively, a character vector with group assignment for every sample. Default is `NULL` (no group structure).
`extract_metadata`	logical indicating whether to incorporate the metadata from the MultiAssayExperiment object into the MOFA object

Value

Returns an untrained MOFA object

`create_mofa_from_Seurat`
create a MOFA object from a Seurat object

Description

Method to create a MOFA object from a Seurat object

Usage

```
create_mofa_from_Seurat(
  seurat,
  groups = NULL,
  assays = NULL,
  slot = "scale.data",
  features = NULL,
  extract_metadata = FALSE
)
```

Arguments

seurat	Seurat object
groups	a string specifying column name of the samples metadata to use it as a group variable. Alternatively, a character vector with group assignment for every sample. Default is NULL (no group structure).
assays	assays to use, default is NULL, it fetched all assays available
slot	assay slot to be used (default is scale.data).
features	a list with vectors, which are used to subset features, with names corresponding to assays; a vector can be provided when only one assay is used
extract_metadata	logical indicating whether to incorporate the metadata from the Seurat object into the MOFA object

Value

Returns an untrained MOFA object

---

create_mofa_from_SingleCellExperiment

create a MOFA object from a SingleCellExperiment object

Description

Method to create a MOFA object from a SingleCellExperiment object

Usage

```
create_mofa_from_SingleCellExperiment(
  sce,
  groups = NULL,
  assay = "logcounts",
  extract_metadata = FALSE
)
```

Arguments

sce	SingleCellExperiment object
groups	a string specifying column name of the colData to use it as a group variable. Alternatively, a character vector with group assignment for every sample. Default is NULL (no group structure).
assay	assay to use, default is logcounts.
extract_metadata	logical indicating whether to incorporate the metadata from the SingleCellExperiment object into the MOFA object

Value

Returns an untrained MOFA object

---

factors_names *factors_names: set and retrieve factor names*

Description

factors_names: set and retrieve factor names

Usage

```
factors_names(object)

factors_names(object) <- value

## S4 method for signature 'MOFA'
factors_names(object)

## S4 replacement method for signature 'MOFA,vector'
factors_names(object) <- value
```

Arguments

object	a MOFA object.
value	a character vector of factor names

Value

character vector with the factor names

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
factors_names(model)
```

features_metadata *features_metadata: set and retrieve feature metadata*

Description

features_metadata: set and retrieve feature metadata

Usage

```
features_metadata(object)

features_metadata(object) <- value

## S4 method for signature 'MOFA'
features_metadata(object)

## S4 replacement method for signature 'MOFA,data.frame'
features_metadata(object) <- value
```

Arguments

object	a MOFA object.
value	data frame with feature information, it at least must contain the columns feature and view

Value

a data frame with sample metadata

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
features_metadata(model)
```

features_names *features_names: set and retrieve feature names*

Description

features_names: set and retrieve feature names

Usage

```
features_names(object)

features_names(object) <- value

## S4 method for signature 'MOFA'
features_names(object)

## S4 replacement method for signature 'MOFA,list'
features_names(object) <- value
```

Arguments

object	a MOFA object.
value	list of character vectors with the feature names for every view

Value

list of character vectors with the feature names for each view

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
features_names(model)
```

---

get_covariates *Get sample covariates*

Description

Function to extract the covariates from a MOFA object using MEFISTO.

Usage

```
get_covariates(
  object,
  covariates = "all",
  as.data.frame = FALSE,
  warped = FALSE
)
```

Arguments

object	a MOFA object.
covariates	character vector with the covariate name(s), or numeric vector with the covariate index(es).
as.data.frame	logical indicating whether to output the result as a long data frame, default is FALSE.
warped	logical indicating whether to extract the aligned covariates

Value

a matrix with dimensions (samples,covariates). If as.data.frame is TRUE, a long-formatted data frame with columns (sample,factor,value)

Examples

```
# Using an existing trained model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
covariates <- get_covariates(model)
```

---

get_data *Get data*

Description

Fetch the input data

Usage

```
get_data(
  object,
  views = "all",
  groups = "all",
  features = "all",
  as.data.frame = FALSE,
  add_intercept = TRUE,
  denoise = FALSE,
  na.rm = TRUE
)
```

Arguments

object	a MOFA object.
views	character vector with the view name(s), or numeric vector with the view index(es). Default is "all".

groups	character vector with the group name(s), or numeric vector with the group index(es). Default is "all".
features	a *named* list of character vectors. Example: list("view1"=c("feature_1","feature_2"), "view2"=c("feature_3","feature_4")) Default is "all".
as.data.frame	logical indicating whether to return a long data frame instead of a list of matrices. Default is FALSE.
add_intercept	logical indicating whether to add feature intercepts to the data. Default is TRUE.
denoise	logical indicating whether to return the denoised data (i.e. the model predictions). Default is FALSE.
na.rm	remove NAs from the data.frame (only if as.data.frame is TRUE).

Details

By default this function returns a list where each element is a data matrix with dimensionality (D,N) where D is the number of features and N is the number of samples.
Alternatively, if as.data.frame is TRUE, the function returns a long-formatted data frame with columns (view,feature,sample,value). Missing values are not included in the the long data.frame format by default. To include them use the argument na.rm=FALSE.

Value

A list of data matrices with dimensionality (D,N) or a data.frame (if as.data.frame is TRUE)

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Fetch data
data <- get_data(model)

# Fetch a specific view
data <- get_data(model, views = "view_0")

# Fetch data in data.frame format instead of matrix format
data <- get_data(model, as.data.frame = TRUE)

# Fetch centered data (do not add the feature intercepts)
data <- get_data(model, as.data.frame = FALSE)

# Fetch denoised data (do not add the feature intercepts)
data <- get_data(model, denoise = TRUE)
```

get_default_data_options *Get default data options*

Description

Function to obtain the default data options.

Usage

```
get_default_data_options(object)
```

Arguments

object	an untrained MOFA object

Details

This function provides a default set of data options that can be modified and passed to the MOFA object in the prepare_mofa step (see example), i.e. after creating a MOFA object (using create_mofa) and before starting the training (using run_mofa) The data options are the following:

- **scale_views**: logical indicating whether to scale views to have the same unit variance. As long as the scale differences between the views is not too high, this is not required. Default is FALSE.
- **scale_groups**: logical indicating whether to scale groups to have the same unit variance. As long as the scale differences between the groups is not too high, this is not required. Default is FALSE.
- **use_float32**: logical indicating whether use float32 instead of float64 arrays to increase speed and memory usage. Default is FALSE.

Value

Returns a list with the default data options.

Examples

```
# Using an existing simulated data with two groups and two views
file <- system.file("extdata", "test_data.RData", package = "MOFA2")

# Load data dt (in data.frame format)
load(file)

# Create the MOFA object
MOFAmodel <- create_mofa(dt)

# Load default data options
```

```
data_opts <- get_default_data_options(MOFAmodel)

# Edit some of the data options
data_opts$scale_views <- TRUE

# Prepare the MOFA object
MOFAmodel <- prepare_mofa(MOFAmodel, data_options = data_opts)
```

`get_default_mefisto_options` *Get default options for MEFISTO covariates*

Description

Function to obtain the default options for the usage of MEFISTO covariates with MEFISTO

Usage

```
get_default_mefisto_options(object)
```

Arguments

`object`	an untrained `MOFA` object

Details

The options are the following:

- **scale_cov**: logical: Scale covariates?
- **start_opt**: integer: First iteration to start the optimisation of GP hyperparameters
- **n_grid**: integer: Number of points for the grid search in the optimisation of GP hyperparameters
- **opt_freq**: integer: Frequency of optimisation of GP hyperparameters
- **sparseGP**: logical: Use sparse GPs to speed up the optimisation of the GP parameters?
- **frac_inducing**: numeric between 0 and 1: Fraction of samples to use as inducing points (only relevant if sparseGP is `TRUE`)
- **warping**: logical: Activate warping functionality to align covariates between groups (requires a multi-group design)
- **warping_freq**: numeric: frequency of the warping (only relevant if warping is `TRUE`)
- **warping_ref**: A character specifying the reference group for warping (only relevant if warping is `TRUE`)
- **warping_open_begin**: logical: Warping: Allow for open beginning? (only relevant warping is `TRUE`)
- **warping_open_end**: logical: Warping: Allow for open end? (only relevant warping is `TRUE`)

- **warping_groups**: Assignment of groups to classes used for alignment (advanced option). Needs to be a vector of length number of samples, e.g. a column of samples_metadata, which needs to have the same value within each group. By default groups are used specified in 'create_mofa'.
- **model_groups**: logical: Model covariance structure across groups (for more than one group, otherwise FALSE)? If FALSE, we assume the same patterns in all groups.
- **new_values**: Values for which to predict the factor values (for interpolation / extrapolation). This should be numeric matrix in the same format with covariate(s) in rows and new values in columns. Default is NULL, leading to no interpolation.

Value

Returns a list with default options for the MEFISTO covariate(s) functionality.

Examples

```
# generate example data
dd <- make_example_data(sample_cov = seq(0,1,length.out = 200), n_samples = 200,
n_factors = 4, n_features = 200, n_views = 4, lscales = c(0.5, 0.2, 0, 0))
# input data
data <- dd$data
# covariate matrix with samples in columns
time <- dd$sample_cov
rownames(time) <- "time"

# create mofa and set covariates
sm <- create_mofa(data = dd$data)
sm <- set_covariates(sm, covariates = time)

MEFISTO_opt <- get_default_mefisto_options(sm)
```

---

get_default_model_options

Get default model options

Description

Function to obtain the default model options.

Usage

```
get_default_model_options(object)
```

Arguments

object	an untrained MOFA object

Details

This function provides a default set of model options that can be modified and passed to the MOFA object in the prepare_mofa step (see example), i.e. after creating a MOFA object (using create_mofa) and before starting the training (using run_mofa) The model options are the following:

- **likelihoods**: character vector with data likelihoods per view: 'gaussian' for continuous data (Default for all views), 'bernoulli' for binary data and 'poisson' for count data.
- **num_factors**: numeric value indicating the (initial) number of factors. Default is 15.
- **spikeslab_factors**: logical indicating whether to use spike and slab sparsity on the factors (Default is FALSE)
- **spikeslab_weights**: logical indicating whether to use spike and slab sparsity on the weights (Default is TRUE)
- **ard_factors**: logical indicating whether to use ARD sparsity on the factors (Default is TRUE only if using multiple groups)
- **ard_weights**: logical indicating whether to use ARD sparsity on the weights (Default is TRUE)

Value

Returns a list with the default model options.

Examples

```
# Using an existing simulated data with two groups and two views
file <- system.file("extdata", "test_data.RData", package = "MOFA2")

# Load data dt (in data.frame format)
load(file)

# Create the MOFA object
MOFAmodel <- create_mofa(dt)

# Load default model options
model_opts <- get_default_model_options(MOFAmodel)

# Edit some of the model options
model_opts$num_factors <- 10
model_opts$spikeslab_weights <- FALSE

# Prepare the MOFA object
MOFAmodel <- prepare_mofa(MOFAmodel, model_options = model_opts)
```

get_default_stochastic_options *Get default stochastic options*

Description

Function to obtain the default options for stochastic variational inference.

Usage

```
get_default_stochastic_options(object)
```

Arguments

object	an untrained MOFA

Details

This function provides a default set of stochastic inference options that can be modified and passed to the MOFA object in the prepare_mofa step), i.e. after creating a MOFA object (using create_mofa) and before starting the training (using run_mofa) These options are only relevant when activating stochastic inference in training_options (see example). The stochastic inference options are the following:

- **batch_size**: numeric value indicating the batch size (as a fraction). Default is 0.5 (half of the data set).
- **learning_rate**: numeric value indicating the learning rate. Default is 1.0
- **forgetting_rate**: numeric indicating the forgetting rate. Default is 0.5
- **start_stochastic**: integer indicating the first iteration to start stochastic inference Default is 1

Value

Returns a list with default options

Examples

```
# Using an existing simulated data with two groups and two views
file <- system.file("extdata", "test_data.RData", package = "MOFA2")

# Load data dt (in data.frame format)
load(file)

# Create the MOFA object
MOFAmodel <- create_mofa(dt)

# activate stochastic inference in training options
train_opts <- get_default_training_options(MOFAmodel)
```

```
train_opts$stochastic <- TRUE

# Load default stochastic options
stochastic_opts <- get_default_stochastic_options(MOFAmodel)

# Edit some of the stochastic options
stochastic_opts$learning_rate <- 0.75
stochastic_opts$batch_size <- 0.25

# Prepare the MOFA object
MOFAmodel <- prepare_mofa(MOFAmodel,
  training_options = train_opts,
  stochastic_options = stochastic_opts
)
```

`get_default_training_options` *Get default training options*

Description

Function to obtain the default training options.

Usage

```
get_default_training_options(object)
```

Arguments

`object`	an untrained `MOFA`

Details

This function provides a default set of training options that can be modified and passed to the `MOFA` object in the `prepare_mofa` step (see example), i.e. after creating a `MOFA` object (using `create_mofa`) and before starting the training (using `run_mofa`) The training options are the following:

- **maxiter**: numeric value indicating the maximum number of iterations. Default is 1000. Convergence is assessed using the ELBO statistic.
- **drop_factor_threshold**: numeric indicating the threshold on fraction of variance explained to consider a factor inactive and drop it from the model. For example, a value of 0.01 implies that factors explaining less than 1% of variance (in each view) will be dropped. Default is -1 (no dropping of factors)
- **convergence_mode**: character indicating the convergence criteria, either "fast", "medium" or "slow", corresponding to 0.0005%, 0.00005% or 0.000005% deltaELBO change.

- **verbose**: logical indicating whether to generate a verbose output.
- **startELBO**: integer indicating the first iteration to compute the ELBO (default is 1).
- **freqELBO**: integer indicating the first iteration to compute the ELBO (default is 1).
- **stochastic**: logical indicating whether to use stochastic variational inference (only required for very big data sets, default is FALSE).
- **gpu_mode**: logical indicating whether to use GPUs (see details).
- **gpu_device**: integer indicating which GPU to use.
- **seed**: numeric indicating the seed for reproducibility (default is 42).

Value

Returns a list with default training options

Examples

```
# Using an existing simulated data with two groups and two views
file <- system.file("extdata", "test_data.RData", package = "MOFA2")

# Load data dt (in data.frame format)
load(file)

# Create the MOFA object
MOFAmodel <- create_mofa(dt)

# Load default training options
train_opts <- get_default_training_options(MOFAmodel)

# Edit some of the training options
train_opts$convergence_mode <- "medium"
train_opts$startELBO <- 100
train_opts$seed <- 42

# Prepare the MOFA object
MOFAmodel <- prepare_mofa(MOFAmodel, training_options = train_opts)
```

---

get_dimensions *Get dimensions*

Description

Extract dimensionalities from the model.

Usage

```
get_dimensions(object)
```

Arguments

object a MOFA object.

Details

K indicates the number of factors, M indicates the number of views, D indicates the number of features (per view), N indicates the number of samples (per group) and C indicates the number of covariates.

Value

list containing the dimensionalities of the model

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
dims <- get_dimensions(model)
```

---

get_elbo *Get ELBO*

Description

Extract the value of the ELBO statistics after model training. This can be useful for model selection.

Usage

```
get_elbo(object)
```

Arguments

object a MOFA object.

Details

This can be useful for model selection.

Value

Value of the ELBO

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
elbo <- get_elbo(model)
```

`get_expectations` *Get expectations*

Description

Function to extract the expectations from the (variational) posterior distributions of a trained `MOFA` object.

Usage

```
get_expectations(object, variable, as.data.frame = FALSE)
```

Arguments

`object`	a trained `MOFA` object.
`variable`	variable name: 'Z' for factors and 'W' for weights.
`as.data.frame`	logical indicating whether to output the result as a long data frame, default is `FALSE`.

Details

Technical note: MOFA is a Bayesian model where each variable has a prior distribution and a posterior distribution. In particular, to achieve scalability we used the variational inference framework, thus true posterior distributions are replaced by approximated variational distributions. This function extracts the expectations of the variational distributions, which can be used as final point estimates to analyse the results of the model.
The priors and variational distributions of each variable are extensively described in the supplementary methods of the original paper.

Value

the output varies depending on the variable of interest:

- "Z"a matrix with dimensions (samples,factors). If `as.data.frame` is `TRUE`, a long-formatted data frame with columns (sample,factor,value)
- "W"a list of length (views) where each element is a matrix with dimensions (features,factors). If `as.data.frame` is `TRUE`, a long-formatted data frame with columns (view,feature,factor,value)

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
factors <- get_expectations(model, "Z")
weights <- get_expectations(model, "W")
```

get_factors *Get factors*

Description

Extract the latent factors from the model.

Usage

```
get_factors(
  object,
  groups = "all",
  factors = "all",
  scale = FALSE,
  as.data.frame = FALSE
)
```

Arguments

object	a trained MOFA object.
groups	character vector with the group name(s), or numeric vector with the group index(es). Default is "all".
factors	character vector with the factor name(s), or numeric vector with the factor index(es). Default is "all".
scale	logical indicating whether to scale factor values.
as.data.frame	logical indicating whether to return a long data frame instead of a matrix. Default is FALSE.

Value

By default it returns the latent factor matrix of dimensionality (N,K), where N is number of samples and K is number of factors.
Alternatively, if as.data.frame is TRUE, returns a long-formatted data frame with columns (sample,factor,value).

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Fetch factors in matrix format (a list, one matrix per group)
factors <- get_factors(model)

# Concatenate groups
factors <- do.call("rbind",factors)
```

```
# Fetch factors in data.frame format instead of matrix format
factors <- get_factors(model, as.data.frame = TRUE)
```

`get_group_kernel` *Get group covariance matrix*

Description

Extract the inferred group-group covariance matrix per factor

Usage

```
get_group_kernel(object)
```

Arguments

`object`	a `MOFA` object.

Details

This can be used only if covariates are passed to the MOFAobject upon creation and GP_factors is set to True.

Value

A list of group-group correlation matrices per factor

`get_imputed_data` *Get imputed data*

Description

Function to get the imputed data. It requires the previous use of the `impute` method.

Usage

```
get_imputed_data(
  object,
  views = "all",
  groups = "all",
  features = "all",
  as.data.frame = FALSE
)
```

Arguments

object	a trained MOFA object.
views	character vector with the view name(s), or numeric vector with the view index(es). Default is "all".
groups	character vector with the group name(s), or numeric vector with the group index(es). Default is "all".
features	list of character vectors with the feature names or list of numeric vectors with the feature indices. Default is "all".
as.data.frame	logical indicating whether to return a long-formatted data frame instead of a list of matrices. Default is FALSE.

Details

Data is imputed from the generative model of MOFA.

Value

A list containing the imputed valued or a data.frame if as.data.frame is TRUE

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
model <- impute(model)
imputed <- get_imputed_data(model)
```

---

get_interpolated_factors

Get interpolated factor values

Description

Extract the interpolated factor values

Usage

```
get_interpolated_factors(object, as.data.frame = FALSE, only_mean = FALSE)
```

Arguments

object	a MOFA object
as.data.frame	logical indicating whether to return data as a data.frame
only_mean	logical indicating whether include only mean or also uncertainties

Details

This can be used only if covariates are passed to the object upon creation, GP_factors is set to True and new covariates were passed for interpolation.

Value

By default, a nested list containing for each group a list with a matrix with the interpolated factor values ("mean"), their variance ("variance") and the values of the covariate at which interpolation took place ("new_values"). Alternatively, if `as.data.frame` is `TRUE`, returns a long-formatted data frame with columns containing the covariates and (factor, group, mean and variance).

`get_lengthscales` *Get lengthscales*

Description

Extract the inferred lengthscale for each factor after model training.

Usage

```
get_lengthscales(object)
```

Arguments

`object`	a `MOFA` object.

Details

This can be used only if covariates are passed to the MOFAobject upon creation and GP_factors is set to True.

Value

A numeric vector containing the lengthscale for each factor.

Examples

```
# Using an existing trained model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
ls <- get_lengthscales(model)
```

get_scales *Get scales*

Description

Extract the inferred scale for each factor after model training.

Usage

```
get_scales(object)
```

Arguments

object	a MOFA object.

Details

This can be used only if covariates are passed to the MOFAobject upon creation and GP_factors is set to True.

Value

A numeric vector containing the scale for each factor.

Examples

```
# Using an existing trained model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
s <- get_scales(model)
```

get_variance_explained *Get variance explained values*

Description

Extract the latent factors from the model.

Usage

```
get_variance_explained(
  object,
  groups = "all",
  views = "all",
  factors = "all",
  as.data.frame = FALSE
)
```

Arguments

object	a trained MOFA object.
groups	character vector with the group name(s), or numeric vector with the group index(es). Default is "all".
views	character vector with the view name(s), or numeric vector with the view index(es). Default is "all".
factors	character vector with the factor name(s), or numeric vector with the factor index(es). Default is "all".
as.data.frame	logical indicating whether to return a long data frame instead of a matrix. Default is FALSE.

Value

A list of data matrices with variance explained per group or a data.frame (if as.data.frame is TRUE)

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Fetch variance explained values (in matrix format)
r2 <- get_variance_explained(model)

# Fetch variance explained values (in data.frame format)
r2 <- get_variance_explained(model, as.data.frame = TRUE)
```

---

get_weights *Get weights*

Description

Extract the weights from the model.

Usage

```
get_weights(
  object,
  views = "all",
  factors = "all",
  abs = FALSE,
  scale = FALSE,
  as.data.frame = FALSE
)
```

Arguments

object	a trained MOFA object.
views	character vector with the view name(s), or numeric vector with the view index(es). Default is "all".
factors	character vector with the factor name(s) or numeric vector with the factor index(es). Default is "all".
abs	logical indicating whether to take the absolute value of the weights.
scale	logical indicating whether to scale all weights from -1 to 1 (or from 0 to 1 if abs=TRUE).
as.data.frame	logical indicating whether to return a long data frame instead of a list of matrices. Default is FALSE.

Value

By default it returns a list where each element is a loading matrix with dimensionality (D,K), where D is the number of features and K is the number of factors.
Alternatively, if as.data.frame is TRUE, returns a long-formatted data frame with columns (view,feature,factor,value).

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Fetch weights in matrix format (a list, one matrix per view)
weights <- get_weights(model)

# Fetch weights for factor 1 and 2 and view 1
weights <- get_weights(model, views = 1, factors = c(1,2))

# Fetch weights in data.frame format
weights <- get_weights(model, as.data.frame = TRUE)
```

---

groups_names *groups_names: set and retrieve group names*

Description

groups_names: set and retrieve group names

Usage

```
groups_names(object)

groups_names(object) <- value

## S4 method for signature 'MOFA'
groups_names(object)

## S4 replacement method for signature 'MOFA,character'
groups_names(object) <- value
```

Arguments

object	a MOFA object.
value	character vector with the names for each group

Value

character vector with the names for each sample group

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
groups_names(model)
groups_names(model) <- c("my_group")
```

---

impute *Impute missing values from a fitted MOFA*

Description

This function uses the latent factors and the loadings to impute missing values.

Usage

```
impute(
  object,
  views = "all",
  groups = "all",
  factors = "all",
  add_intercept = TRUE
)
```

Arguments

object	a MOFA object.
views	character vector with the view name(s), or numeric vector with view index(es).
groups	character vector with the group name(s), or numeric vector with group index(es).
factors	character vector with the factor names, or numeric vector with the factor index(es). • **response**:gives mean for gaussian and poisson and probabilities for bernoulli. • **link**: gives the linear predictions. • **inRange**: rounds the fitted values from "terms" for integer-valued distributions to the next integer (default).
add_intercept	add feature intercepts to the imputation (default is TRUE).

Details

MOFA generates a denoised and condensed low-dimensional representation of the data that captures the main sources of heterogeneity of the data. This representation can be used to reconstruct the data, simply using the equation Y = WX. For more details read the supplementary methods of the manuscript.
Note that with impute you can only generate the point estimates (the means of the posterior distributions). If you want to add uncertainity estimates (the variance) you need to set impute=TRUE in the training options. See get_default_training_options.

Value

This method fills the imputed_data slot by replacing the missing values in the input data with the model predictions.

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Impute missing values in all data modalities
imputed_data <- impute(model, views = "all")

# Impute missing values in all data modalities using factors 1:3
imputed_data <- impute(model, views = "all", factors = 1:3)
```

---

interpolate_factors *Interpolate factors in MEFISTO based on new covariate values*

Description

Function to interpolate factors in MEFISTO based on new covariate values.

Usage

```
interpolate_factors(object, new_values)
```

Arguments

object	a MOFA object trained with MEFISTO options and a covariate
new_values	a matrix containing the new covariate values to inter/extrapolate to. Should be in the same format as the covariated used for training.

Details

This function requires the functional MEFISTO framework to be used in training. Use `set_covariates` and specify mefisto_options when preparing the training using `prepare_mofa`. Currenlty, only the mean of the interpolation is provided from R.

Value

Returns the MOFA with interpolated factor values filled in the corresponding slot (interpolatedZ)

Examples

```
# Using an existing trained model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
model <- interpolate_factors(model, new_values = seq(0,1.1,0.01))
```

---

load_model *Load a trained MOFA*

Description

Method to load a trained MOFA
The training of mofa is done using a Python framework, and the model output is saved as an .hdf5 file, which has to be loaded in the R package.

Usage

```
load_model(
  file,
  sort_factors = TRUE,
  on_disk = FALSE,
  load_data = TRUE,
  remove_outliers = FALSE,
  remove_inactive_factors = TRUE,
  verbose = FALSE,
  load_interpol_Z = FALSE
)
```

Arguments

file	an hdf5 file saved by the mofa Python framework
sort_factors	logical indicating whether factors should be sorted by variance explained (default is TRUE)
on_disk	logical indicating whether to work from memory (FALSE) or disk (TRUE). This should be set to TRUE when the training data is so big that cannot fit into memory. On-disk operations are performed using the HDF5Array and DelayedArray framework.
load_data	logical indicating whether to load the training data (default is TRUE, it can be memory expensive)
remove_outliers	logical indicating whether to mask outlier values.
remove_inactive_factors	logical indicating whether to remove inactive factors from the model.
verbose	logical indicating whether to print verbose output (default is FALSE)
load_interpol_Z	(MEFISTO) logical indicating whether to load predictions for factor values based on latent processed (only relevant for models trained with covariates and Gaussian processes, where prediction was enabled)

Value

a MOFA model

Examples

```
#' # Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
```

---

make_example_data *Simulate a data set using the generative model of MOFA*

Description

Function to simulate an example multi-view multi-group data set according to the generative model of MOFA2.

Usage

```
make_example_data(
  n_views = 3,
  n_features = 100,
  n_samples = 50,
  n_groups = 1,
  n_factors = 5,
  likelihood = "gaussian",
  lscales = 1,
  sample_cov = NULL,
  as.data.frame = FALSE
)
```

Arguments

n_views	number of views
n_features	number of features in each view
n_samples	number of samples in each group
n_groups	number of groups
n_factors	number of factors
likelihood	likelihood for each view, one of "gaussian" (default), "bernoulli", "poisson", or a character vector of length n_views
lscales	vector of lengthscales, needs to be of length n_factors (default is 0 - no smooth factors)
sample_cov	(only for use with MEFISTO) matrix of sample covariates for one group with covariates in rows and samples in columns or "equidistant" for sequential ordering, default is NULL (no smooth factors)
as.data.frame	return data and covariates as long dataframe

Value

Returns a list containing the simulated data and simulation parameters.

Examples

```
# Generate a simulated data set
MOFAexample <- make_example_data()
```

MOFA *Class to store a mofa model*

Description

The MOFA is an S4 class used to store all relevant data to analyse a MOFA model

Slots

`data` The input data

`intercepts` Feature intercepts

`samples_metadata` Samples metadata

`features_metadata` Features metadata.

`imputed_data` The imputed data.

`expectations` expected values of the factors and the loadings.

`dim_red` non-linear dimensionality reduction manifolds.

`training_stats` model training statistics.

`data_options` Data processing options.

`training_options` Model training options.

`stochastic_options` Stochastic variational inference options.

`model_options` Model options.

`mefisto_options` Options for the use of MEFISO

`dimensions` Dimensionalities of the model: M for the number of views, G for the number of groups, N for the number of samples (per group), C for the number of covariates per sample, D for the number of features (per view), K for the number of factors.

`on_disk` Logical indicating whether data is loaded from disk.

`cache` Cache.

`status` Auxiliary variable indicating whether the model has been trained.

`covariates` optional slot to store sample covariate for training in MEFISTO

`covariates_warped` optional slot to store warped sample covariate for training in MEFISTO

`interpolated_Z` optional slot to store interpolated factor values (used only with MEFISTO)

plot_alignment *Plot covariate alignment acorss groups*

Description

Function to plot the alignment learnt by MEFISTO for the covariate values between different groups

Usage

```
plot_alignment(object)
```

Arguments

object	a MOFA object using MEFISTO with warping

Details

This function requires the functional MEFISTO framework to be used in training. Use `set_covariates` and specify mefisto_options when preparing the training using `prepare_mofa`.

Value

ggplot object showing the alignment

plot_ascii_data *Visualize the structure of the data in the terminal*

Description

A Fancy printing method

Usage

```
plot_ascii_data(object, nonzero = FALSE)
```

Arguments

object	a MOFA object
nonzero	a logical value specifying whether to calculate the fraction of non-zero values (non-NA values by default)

Details

This function is helpful to get an overview of the structure of the data as a text output

Value

None

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_ascii_data(model)
```

`plot_data_heatmap` *Plot heatmap of relevant features*

Description

Function to plot a heatmap of the data for relevant features, typically the ones with high weights.

Usage

```
plot_data_heatmap(
  object,
  factor,
  view = 1,
  groups = "all",
  features = 50,
  annotation_features = NULL,
  annotation_samples = NULL,
  transpose = FALSE,
  imputed = FALSE,
  denoise = FALSE,
  max.value = NULL,
  min.value = NULL,
  ...
)
```

Arguments

`object`	a MOFA object.
`factor`	a string with the factor name, or an integer with the index of the factor.
`view`	a string with the view name, or an integer with the index of the view. Default is the first view.
`groups`	groups to plot. Default is "all".
`features`	if an integer (default), the total number of features to plot based on the absolute value of the weights. If a character vector, a set of manually defined features.

annotation_features
: annotation metadata for features (rows). Either a character vector specifying columns in the feature metadata, or a data.frame that will be passed to pheatmap as annotation_col

annotation_samples
: annotation metadata for samples (columns). Either a character vector specifying columns in the sample metadata, or a data.frame that will be passed to pheatmap as annotation_row

transpose
: logical indicating whether to transpose the heatmap. Default corresponds to features as rows and samples as columns.

imputed
: logical indicating whether to plot the imputed data instead of the original data. Default is FALSE.

denoise
: logical indicating whether to plot a denoised version of the data reconstructed using the MOFA factors.

max.value
: numeric indicating the maximum value to display in the heatmap (i.e. the matrix values will be capped at max.value).

min.value
: numeric indicating the minimum value to display in the heatmap (i.e. the matrix values will be capped at min.value). See predict. Default is FALSE.

...
: further arguments that can be passed to pheatmap

Details

One of the first steps for the annotation of a given factor is to visualise the corresponding weights, using for example plot_weights or plot_top_weights.
However, one might also be interested in visualising the direct relationship between features and factors, rather than looking at "abstract" weights.
This function generates a heatmap for selected features, which should reveal the underlying pattern that is captured by the latent factor.
A similar function for doing scatterplots rather than heatmaps is plot_data_scatter.

Value

A pheatmap object

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_data_heatmap(model, factor = 1, show_rownames = FALSE, show_colnames = FALSE)
```

plot_data_overview *Overview of the input data*

Description

Function to do a tile plot showing the missing value structure of the input data

Usage

```
plot_data_overview(
  object,
  covariate = 1,
  colors = NULL,
  show_covariate = FALSE,
  show_dimensions = TRUE
)
```

Arguments

object	a MOFA object.
covariate	(only for MEFISTO) specifies sample covariate to order samples by in the plot. This should be a character or a numeric index giving the name or position of a column present in the covariates slot of the object. Default is the first sample covariate in covariates slot. NULL does not order by covariate
colors	a vector specifying the colors per view (see example for details).
show_covariate	(only for MEFISTO) boolean specifying whether to include the covariate in the plot
show_dimensions	logical indicating whether to plot the dimensions of the data (default is TRUE).

Details

This function is helpful to get an overview of the structure of the data. It shows the model dimensionalities (number of samples, groups, views and features) and it indicates which measurements are missing.

Value

A ggplot object

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_data_overview(model)
```

plot_data_scatter *Scatterplots of feature values against latent factors*

Description

Function to do a scatterplot of features against factor values.

Usage

```
plot_data_scatter(
  object,
  factor = 1,
  view = 1,
  groups = "all",
  features = 10,
  sign = "all",
  color_by = "group",
  legend = TRUE,
  alpha = 1,
  shape_by = NULL,
  stroke = NULL,
  dot_size = 2.5,
  text_size = NULL,
  add_lm = TRUE,
  lm_per_group = TRUE,
  imputed = FALSE
)
```

Arguments

object	a MOFA object.
factor	string with the factor name, or an integer with the index of the factor.
view	string with the view name, or an integer with the index of the view. Default is the first view.
groups	groups to plot. Default is "all".
features	if an integer (default), the total number of features to plot. If a character vector, a set of manually-defined features.
sign	can be 'positive', 'negative' or 'all' (default) to show only positive, negative or all weights, respectively.
color_by	specifies groups or values (either discrete or continuous) used to color the dots (samples). This can be either: • the string "group": dots are coloured with respect to their predefined groups. • a character giving the name of a feature that is present in the input data • a character giving the same of a column in the sample metadata slot

- a vector of the same length as the number of samples specifying the value for each sample.
- a dataframe with two columns: "sample" and "color"

`legend` logical indicating whether to add a legend

`alpha` numeric indicating dot transparency (default is 1).

`shape_by` specifies groups or values (only discrete) used to shape the dots (samples). This can be either:

- the string "group": dots are shaped with respect to their predefined groups.
- a character giving the name of a feature that is present in the input data
- a character giving the same of a column in the sample metadata slot
- a vector of the same length as the number of samples specifying the value for each sample.
- a dataframe with two columns: "sample" and "shape"

`stroke` numeric indicating the stroke size (the black border around the dots, default is NULL, infered automatically).

`dot_size` numeric indicating dot size (default is 5).

`text_size` numeric indicating text size (default is 5).

`add_lm` logical indicating whether to add a linear regression line for each plot

`lm_per_group` logical indicating whether to add a linear regression line separately for each group

`imputed` logical indicating whether to include imputed measurements

Details

One of the first steps for the annotation of factors is to visualise the weights using `plot_weights` or `plot_top_weights`. However, one might also be interested in visualising the direct relationship between features and factors, rather than looking at "abstract" weights.
A similar function for doing heatmaps rather than scatterplots is `plot_data_heatmap`.

Value

A `ggplot` object

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_data_scatter(model)
```

plot_data_vs_cov *Scatterplots of feature values against sample covariates*

Description

Function to do a scatterplot of features against sample covariate values.

Usage

```
plot_data_vs_cov(
  object,
  covariate = 1,
  warped = TRUE,
  factor = 1,
  view = 1,
  groups = "all",
  features = 10,
  sign = "all",
  color_by = "group",
  legend = TRUE,
  alpha = 1,
  shape_by = NULL,
  stroke = NULL,
  dot_size = 2.5,
  text_size = NULL,
  add_lm = FALSE,
  lm_per_group = FALSE,
  imputed = FALSE,
  return_data = FALSE
)
```

Arguments

object	a MOFA object using MEFISTO.
covariate	string with the covariate name or a samples_metadata column, or an integer with the index of the covariate
warped	logical indicating whether to show the aligned covariate (default: TRUE), only relevant if warping has been used to align multiple sample groups
factor	string with the factor name, or an integer with the index of the factor to take top features from
view	string with the view name, or an integer with the index of the view. Default is the first view.
groups	groups to plot. Default is "all".
features	if an integer (default), the total number of features to plot (given by highest weights). If a character vector, a set of manually-defined features.

sign	can be 'positive', 'negative' or 'all' (default) to show only features with highest positive, negative or all weights, respectively.
color_by	specifies groups or values (either discrete or continuous) used to color the dots (samples). This can be either: • the string "group": dots are coloured with respect to their predefined groups. • a character giving the name of a feature that is present in the input data • a character giving the same of a column in the sample metadata slot • a vector of the same length as the number of samples specifying the value for each sample. • a dataframe with two columns: "sample" and "color"
legend	logical indicating whether to add a legend
alpha	numeric indicating dot transparency (default is 1).
shape_by	specifies groups or values (only discrete) used to shape the dots (samples). This can be either: • the string "group": dots are shaped with respect to their predefined groups. • a character giving the name of a feature that is present in the input data • a character giving the same of a column in the sample metadata slot • a vector of the same length as the number of samples specifying the value for each sample. • a dataframe with two columns: "sample" and "shape"
stroke	numeric indicating the stroke size (the black border around the dots, default is NULL, inferred automatically).
dot_size	numeric indicating dot size (default is 5).
text_size	numeric indicating text size (default is 5).
add_lm	logical indicating whether to add a linear regression line for each plot
lm_per_group	logical indicating whether to add a linear regression line separately for each group
imputed	logical indicating whether to include imputed measurements
return_data	logical indicating whether to return a data frame instead of a plot

Details

One of the first steps for the annotation of factors is to visualise the weights using `plot_weights` or `plot_top_weights` and inspect the relationshio of the factor to the covariate(s) using `plot_factors_vs_cov`. However, one might also be interested in visualising the direct relationship between features and covariate(s), rather than looking at "abstract" weights and possibly look at the interpolated and extrapolated values by setting imputed to True.

Value

Returns a `ggplot2` object or the underlying dataframe if return_data is set to `TRUE`.

Examples

```
# Using an existing trained model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_data_vs_cov(model, factor = 3, features = 2)
```

`plot_dimred` *Plot dimensionality reduction based on MOFA factors*

Description

Plot dimensionality reduction based on MOFA factors

Usage

```
plot_dimred(
  object,
  method = c("UMAP", "TSNE"),
  groups = "all",
  show_missing = TRUE,
  color_by = NULL,
  shape_by = NULL,
  color_name = NULL,
  shape_name = NULL,
  label = FALSE,
  dot_size = 1.5,
  stroke = NULL,
  alpha_missing = 1,
  legend = TRUE,
  rasterize = FALSE,
  return_data = FALSE,
  ...
)
```

Arguments

`object`	a trained `MOFA` object.
`method`	string indicating which method has been used for non-linear dimensionality reduction (either 'umap' or 'tsne')
`groups`	character vector with the groups names, or numeric vector with the indices of the groups of samples to use, or "all" to use samples from all groups.
`show_missing`	logical indicating whether to include samples for which `shape_by` or `color_by` is missing

color_by	specifies groups or values used to color the samples. This can be either: (1) a character giving the name of a feature present in the training data. (2) a character giving the same of a column present in the sample metadata. (3) a vector of the same length as the number of samples specifying discrete groups or continuous numeric values.
shape_by	specifies groups or values used to shape the samples. This can be either: (1) a character giving the name of a feature present in the training data, (2) a character giving the same of a column present in the sample metadata. (3) a vector of the same length as the number of samples specifying discrete groups.
color_name	name for color legend.
shape_name	name for shape legend.
label	logical indicating whether to label the medians of the clusters. Only if color_by is specified
dot_size	numeric indicating dot size.
stroke	numeric indicating the stroke size (the black border around the dots, default is NULL, infered automatically).
alpha_missing	numeric indicating dot transparency of missing data.
legend	logical indicating whether to add legend.
rasterize	logical indicating whether to rasterize plot using geom_point_rast
return_data	logical indicating whether to return the long data frame to plot instead of plotting
...	extra arguments passed to run_umap or run_tsne.

Details

This function plots dimensionality reduction projections that are stored in the dim_red slot. Typically this contains UMAP or t-SNE projections computed using run_tsne or run_umap, respectively.

Value

Returns a ggplot2 object or a long data.frame (if return_data is TRUE)

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Run UMAP
model <- run_umap(model)

# Plot UMAP
plot_dimred(model, method = "UMAP")

# Plot UMAP, colour by Factor 1 values
plot_dimred(model, method = "UMAP", color_by = "Factor1")
```

```
# Plot UMAP, colour by the values of a specific feature
plot_dimred(model, method = "UMAP", color_by = "feature_0_view_0")
```

`plot_enrichment` *Plot output of gene set Enrichment Analysis*

Description

Method to plot the results of the gene set Enrichment Analyisis

Usage

```
plot_enrichment(
  enrichment.results,
  factor,
  alpha = 0.1,
  max.pathways = 25,
  text_size = 1,
  dot_size = 5
)
```

Arguments

`enrichment.results`	output of run_enrichment function
`factor`	a string with the factor name or an integer with the factor index
`alpha`	p.value threshold to filter out gene sets
`max.pathways`	maximum number of enriched pathways to display
`text_size`	text size
`dot_size`	dot size

Details

it requires `run_enrichment` to be run beforehand.

Value

a `ggplot2` object

`plot_enrichment_detailed` *Plot detailed output of the Feature Set Enrichment Analysis*

Description

Method to plot a detailed output of the Feature Set Enrichment Analyisis (FSEA).
Each row corresponds to a significant pathway, sorted by statistical significance, and each dot corresponds to a gene.
For each pathway, we display the top genes of the pathway sorted by the corresponding feature statistic (by default, the absolute value of the weight) The top genes with the highest statistic (max.genes argument) are displayed and labeled in black. The remaining genes are colored in grey.

Usage

```
plot_enrichment_detailed(
  enrichment.results,
  factor,
  alpha = 0.1,
  max.genes = 5,
  max.pathways = 10,
  text_size = 3
)
```

Arguments

`enrichment.results`	output of run_enrichment function
`factor`	string with factor name or numeric with factor index
`alpha`	p.value threshold to filter out feature sets
`max.genes`	maximum number of genes to display, per pathway
`max.pathways`	maximum number of enriched pathways to display
`text_size`	size of the text to label the top genes

Value

a `ggplot2` object

plot_enrichment_heatmap *Heatmap of Feature Set Enrichment Analysis results*

Description

This method generates a heatmap with the adjusted p.values that result from the the feature set enrichment analysis. Rows are feature sets and columns are factors.

Usage

```
plot_enrichment_heatmap(
  enrichment.results,
  alpha = 0.1,
  cap = 1e-50,
  log_scale = TRUE,
  ...
)
```

Arguments

enrichment.results	output of run_enrichment function
alpha	FDR threshold to filter out unsignificant feature sets which are not represented in the heatmap. Default is 0.10.
cap	cap p-values below this threshold
log_scale	logical indicating whether to plot the -log of the p.values.
...	extra arguments to be passed to the pheatmap function

Value

produces a heatmap

plot_factor *Beeswarm plot of factor values*

Description

Beeswarm plot of the latent factor values.

Usage

```
plot_factor(
  object,
  factors = 1,
  groups = "all",
  group_by = "group",
  color_by = "group",
  shape_by = NULL,
  add_dots = TRUE,
  dot_size = 2,
  dot_alpha = 1,
  add_violin = FALSE,
  violin_alpha = 0.5,
  color_violin = TRUE,
  add_boxplot = FALSE,
  boxplot_alpha = 0.5,
  color_boxplot = TRUE,
  show_missing = TRUE,
  scale = FALSE,
  dodge = FALSE,
  color_name = "",
  shape_name = "",
  stroke = NULL,
  legend = TRUE,
  rasterize = FALSE
)
```

Arguments

object	a trained MOFA object.
factors	character vector with the factor names, or numeric vector with the indices of the factors to use, or "all" to plot all factors.
groups	character vector with the groups names, or numeric vector with the indices of the groups of samples to use, or "all" to use samples from all groups.
group_by	specifies grouping of samples: • (default) the string "group": in this case, the plot will color samples with respect to their predefined groups. • a character giving the name of a feature that is present in the input data • a character giving the name of a column in the sample metadata slot • a vector of the same length as the number of samples specifying the value for each sample.
color_by	specifies color of samples. This can be either: • (default) the string "group": in this case, the plot will color the dots with respect to their predefined groups. • a character giving the name of a feature that is present in the input data • a character giving the name of a column in the sample metadata slot

- a vector of the same length as the number of samples specifying the value for each sample.

`shape_by` specifies shape of samples. This can be either:

- (default) the string "group": in this case, the plot will shape the dots with respect to their predefined groups.
- a character giving the name of a feature that is present in the input data
- a character giving the name of a column in the sample metadata slot
- a vector of the same length as the number of samples specifying the value for each sample.

`add_dots` logical indicating whether to add dots.

`dot_size` numeric indicating dot size.

`dot_alpha` numeric indicating dot transparency.

`add_violin` logical indicating whether to add violin plots

`violin_alpha` numeric indicating violin plot transparency.

`color_violin` logical indicating whether to color violin plots.

`add_boxplot` logical indicating whether to add box plots

`boxplot_alpha` numeric indicating boxplot transparency.

`color_boxplot` logical indicating whether to color box plots.

`show_missing` logical indicating whether to remove samples for which `shape_by` or `color_by` is missing.

`scale` logical indicating whether to scale factor values.

`dodge` logical indicating whether to dodge the dots (default is FALSE).

`color_name` name for color legend (usually only used if color_by is not a character itself).

`shape_name` name for shape legend (usually only used if shape_by is not a character itself).

`stroke` numeric indicating the stroke size (the black border around the dots).

`legend` logical indicating whether to add a legend to the plot (default is TRUE).

`rasterize` logical indicating whether to rasterize the plot (default is FALSE).

Details

One of the main steps for the annotation of factors is to visualise and color them using known covariates or phenotypic data.
This function generates a Beeswarm plot of the sample values in a given latent factor.
Similar functions are `plot_factors` for doing scatter plots.

Value

Returns a `ggplot2`

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Plot Factors 1 and 2 and colour by "group"
plot_factor(model, factors = c(1,2), color_by="group")

# Plot Factor 3 and colour by the value of a specific feature
plot_factor(model, factors = 3, color_by="feature_981_view_1")

# Add violin plots
plot_factor(model, factors = c(1,2), color_by="group", add_violin = TRUE)

# Scale factor values from -1 to 1
plot_factor(model, factors = c(1,2), scale = TRUE)
```

---

plot_factors *Scatterplots of two factor values*

Description

Scatterplot of the values of two latent factors.

Usage

```
plot_factors(
  object,
  factors = c(1, 2),
  groups = "all",
  show_missing = TRUE,
  scale = FALSE,
  color_by = NULL,
  shape_by = NULL,
  color_name = NULL,
  shape_name = NULL,
  dot_size = 2,
  alpha = 1,
  legend = TRUE,
  stroke = NULL,
  return_data = FALSE
)
```

Arguments

object	a trained MOFA object.

factors	a vector of length two with the factors to plot. Factors can be specified either as a characters
groups	character vector with the groups names, or numeric vector with the indices of the groups of samples to use, or "all" to use samples from all groups.
show_missing	logical indicating whether to include samples for which shape_by or color_by is missing
scale	logical indicating whether to scale factor values.
color_by	specifies groups or values used to color the samples. This can be either: (1) a character giving the name of a feature present in the training data. (2) a character giving the name of a column present in the sample metadata. (3) a vector of the name length as the number of samples specifying discrete groups or continuous numeric values.
shape_by	specifies groups or values used to shape the samples. This can be either: (1) a character giving the name of a feature present in the training data, (2) a character giving the name of a column present in the sample metadata. (3) a vector of the same length as the number of samples specifying discrete groups.
color_name	name for color legend.
shape_name	name for shape legend.
dot_size	numeric indicating dot size (default is 2).
alpha	numeric indicating dot transparency (default is 1).
legend	logical indicating whether to add legend.
stroke	numeric indicating the stroke size (the black border around the dots, default is NULL, infered automatically).
return_data	logical indicating whether to return the data frame to plot instead of plotting

Details

One of the first steps for the annotation of factors is to visualise and group/color them using known covariates such as phenotypic or clinical data. This method generates a single scatterplot for the combination of two latent factors. TO-FINISH... plot_factors for doing Beeswarm plots for factors.

Value

Returns a ggplot2 object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Scatterplot of factors 1 and 2
plot_factors(model, factors = c(1,2))

# Shape dots by a column in the metadata
```

```
plot_factors(model, factors = c(1,2), shape_by="group")

# Scale factor values from -1 to 1
plot_factors(model, factors = c(1,2), scale = TRUE)
```

`plot_factors_vs_cov` *Scatterplots of a factor's values againt the sample covariates*

Description

Scatterplots of a factor's values againt the sample covariates

Usage

```
plot_factors_vs_cov(
  object,
  factors = "all",
  covariates = NULL,
  warped = TRUE,
  show_missing = TRUE,
  scale = FALSE,
  color_by = NULL,
  shape_by = NULL,
  color_name = NULL,
  shape_name = NULL,
  dot_size = 1.5,
  alpha = 1,
  stroke = NULL,
  legend = TRUE,
  rotate_x = FALSE,
  rotate_y = FALSE,
  return_data = FALSE,
  show_variance = FALSE
)
```

Arguments

`object`	a trained MOFA object using MEFISTO.
`factors`	character or numeric specifying the factor(s) to plot, default is "all"
`covariates`	specifies sample covariate(s) to plot against: (1) a character giving the name of a column present in the sample covariates or sample metadata. (2) a character giving the name of a feature present in the training data. (3) a vector of the same length as the number of samples specifying continuous numeric values per sample. Default is the first sample covariates in covariates slot
`warped`	logical indicating whether to show the aligned covariate (default: TRUE), only relevant if warping has been used to align multiple sample groups

show_missing	(for 1-dim covariates) logical indicating whether to include samples for which shape_by or color_by is missing
scale	logical indicating whether to scale factor values.
color_by	(for 1-dim covariates) specifies groups or values used to color the samples. This can be either: (1) a character giving the name of a feature present in the training data. (2) a character giving the same of a column present in the sample metadata. (3) a vector of the same length as the number of samples specifying discrete groups or continuous numeric values.
shape_by	(for 1-dim covariates) specifies groups or values used to shape the samples. This can be either: (1) a character giving the name of a feature present in the training data, (2) a character giving the same of a column present in the sample metadata. (3) a vector of the same length as the number of samples specifying discrete groups.
color_name	(for 1-dim covariates) name for color legend.
shape_name	(for 1-dim covariates) name for shape legend.
dot_size	(for 1-dim covariates) numeric indicating dot size.
alpha	(for 1-dim covariates) numeric indicating dot transparency.
stroke	(for 1-dim covariates) numeric indicating the stroke size
legend	(for 1-dim covariates) logical indicating whether to add legend.
rotate_x	(for spatial, 2-dim covariates) Rotate covariate on x-axis
rotate_y	(for spatial, 2-dim covariates) Rotate covariate on y-axis
return_data	logical indicating whether to return the data frame to plot instead of plotting
show_variance	(for 1-dim covariates) logical indicating whether to show the marginal variance of inferred factor values (only relevant for 1-dimensional covariates)

Details

To investigate the factors pattern along the covariates (such as time or a spatial coordinate) this function an be used to plot a scatterplot of the factor againt the values of each covariate

Value

Returns a ggplot2 object

Examples

```
# Using an existing trained model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_factors_vs_cov(model)
```

plot_factor_cor *Plot correlation matrix between latent factors*

Description

Function to plot the correlation matrix between the latent factors.

Usage

```
plot_factor_cor(object, method = "pearson", ...)
```

Arguments

object	a trained MOFA object.
method	a character indicating the type of correlation coefficient to be computed: pearson (default), kendall, or spearman.
...	arguments passed to corrplot

Details

This method plots the correlation matrix between the latent factors.
The model encourages the factors to be uncorrelated, so this function usually yields a diagonal correlation matrix.
However, it is not a hard constraint such as in Principal Component Analysis and correlations between factors can occur, particularly with large number factors.
Generally, correlated factors are redundant and should be avoided, as they make interpretation harder. Therefore, if you have too many correlated factors we suggest you try reducing the number of factors.

Value

Returns a symmetric matrix with the correlation coefficient between every pair of factors.

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Plot correlation between all factors
plot_factor_cor(model)
```

plot_group_kernel *Heatmap plot showing the group-group correlations per factor*

Description

Heatmap plot showing the group-group correlations inferred by the model per factor

Usage

```
plot_group_kernel(object, factors = "all", groups = "all", ...)
```

Arguments

object	a trained MOFA object using MEFISTO.
factors	character vector with the factors names, or numeric vector indicating the indices of the factors to use
groups	character vector with the groups names, or numeric vector with the indices of the groups of samples to use, or "all" to use samples from all groups.
...	additional parameters that can be passed to pheatmap

Details

The heatmap gives insight into the clustering of the patterns that factors display along the covariate in each group. A correlation of 1 indicates that the module caputred by a factor shows identical patterns across groups, a correlation of zero that it shows distinct patterns, a negative correlation that the patterns go in opposite directions.

Value

Returns a ggplot,gg object containing the heatmaps

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_group_kernel(model)
```

`plot_interpolation_vs_covariate` *Plot interpolated factors versus covariate (1-dimensional)*

Description

make a plot of interpolated covariates versus covariate

Usage

```
plot_interpolation_vs_covariate(
  object,
  covariate = 1,
  factors = "all",
  only_mean = TRUE,
  show_observed = TRUE
)
```

Arguments

`object`	a trained `MOFA` object using MEFISTO.
`covariate`	covariate to use for plotting
`factors`	character or numeric specifying the factor(s) to plot, default is "all"
`only_mean`	show only mean or include uncertainties?
`show_observed`	include observed factor values as dots on the plot

Details

to be filled

Value

Returns a `ggplot2` object

Examples

```
# Using an existing trained model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
model <- interpolate_factors(model, new_values = seq(0,1.1,0.1))
plot_interpolation_vs_covariate(model, covariate = "time", factors = 1)
```

plot_sharedness *Barplot showing the sharedness per factor*

Description

Barplot indicating a sharedness score (between 0 (non-shared) and 1 (shared)) per factor

Usage

```
plot_sharedness(object, factors = "all", color = "#B8CF87")
```

Arguments

object	a trained MOFA object using MEFISTO.
factors	character vector with the factors names, or numeric vector indicating the indices of the factors to use
color	for the shared part of the bar

Details

The sharedness score is calculated as the distance of the learnt group correlation matrix to the identity matrix in terms of the mean absolute distance on the off-diagonal elements.

Value

Returns a ggplot2 object

plot_smoothness *Barplot showing the smoothness per factor*

Description

Barplot indicating a smoothness score (between 0 (non-smooth) and 1 (smooth)) per factor

Usage

```
plot_smoothness(object, factors = "all", color = "cadetblue")
```

Arguments

object	a trained MOFA object using MEFISTO.
factors	character vector with the factors names, or numeric vector indicating the indices of the factors to use
color	for the smooth part of the bar

Details

The smoothness score is given by the scale parameter for the underlying Gaussian process of each factor.

Value

Returns a ggplot2 object

Examples

```
# Using an existing trained model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
smoothness_bars <- plot_smoothness(model)
```

---

plot_top_weights *Plot top weights*

Description

Plot top weights for a given factor and view.

Usage

```
plot_top_weights(
  object,
  view = 1,
  factors = 1,
  nfeatures = 10,
  abs = TRUE,
  scale = TRUE,
  sign = "all"
)
```

Arguments

object	a trained MOFA object.
view	a string with the view name, or an integer with the index of the view.
factors	a character string with factors names, or an integer vector with factors indices.
nfeatures	number of top features to display. Default is 10
abs	logical indicating whether to use the absolute value of the weights (Default is FALSE).
scale	logical indicating whether to scale all weights from -1 to 1 (or from 0 to 1 if abs=TRUE). Default is TRUE.
sign	can be 'positive', 'negative' or 'all' to show only positive, negative or all weights, respectively. Default is 'all'.

Details

An important step to annotate factors is to visualise the corresponding feature weights.
This function displays the top features with highest loading whereas the function `plot_top_weights` plots all weights for a given latent factor and view.
Importantly, the weights of the features within a view have relative values and they should not be interpreted in an absolute scale. Therefore, for interpretability purposes we always recommend to scale the weights with `scale=TRUE`.

Value

Returns a `ggplot2` object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Plot top weights for Factors 1 and 2 and View 1
plot_top_weights(model, view = 1, factors = c(1,2))

# Do not take absolute value
plot_weights(model, abs = FALSE)
```

---

plot_variance_explained

Plot variance explained by the model

Description

plots the variance explained by the MOFA factors across different views and groups, as specified by the user. Consider using cowplot::plot_grid(plotlist = ...) to combine the multiple plots that this function generates.

Usage

```
plot_variance_explained(
  object,
  x = "view",
  y = "factor",
  split_by = NA,
  plot_total = FALSE,
  factors = "all",
  min_r2 = 0,
  max_r2 = NULL,
  legend = TRUE,
  use_cache = TRUE,
```

```
  ...
)
```

Arguments

object	a MOFA object
x	character specifying the dimension for the x-axis ("view", "factor", or "group").
y	character specifying the dimension for the y-axis ("view", "factor", or "group").
split_by	character specifying the dimension to be faceted ("view", "factor", or "group").
plot_total	logical value to indicate if to plot the total variance explained (for the variable in the x-axis)
factors	character vector with a factor name(s), or numeric vector with the index(es) of the factor(s). Default is "all".
min_r2	minimum variance explained for the color scheme (default is 0).
max_r2	maximum variance explained for the color scheme.
legend	logical indicating whether to add a legend to the plot (default is TRUE).
use_cache	logical indicating whether to use cache (default is TRUE)
...	extra arguments to be passed to calculate_variance_explained

Value

A list of ggplot objects (if plot_total is TRUE) or a single ggplot object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Calculate variance explained (R2)
r2 <- calculate_variance_explained(model)

# Plot variance explained values (view as x-axis, and factor as y-axis)
plot_variance_explained(model, x="view", y="factor")

# Plot variance explained values (view as x-axis, and group as y-axis)
plot_variance_explained(model, x="view", y="group")

# Plot variance explained values for factors 1 to 3
plot_variance_explained(model, x="view", y="group", factors=1:3)

# Scale R2 values
plot_variance_explained(model, max_r2=0.25)
```

`plot_variance_explained_by_covariates` *Plot variance explained by the smooth components of the model*

Description

This function plots the variance explained by the smooth components (Gaussian processes) underlying the factors in MEFISTO across different views and groups, as specified by the user.

Usage

```
plot_variance_explained_by_covariates(
  object,
  factors = "all",
  x = "view",
  y = "factor",
  split_by = NA,
  min_r2 = 0,
  max_r2 = NULL,
  compare_total = FALSE,
  legend = TRUE
)
```

Arguments

`object`	a `MOFA` object
`factors`	character vector with a factor name(s), or numeric vector with the index(es) of the factor(s). Default is "all".
`x`	character specifying the dimension for the x-axis ("view", "factor", or "group").
`y`	character specifying the dimension for the y-axis ("view", "factor", or "group").
`split_by`	character specifying the dimension to be faceted ("view", "factor", or "group").
`min_r2`	minimum variance explained for the color scheme (default is 0).
`max_r2`	maximum variance explained for the color scheme.
`compare_total`	plot corresponding variance explained in total in addition
`legend`	logical indicating whether to add a legend to the plot (default is TRUE).

Details

Note that this function requires the use of MEFISTO. To activate the functional MEFISTO framework, specify mefisto_options when preparing the training using `prepare_mofa`

Value

A list of `ggplot` objects (if `compare_total` is TRUE) or a single `ggplot` object. Consider using cowplot::plot_grid(plotlist = ...) to combine the multiple plots that this function generates.

Examples

```
# load_model
file <- system.file("extdata", "MEFISTO_model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_variance_explained_by_covariates(model)

# compare to toal variance explained
plist <- plot_variance_explained_by_covariates(model, compare_total = TRUE)
cowplot::plot_grid(plotlist = plist)
```

`plot_variance_explained_per_feature` *Plot variance explained by the model for a set of features Returns a tile plot with a group on the X axis and a feature along the Y axis*

Description

Plot variance explained by the model for a set of features

Returns a tile plot with a group on the X axis and a feature along the Y axis

Usage

```
plot_variance_explained_per_feature(
  object,
  view,
  features = 10,
  split_by_factor = FALSE,
  group_features_by = NULL,
  groups = "all",
  factors = "all",
  min_r2 = 0,
  max_r2 = NULL,
  legend = TRUE,
  return_data = FALSE,
  ...
)
```

Arguments

`object`	a MOFA object.
`view`	a view name or index.
`features`	a vector with indices or names for features from the respective view, or number of top features to be fetched by their loadings across specified factors. "all" to plot all features.
`split_by_factor`	logical indicating whether to split R2 per factor or plot R2 jointly

group_features_by	column name of features metadata to group features by
groups	a vector with indices or names for sample groups (default is all)
factors	a vector with indices or names for factors (default is all)
min_r2	minimum variance explained for the color scheme (default is 0).
max_r2	maximum variance explained for the color scheme.
legend	logical indicating whether to add a legend to the plot (default is TRUE).
return_data	logical indicating whether to return the data frame to plot instead of plotting
...	extra arguments to be passed to calculate_variance_explained

Value

ggplot object

Examples

```
# Using an existing trained model
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_variance_explained_per_feature(model, view = 1)
```

---

plot_weights *Plot distribution of feature weights (weights)*

Description

An important step to annotate factors is to visualise the corresponding feature weights.
This function plots all weights for a given latent factor and view, labeling the top ones.
In contrast, the function plot_top_weights displays only the top features with highest loading.

Usage

```
plot_weights(
  object,
  view = 1,
  factors = 1,
  nfeatures = 10,
  color_by = NULL,
  shape_by = NULL,
  abs = FALSE,
  manual = NULL,
  color_manual = NULL,
  scale = TRUE,
  dot_size = 1,
  text_size = 5,
  legend = TRUE,
  return_data = FALSE
)
```

Arguments

object	a MOFA object.
view	a string with the view name, or an integer with the index of the view.
factors	character vector with the factor name(s), or numeric vector with the index of the factor(s).
nfeatures	number of top features to label.
color_by	specifies groups or values (either discrete or continuous) used to color the dots (features). This can be either: • (default) the string "group": in this case, the plot will color the dots with respect to their predefined groups. • a character giving the name of a feature that is present in the input data • a character giving the same of a column in the features metadata slot • a vector of the same length as the number of features specifying the value for each feature • a dataframe with two columns: "feature" and "color"
shape_by	specifies groups or values (only discrete) used to shape the dots (features). This can be either: • (default) the string "group": in this case, the plot will shape the dots with respect to their predefined groups. • a character giving the name of a feature that is present in the input data • a character giving the same of a column in the features metadata slot • a vector of the same length as the number of features specifying the value for each feature • a dataframe with two columns: "feature" and "shape"
abs	logical indicating whether to take the absolute value of the weights.
manual	A nested list of character vectors with features to be manually labelled (see the example for details).
color_manual	a character vector with colors, one for each element of 'manual'
scale	logical indicating whether to scale all weights from -1 to 1 (or from 0 to 1 if abs=TRUE).
dot_size	numeric indicating the dot size.
text_size	numeric indicating the text size.
legend	logical indicating whether to add legend.
return_data	logical indicating whether to return the data frame to plot instead of plotting

Value

A ggplot object or a data.frame if return_data is TRUE

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Plot distribution of weights for Factor 1 and View 1
plot_weights(model, view = 1, factors = 1)

# Plot distribution of weights for Factors 1 to 3 and View 1
plot_weights(model, view = 1, factors = 1:3)

# Take the absolute value and highlight the top 10 features
plot_weights(model, view = 1, factors = 1, nfeatures = 10, abs = TRUE)

# Change size of dots and text
plot_weights(model, view = 1, factors = 1, text_size = 5, dot_size = 1)
```

---

plot_weights_heatmap *Plot heatmap of the weights*

Description

Function to visualize the weights for a given set of factors in a given view.
This is useful to visualize the overall pattern of the weights but not to individually characterise the factors.
To inspect the weights of individual factors, use the functions `plot_weights` and `plot_top_weights`

Usage

```
plot_weights_heatmap(
  object,
  view = 1,
  features = "all",
  factors = "all",
  threshold = 0,
  ...
)
```

Arguments

`object`	a trained `MOFA` object.
`view`	character vector with the view name(s), or numeric vector with the index of the view(s) to use. Default is the first view.
`features`	character vector with the feature name(s), or numeric vector with the index of the feature(s) to use. Default is 'all'.

factors	character vector with the factor name(s), or numeric vector with the index of the factor(s) to use. Default is 'all'.
threshold	threshold on absolute weight values, so that weights with a magnitude below this threshold (in all factors) are removed
...	extra arguments passed to pheatmap.

Value

A pheatmap object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_weights_heatmap(model)
```

---

plot_weights_scatter *Scatterplots of weights*

Description

Scatterplot of the weights values for two factors

Usage

```
plot_weights_scatter(
  object,
  factors,
  view = 1,
  color_by = NULL,
  shape_by = NULL,
  dot_size = 1,
  name_color = "",
  name_shape = "",
  show_missing = TRUE,
  abs = FALSE,
  scale = TRUE,
  legend = TRUE
)
```

Arguments

object	a trained MOFA object.
factors	a vector of length two with the factors to plot. Factors can be specified either as a characters using the factor names, or as numeric with the index of the factors

view	character vector with the voiew name, or numeric vector with the index of the view to use. Default is the first view.
color_by	specifies groups or values used to color the features. This can be either • a character giving the same of a column in the feature metadata slot • a vector specifying the value for each feature. • a dataframe with two columns: "feature" and "color"
shape_by	specifies groups or values used to shape the features. This can be either • a character giving the same of a column in the feature metadata slot • a vector specifying the value for each feature. • a dataframe with two columns: "feature" and "shape"
dot_size	numeric indicating dot size.
name_color	name for color legend (usually only used if color_by is not a character itself)
name_shape	name for shape legend (usually only used if shape_by is not a character itself)
show_missing	logical indicating whether to include dots for which shape_by or color_by is missing
abs	logical indicating whether to take the absolute value of the weights.
scale	logical indicating whether to scale all weights from -1 to 1 (or from 0 to 1 if abs=TRUE).
legend	logical indicating whether to add a legend to the plot (default is TRUE).

Details

One of the first steps for the annotation of factors is to visualise and group/color them using known covariates such as phenotypic or clinical data. This method generates a single scatterplot for the combination of two latent factors.

Value

Returns a ggplot2 object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
plot_weights_scatter(model, factors = 1:2)
```

predict *Do predictions using a fitted MOFA*

Description

This function uses the latent factors and the weights to do data predictions.

Usage

```
predict(
  object,
  views = "all",
  groups = "all",
  factors = "all",
  add_intercept = TRUE
)
```

Arguments

object	a MOFA object.
views	character vector with the view name(s), or numeric vector with the view index(es). Default is "all".
groups	character vector with the group name(s), or numeric vector with the group index(es). Default is "all".
factors	character vector with the factor name(s) or numeric vector with the factor index(es). Default is "all".
add_intercept	add feature intercepts to the prediction (default is TRUE).

Details

MOFA generates a denoised and condensed low-dimensional representation of the data that captures the main sources of heterogeneity of the data. This representation can be used to reconstruct a denoised representation of the data, simply using the equation Y = WX. For more mathematical details read the supplementary methods of the manuscript.

Value

Returns a list with the data reconstructed by the model predictions.

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Predict observations for all data modalities
predictions <- predict(model)
```

prepare_mofa *Prepare a MOFA for training*

Description

Function to prepare a MOFA object for training. It requires defining data, model and training options.

Usage

```
prepare_mofa(
  object,
  data_options = NULL,
  model_options = NULL,
  training_options = NULL,
  stochastic_options = NULL,
  mefisto_options = NULL
)
```

Arguments

object	an untrained MOFA
data_options	list of data_options (see get_default_data_options details). If NULL, default options are used.
model_options	list of model options (see get_default_model_options for details). If NULL, default options are used.
training_options	list of training options (see get_default_training_options for details). If NULL, default options are used.
stochastic_options	list of options for stochastic variational inference (see get_default_stochastic_options for details). If NULL, default options are used.
mefisto_options	list of options for mefisto (see get_default_mefisto_options for details). If NULL, default options are used.

Details

This function is called after creating a MOFA object (using create_mofa) and before starting the training (using run_mofa). Here, we can specify different options for the data (data_options), the model (model_options) and the trainig (training_options, stochastic_options). Take a look at the individual default options for an overview using the get_default_XXX_options functions above.

Value

Returns an untrained MOFA with specified options filled in the corresponding slots

Examples

```
# Using an existing simulated data with two groups and two views
file <- system.file("extdata", "test_data.RData", package = "MOFA2")

# Load data dt (in data.frame format)
load(file)

# Create the MOFA object
MOFAmodel <- create_mofa(dt)

# Prepare MOFA object using default options
MOFAmodel <- prepare_mofa(MOFAmodel)

# Prepare MOFA object changing some of the default model options values
model_opts <- get_default_model_options(MOFAmodel)
model_opts$num_factors <- 10
MOFAmodel <- prepare_mofa(MOFAmodel, model_options = model_opts)
```

`run_enrichment` *Run feature set Enrichment Analysis*

Description

Method to perform feature set enrichment analysis. Here we use a slightly modified version of the `pcgse` function.

Usage

```
run_enrichment(
  object,
  view,
  feature.sets,
  factors = "all",
  set.statistic = c("mean.diff", "rank.sum"),
  statistical.test = c("parametric", "cor.adj.parametric", "permutation"),
  sign = c("all", "positive", "negative"),
  min.size = 10,
  nperm = 1000,
  p.adj.method = "BH",
  alpha = 0.1,
  verbose = TRUE
)
```

Arguments

`object`	a `MOFA` object.
`view`	a character with the view name, or a numeric vector with the index of the view to use.

`feature.sets` data structure that holds feature set membership information. Must be a binary membership matrix (rows are feature sets and columns are features). See details below for some pre-built gene set matrices.

`factors` character vector with the factor names, or numeric vector with the index of the factors for which to perform the enrichment.

`set.statistic` the set statisic computed from the feature statistics. Must be one of the following: "mean.diff" (default) or "rank.sum".

`statistical.test` the statistical test used to compute the significance of the feature set statistics under a competitive null hypothesis. Must be one of the following: "parametric" (default), "cor.adj.parametric", "permutation".

`sign` use only "positive" or "negative" weights. Default is "all".

`min.size` Minimum size of a feature set (default is 10).

`nperm` number of permutations. Only relevant if statistical.test is set to "permutation". Default is 1000

`p.adj.method` Method to adjust p-values factor-wise for multiple testing. Can be any method in p.adjust.methods(). Default uses Benjamini-Hochberg procedure.

`alpha` FDR threshold to generate lists of significant pathways. Default is 0.1

`verbose` boolean indicating whether to print messages on progress

Details

The aim of this function is to relate each factor to pre-defined biological pathways by performing a gene set enrichment analysis on the feature weights.
This function is particularly useful when a factor is difficult to characterise based only on the genes with the highest weight.
We provide a few pre-built gene set matrices in the MOFAdata package. See `https://github.com/bioFAM/MOFAdata` for details.
The function we implemented is based on the `pcgse` function with some modifications. Please read this paper https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4543476 for details on the math.

Value

a list with five elements:

`\strong{pval}:` matrices with nominal p-values.

`\strong{pval.adj}:` matrices with FDR-adjusted p-values.

`\strong{feature.statistics}:` matrices with the local (feature-wise) statistics.

`\strong{set.statistics}:` matrices with the global (gene set-wise) statistics.

`\strong{sigPathways}` list with significant pathways per factor.

run_mofa *Train a MOFA model*

Description

Function to train an untrained MOFA object.

Usage

```
run_mofa(object, outfile = NULL, save_data = TRUE, use_basilisk = FALSE)
```

Arguments

object	an untrained MOFA object
outfile	output file for the model (.hdf5 format). If NULL, a temporary file is created.
save_data	logical indicating whether to save the training data in the hdf5 file. This is useful for some downstream analysis (mainly functions with the prefix plot_data), but it can take a lot of disk space.
use_basilisk	use basilisk to automatically install a conda environment with mofapy2 and all dependencies? If FALSE (default), you should specify the right python binary when loading R with reticulate::use_python(..., force=TRUE) or the right conda environment with reticulate::use_condaenv(..., force=TRUE).

Details

This function is called once a MOFA object has been prepared (using prepare_mofa) In this step the R package calls the mofapy2 Python package, where model training is performed.
The interface with Python is done with the reticulate package. If you have several versions of Python installed and R is not detecting the correct one, you can change it using reticulate::use_python when loading the R session. Alternatively, you can let us install mofapy2 for you using basilisk if you set use_basilisk to TRUE

Value

a trained MOFA object

Examples

```
# Load data (in data.frame format)
file <- system.file("extdata", "test_data.RData", package = "MOFA2")
load(file)

# Create the MOFA object
MOFAmodel <- create_mofa(dt)

# Prepare the MOFA object with default options
MOFAmodel <- prepare_mofa(MOFAmodel)
```

```
# Run the MOFA model
## Not run:  MOFAmodel <- run_mofa(MOFAmodel, use_basilisk = TRUE)
```

`run_tsne` *Run t-SNE on the MOFA factors*

Description

Run t-SNE on the MOFA factors

Usage

```
run_tsne(object, factors = "all", groups = "all", ...)
```

Arguments

`object`	a trained `MOFA` object.
`factors`	character vector with the factor names, or numeric vector with the indices of the factors to use, or "all" to use all factors (default).
`groups`	character vector with the groups names, or numeric vector with the indices of the groups of samples to use, or "all" to use all groups (default).
`...`	arguments passed to `Rtsne`

Details

This function calls `Rtsne` to calculate a TSNE representation from the MOFA factors. Subsequently, you can plot the TSNE representation with `plot_dimred` or fetch the coordinates using `plot_dimred(..., method="TSNE", return_data=TRUE)`. Remember to use set.seed before the function call to get reproducible results.

Value

Returns a `MOFA` object with the `MOFAobject@dim_red` slot filled with the t-SNE output

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Run
## Not run:  model <- run_tsne(model, perplexity = 15)

# Plot
## Not run:  model <- plot_dimred(model, method="TSNE")

# Fetch data
## Not run:  tsne.df <- plot_dimred(model, method="TSNE", return_data=TRUE)
```

run_umap *Run UMAP on the MOFA factors*

Description

Run UMAP on the MOFA factors

Usage

```
run_umap(
  object,
  factors = "all",
  groups = "all",
  n_neighbors = 30,
  min_dist = 0.3,
  metric = "cosine",
  ...
)
```

Arguments

object	a trained MOFA object.
factors	character vector with the factor names, or numeric vector with the indices of the factors to use, or "all" to use all factors (default).
groups	character vector with the groups names, or numeric vector with the indices of the groups of samples to use, or "all" to use all groups (default).
n_neighbors	number of neighboring points used in local approximations of manifold structure. Larger values will result in more global structure being preserved at the loss of detailed local structure. In general this parameter should often be in the range 5 to 50.
min_dist	This controls how tightly the embedding is allowed compress points together. Larger values ensure embedded points are moreevenly distributed, while smaller values allow the algorithm to optimise more accurately with regard to local structure. Sensible values are in the range 0.01 to 0.5
metric	choice of metric used to measure distance in the input space
...	arguments passed to umap

Details

This function calls umap to calculate a UMAP representation from the MOFA factors For details on the hyperparameters of UMAP see the documentation of umap. Subsequently, you can plot the UMAP representation with plot_dimred or fetch the coordinates using plot_dimred(..., method="UMAP", return_data=TRUE). Remember to use set.seed before the function call to get reproducible results.

Value

Returns a MOFA object with the MOFAobject@dim_red slot filled with the UMAP output

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Change hyperparameters passed to umap
## Not run:  model <- run_umap(model, min_dist = 0.01, n_neighbors = 10)
# Plot
## Not run:  model <- plot_dimred(model, method="UMAP")

# Fetch data
## Not run:  umap.df <- plot_dimred(model, method="UMAP", return_data=TRUE)
```

---

samples_metadata *samples_metadata: retrieve sample metadata*

Description

samples_metadata: retrieve sample metadata

Usage

```
samples_metadata(object)

samples_metadata(object) <- value

## S4 method for signature 'MOFA'
samples_metadata(object)

## S4 replacement method for signature 'MOFA,data.frame'
samples_metadata(object) <- value
```

Arguments

object	a MOFA object.
value	data frame with sample metadata, it must at least contain the columns sample and group. The order of the rows must match the order of samples_names(object)

Value

a data frame with sample metadata

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
samples_metadata(model)
```

---

samples_names *samples_names: set and retrieve sample names*

Description

samples_names: set and retrieve sample names

Usage

```
samples_names(object)

samples_names(object) <- value

## S4 method for signature 'MOFA'
samples_names(object)

## S4 replacement method for signature 'MOFA,list'
samples_names(object) <- value
```

Arguments

object	a MOFA object.
value	list of character vectors with the sample names for every group

Value

list of character vectors with the sample names for each group

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
samples_names(model)
```

select_model — *Select a model from a list of trained MOFA objects based on the best ELBO value*

Description

Different objects of MOFA are compared in terms of the final value of the ELBO statistics and the model with the highest ELBO value is selected.

Usage

```
select_model(models, plot = FALSE)
```

Arguments

models	a list containing MOFA objects.
plot	boolean indicating whether to show a plot of the ELBO for each model instance

Value

A MOFA object

set_covariates — *Add covariates to a MOFA model*

Description

Function to add continuous covariate(s) to a MOFA object for training with MEFISTO

Usage

```
set_covariates(object, covariates)
```

Arguments

object	an untrained MOFA
covariates	Sample-covariates to be passed to the model. This can be either: • a character, specifying columns already present in the samples_metadata of the object • a data.frame with columns "sample", "covariate", "value". Sample names need to match those present in the data • a matrix with smaples in columns and covariate(s) in row(s) Note that the covariate should be numeric and continuous.

Details

To activate the functional MEFISTO framework, specify mefisto_options when preparing the training using `prepare_mofa`

Value

Returns an untrained `MOFA` with covariates filled in the corresponding slots

Examples

```
#' # Simulate data
dd <- make_example_data(sample_cov = seq(0,1,length.out = 100), n_samples = 100, n_factors = 4)

# Create MOFA object
sm <- create_mofa(data = dd$data)

# Add a covariate
sm <- set_covariates(sm, covariates = dd$sample_cov)
sm
```

`subset_factors` *Subset factors*

Description

Method to subset (or sort) factors

Usage

```
subset_factors(object, factors, recalculate_variance_explained = TRUE)
```

Arguments

`object`	a `MOFA` object.
`factors`	character vector with the factor names, or numeric vector with the index of the factors.
`recalculate_variance_explained`	logical indicating whether to recalculate variance explained values. Default is `TRUE`.

Value

A `MOFA` object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Subset factors 1 to 3
model <- subset_factors(model, factors = 1:3)
```

`subset_features` *Subset features*

Description

Method to subset (or sort) features

Usage

```
subset_features(object, view, features)
```

Arguments

`object`	a `MOFA` object.
`view`	character vector with the view name or integer with the view index
`features`	character vector with the sample names, numeric vector with the feature indices or logical vector with the samples to be kept as TRUE.

Value

A `MOFA` object

`subset_groups` *Subset groups*

Description

Method to subset (or sort) groups

Usage

```
subset_groups(object, groups)
```

Arguments

`object`	a `MOFA` object.
`groups`	character vector with the groups names, numeric vector with the groups indices or logical vector with the groups to be kept as TRUE.

Value

A MOFA object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Subset the first group
model <- subset_groups(model, groups = 1)
```

---

subset_samples *Subset samples*

Description

Method to subset (or sort) samples

Usage

```
subset_samples(object, samples)
```

Arguments

object	a MOFA object.
samples	character vector with the sample names or numeric vector with the sample indices.

Value

A MOFA object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# (TO-DO) Remove a specific sample from the model (an outlier)
```

subset_views *Subset views*

Description

Method to subset (or sort) views

Usage

```
subset_views(object, views)
```

Arguments

object	a MOFA object.
views	character vector with the views names, numeric vector with the views indices, or logical vector with the views to be kept as TRUE.

Value

A MOFA object

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)

# Subset the first view
model <- subset_views(model, views = 1)
```

summarise_factors *Summarise factor values using external groups*

Description

Function to summarise factor values using a discrete grouping of samples.

Usage

```
summarise_factors(
  object,
  df,
  factors = "all",
  groups = "all",
  abs = FALSE,
  return_data = FALSE
)
```

Arguments

object	a trained MOFA object.
df	a data.frame with the columns "sample" and "level", where level is a factor with discrete group assigments for each sample.
factors	character vector with the factor name(s), or numeric vector with the index of the factor(s) to use. Default is 'all'.
groups	character vector with the groups names, or numeric vector with the indices of the groups of samples to use, or "all" to use samples from all groups.
abs	logical indicating whether to take the absolute value of the factors (default is FALSE).
return_data	logical indicating whether to return the fa instead of plotting

Value

A ggplot object or a data.frame if return_data is TRUE

---

views_names *views_names: set and retrieve view names*

Description

views_names: set and retrieve view names

Usage

```
views_names(object)

views_names(object) <- value

## S4 method for signature 'MOFA'
views_names(object)

## S4 replacement method for signature 'MOFA,character'
views_names(object) <- value
```

Arguments

object	a MOFA object.
value	character vector with the names for each view

Value

character vector with the names for each view

Examples

```
# Using an existing trained model on simulated data
file <- system.file("extdata", "model.hdf5", package = "MOFA2")
model <- load_model(file)
views_names(model)
views_names(model) <- c("viewA", "viewB")
```

`%>%` *Re-exporting the pipe operator See* `magrittr::Rpercent>Rpercent` *for details.*

Description

Re-exporting the pipe operator See `magrittr::%>%` for details.

Usage

```
lhs %>% rhs
```

Arguments

`lhs`	see `magrittr::%>%`
`rhs`	see `magrittr::%>%`

Value

depending on lhs and rhs

Index