Package 'MOFA'

October 17, 2020

Type Package

Title Multi-Omics Factor Analysis (MOFA)

Version 1.4.0

Maintainer Britta Velten <britta.velten@gmail.com>

Author
Ricard Argelaguet, Britta Velten, Damien Arnol, Florian Buettner, Wolfgang Huber, Oliver Stegle

Date 2018-06-26

License LGPL-3 | file LICENSE

Description Multi-Omics Factor Analysis: an unsupervised framework for the integration of multi-omics data sets.

Encoding UTF-8

Depends R (>= 3.5)

Imports rhdf5, dplyr, reshape2, pheatmap, corrplot, ggplot2, ggbeeswarm, methods, scales, GGally, RColorBrewer, cowplot, ggrepel, MultiAssayExperiment, Biobase, doParallel, foreach, reticulate, grDevices, stats, utils

Suggests knitr, MOFAdata

biocViews DimensionReduction, Bayesian, Visualization

VignetteBuilder knitr

LazyData false

NeedsCompilation yes

RoxygenNote 6.1.1

SystemRequirements Python (>=2.7.0), numpy, pandas, h5py, scipy, sklearn, mofapy

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

git_branch RELEASE_3_11

git_last_commit baf2eef

git_last_commit_date 2020-04-27

Date/Publication 2020-10-16

R topics documented:

`calculateVarianceExplained` *Calculate variance explained by the model*

Description

Method to calculate variance explained by the MOFA model for each view and latent factor.
As a measure of variance explained for gaussian data we adopt the coefficient of determination (R2).
For non-gaussian views the calculations are based on the normally-distributed pseudo-data (for more information on the non-gaussian model see Supplementary Methods of the MOFA paper or Seeger & Bouchard, 2012).

Usage

```
calculateVarianceExplained(object, views = "all", factors = "all")
```

Arguments

`object`	a trained `MOFAmodel` 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 takes a trained MOFA model as input and calculates for each view the coefficient of determination (R2), i.e. the proportion of variance in the data explained by the MOFA factor(s) (both jointly and for each individual factor). In case of non-Gaussian data the variance explained on the Gaussian pseudo-data is calculated.

Value

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

Examples

```
# Using an existing trained model on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
calculateVarianceExplained(MOFA_CLL)

# Using an existing trained model on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
calculateVarianceExplained(MOFA_scMT)
```

`clusterSamples` *clusterSamples: 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, similar to the iCluster model (Shen, 2009).
The clustering can be performed in a single factor, which is equivalent to setting a manual threshold; or 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

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

Arguments

`object`	a trained `MOFAmodel` object.
`k`	number of clusters
`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, samples can have missing values in the factor space. This occurs when a factor is active in a single view and some samples are missing this data.
In such a case, there are several strategies to follow:

- Use clustering approaches that deal with NAs (not implemented in MOFA)
- If the factor in question is not important, you can remove it with `subsetFactors`
- If the factor in question is important and just a small number of samples are conflictive, you can manually set them to 0 using `object@Expectations$Z[is.na(object@Expectations$Z)] <-0`

By default, the conflictive samples are ignored in the clustering procedure and NAs are returned.

Value

output from kmeans function

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
# cluster samples based into 3 groups based on all factors
clusterSamples(MOFA_CLL, k=3, factors="all")
# cluster samples based into 2 groups based on factor 1
clusters <- clusterSamples(MOFA_CLL, k=2, factors=1)
# cluster can be visualized for example on the factors values:
plotFactorBeeswarm(MOFA_CLL, factor=1, color_by=clusters)

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
# cluster samples based into 2 groups based on all factor 1 and 2
clusters <- clusterSamples(MOFA_CLL, k=2, factors=1:2)
# cluster can be visualized for example on the factors values:
plotFactorScatter(MOFA_CLL, factors=1:2, color_by=clusters)
```

---

compareFactors *Correlation of the latent factors across different trials*

Description

Different objects of MOFAmodel are compared in terms of correlation between their latent factors. The correlation is calculated only on those samples which are present in all models. Ideally, the output should look like a block diagonal matrix, suggesting that all detected factors are robust under different initializations. If not, it suggests that some factors are weak and not captured by all models.

Usage

```
compareFactors(models, comparison = "all", show_rownames = FALSE,
  show_colnames = FALSE, ...)
```

Arguments

models	a list containing MOFAmodel objects.
comparison	type of comparison, either 'pairwise', i.e. compare one model with another one at a time, or 'all', i.e. calculate correlation between factors from all model. By default, all models are compared.
show_rownames	logical indicating whether to show rownames in heatmap (see also pheatmap documentation)
show_colnames	logical indicating whether to show colnames in heatmap (see also pheatmap documentation)
...	extra arguments passed to pheatmap

Details

This function can be helpful to evaluate the robustness of factors across different random initilizations. Large block of factors from different models in the correlation matrix show consistent factors, while stand-alone factors that are only recovered in a single model instance are less reliable.

Value

Plots a heatmap of correlation of Latent Factors in all models when 'comparison' is 'all'. Otherwise, for each pair of models, a seperate heatmap is produced comparing one model againt the other. The corresponding correlation matrix or list or pairwise correlation matrices is returned

Examples

```
### Example on simulated data
# Simulate Data
data <- makeExampleData()
# Create MOFA model
MOFAobject <- createMOFAobject(data)
# Prepare MOFA model
MOFAobject <- prepareMOFA(MOFAobject)
# Train several instances of MOFA models
n_inits <- 3
MOFAlist <- lapply(seq_len(n_inits), function(i) runMOFA(MOFAobject, outfile=tempfile()))
compareFactors(MOFAlist, comparison="all")
compareFactors(MOFAlist, comparison="pairwise")
```

---

compareModels *Compare different instances of trained* MOFAmodel

---

Description

Different objects of MOFAmodel are compared in terms of the final value of the ELBO statistics. For model selection the model with the highest ELBO value is selected. The height of the bar indicates the number of inferred factors and the color of the bar the value of the ELBO statistic.

Usage

```
compareModels(models, show_modelnames = FALSE)
```

Arguments

models	a list containing MOFAmodel objects.
show_modelnames	boolean, whether to indicate the name of each model instance (names of the list in models) or not

Value

a ggplot showing the number of factors and the ELBO statistics of the given models as a barplot

Examples

```
### Example on simulated data
# Simulate Data
data <- makeExampleData()
# Create MOFA model
MOFAobject <- createMOFAobject(data)
# Prepare MOFA model
MOFAobject <- prepareMOFA(MOFAobject)
# Train several instances of MOFA models
n_inits <- 3
MOFAlist <- lapply(seq_len(n_inits), function(i) runMOFA(MOFAobject, outfile=tempfile()))
compareModels(MOFAlist)
```

---

createMOFAobject *Initialize a MOFA object*

Description

Method to initialize a MOFAmodel object with a multi-omics data set.

Usage

```
createMOFAobject(data)
```

Arguments

data	either a MultiAssayExperiment or a list of matrices with features as rows and samples as columns.

Details

If the multi-omics data is provided as a list of matrices, please make sure that features are stored as rows and samples are stored as columns.
If the matrices have sample names, we will use them to match the different matrices, filling the corresponding missing values.
If matrices have no column names, all matrices must have the same number of columns, and you are responsible for filling any missing values.

Value

Returns an untrained MOFAmodel object.
Next step is to define the training, model and data processing options (see prepareMOFA)

Examples

```
# Option 1: Create a MOFAobject from a list of matrices, features in rows and samples in columns.
data("CLL_data", package = "MOFAdata")
MOFAobject <- createMOFAobject(CLL_data)

# Option 2: Create a MOFAobject from a MultiAssayExperiment
library(MultiAssayExperiment)
data("CLL_data", package = "MOFAdata")
data("CLL_covariates", package = "MOFAdata")
```

```
mae_CLL <- MultiAssayExperiment(experiments = CLL_data, colData = CLL_covariates)
MOFAobject <- createMOFAobject(mae_CLL)

# next, this object can be passed to prepareMOFA and runMOFA
# (training in runMOFA can take some time):
## Not run:
# MOFAobject <- prepareMOFA(MOFAobject)
# MOFAobject <- runMOFA(MOFAobject)

## End(Not run)
```

DataOptions *DataOptions: set and retrieve data options*

Description

Function to set and retrieve data options.

Usage

```
DataOptions(object)

DataOptions(object) <- value

## S4 method for signature 'MOFAmodel'
DataOptions(object)

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

Arguments

object	a MOFAmodel object.
value	a list with data options

Value

list of data options

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
DataOptions(MOFAobject)
```

Dimensions *Dimensions: set and retrieve dimensions*

Description

Function to set and retrieve model dimensions.

Usage

```
Dimensions(object)

Dimensions(object) <- value

## S4 method for signature 'MOFAmodel'
Dimensions(object)

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

Arguments

object	a MOFAmodel object.
value	list of dimensions

Value

list of dimensions of the model

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
Dimensions(MOFAobject)
```

Expectations *Expectations: set and retrieve expectations of model components*

Description

Function to set and retrieve expectations of model components.

Usage

```
Expectations(object)

Expectations(object) <- value

## S4 method for signature 'MOFAmodel'
Expectations(object)
```

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

Arguments

object	a MOFAmodel object.
value	a list with matrices for expectations of unobserved model components

Value

list of matrices containing expectations of model components

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
names(Expectations(MOFAobject))
```

---

factorNames *factorNames: set and retrieve factor names*

Description

Function to set and retrieve factor names.

Usage

```
factorNames(object)

factorNames(object) <- value

## S4 method for signature 'MOFAmodel'
factorNames(object)

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

Arguments

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

Value

character vector with the features names

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
factorNames(MOFAobject)
factorNames(MOFAobject) <- paste("Factor",1:3,sep="_")
factorNames(MOFAobject)
```

`FeatureIntercepts` *FeatureIntercepts: set and retrieve feature intercepts*

Description

Function to set and retrieve feature intercepts

Usage

```
FeatureIntercepts(object)

FeatureIntercepts(object) <- value

## S4 method for signature 'MOFAmodel'
FeatureIntercepts(object)

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

Arguments

`object`	a `MOFAmodel` object.
`value`	list of feature intercepts (per view)

Value

list of feature intercepts (per view)

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
str(FeatureIntercepts(MOFAobject))
```

featureNames *featureNames: set and retrieve feature names*

Description

Function to set and retrieve feature names.

Usage

```
featureNames(object)

featureNames(object) <- value

## S4 method for signature 'MOFAmodel'
featureNames(object)

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

Arguments

object	a MOFAmodel object.
value	list of character vectors with the feature names for each view

Value

list of character vectors with the feature names for each view

Examples

```
data("CLL_data", package = "MOFAdata")
MOFAobject  <- createMOFAobject(CLL_data)
featureNames(MOFAobject)$Mutations
head(featureNames(MOFAobject)$Drugs)
```

getCovariates *getCovariates*

Description

This function extracts covariates from the colData in the input MultiAssayExperiment object. Note that if you did not use MultiAssayExperiment to create your createMOFAobject, this function will not work.

Usage

```
getCovariates(object, covariate)
```

Arguments

object	a MOFAmodel object.
covariate	names of the covariate

Value

a vector containing the covariate

Examples

```
# Example on the CLL data
library(MultiAssayExperiment)
data("CLL_data", package = "MOFAdata")
data("CLL_covariates", package = "MOFAdata")
# Create MultiAssayExperiment object
mae_CLL <- MultiAssayExperiment(CLL_data, colData=CLL_covariates)
MOFAobject  <- createMOFAobject(mae_CLL)
# Extract covariates from the colData of a MultiAssayExperiment
gender <- getCovariates(MOFAobject, "Gender")
diagnosis <- getCovariates(MOFAobject, "Diagnosis")
# Example on the scMT data
data("scMT_data", package = "MOFAdata")
MOFAobject  <- createMOFAobject(scMT_data)
# Extract covariates from the colData of a MultiAssayExperiment
culture <- getCovariates(MOFAobject, "culture")
# Extract covariates from the phenoData of the RNA assay
cdr <- getCovariates(MOFAobject, "cellular_detection_rate")
```

---

getDefaultDataOptions *Get default data options*

Description

Function to obtain the default data options.

Usage

```
getDefaultDataOptions()
```

Details

The data options are the following:

- **scaleViews**: logical indicating whether to scale views to have the same unit variance. As long as the scale differences between the data sets is not too high, this is not required. Default is FALSE.
- **removeIncompleteSamples**: logical indicating whether to remove samples that are not profiled in all omics. We recommend this only for testing, as the model can cope with samples having missing assays. Default is FALSE.

Value

Returns a list with the default data options, which have to be passed as an argument to prepareMOFA

Examples

```
DataOptions <- getDefaultDataOptions()
DataOptions
```

`getDefaultModelOptions` *Get default model options*

Description

Function to obtain the default model options.

Usage

```
getDefaultModelOptions(object)
```

Arguments

`object`	an untrained MOFAmodel object

Details

The model options are the following:

- **likelihood**: character vector with data likelihoods per view: 'gaussian' for continuous data, 'bernoulli' for binary data and 'poisson' for count data. By default, they are guessed internally.
- **numFactors**: numeric value indicating the initial number of factors. If you want to learn the number of factors automatically we recommend setting this to a large value, default is 25.
- **sparsity**: logical indicating whether to use sparsity. This is always recommended, as it will make the loadings more interpretable. Default is TRUE.

Value

Returns a list with the default model options, which have to be passed as an argument to prepareMOFA

Examples

```
# load data
data("CLL_data", package = "MOFAdata")
#create a MOFAmodel object
MOFAobject <- createMOFAobject(CLL_data)
# set model options
ModelOptions <- getDefaultModelOptions(MOFAobject)
ModelOptions
```

getDefaultTrainOptions *Get default training options*

Description

Function to obtain the default training options.

Usage

```
getDefaultTrainOptions()
```

Details

The training options are the following:

- **maxiter**: numeric value indicating the maximum number of iterations. Default is 5000, but we recommend using the 'tolerance' as convergence criteria.
- **tolerance**: numeric value indicating the convergence threshold based on the change in Evidence Lower Bound (deltaELBO). For quick exploration we recommend this to be around 1.0, and for a thorough training we recommend a value of 0.01. Default is 0.1
- **DropFactorThreshold**: numeric hyperparamter to automatically learn the number of factors. It indicates the threshold on fraction of variance explained to consider a factor inactive and automatically drop it from the model during training. For example, a value of 0.01 implies that factors explaining less than 1% of variance (in each view) will be dropped. Default is 0, which implies that only factors that explain no variance at all will be removed
- **verbose**: logical indicating whether to generate a verbose output.
- **seed**: random seed for reproducibility (default is NULL, which samples a random seed).

Value

Returns a list with default training options, which have to be passed as an argument to prepareMOFA

Examples

```
TrainOptions <- getDefaultTrainOptions()
TrainOptions
```

getDimensions *getDimensions*

Description

Extract dimensionalities from the model.

Usage

```
getDimensions(object)
```

Arguments

object	a MOFAmodel object.

Details

K indicates the number of factors, D indicates the number of features, N indicates the (total) number of samples and M indicates the number of views.

Value

list with the relevant dimensionalities of the model. N for the number of samples, M for the number of views, D for the number of features of each view and K for the number of infered latent factors.

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
# get dimensions
getDimensions(MOFAobject)
```

---

getELBO *getELBO*

Description

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

Usage

```
getELBO(object)
```

Arguments

object	a MOFAmodel object.

Value

value of the ELBO statistic at end of training

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
# get ELBO statistic
getELBO(MOFAobject)
```

getExpectations *getExpectations*

Description

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

Usage

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

Arguments

object	a trained MOFAmodel object.
variable	variable name: 'Z' for factors, 'W' for weights, 'Tau' for noise, 'Y' for pseudodata, 'Theta' for feature-wise spike-and-slab sparsity, 'Alpha' for view and factor-wise ARD sparsity
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)
- "Y": a list of length (views) where each element is a matrix with dimensions (features,samples). If as.data.frame is TRUE, a long-formatted data frame with columns (view,feature,sample,value)
- "Theta": a list of length (views) where each element is a vector of containing the values for each factor
- "Alpha": a list of length (views) where each element is a vector of containing the values for each factor
- "Tau": a list of length (views) where each element is a matrix with dimensions (samples, features)

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
# get expectations of Alpha as matrix
getExpectations(MOFAobject, variable="Alpha")
```

`getFactors` *getFactors*

Description

Extract the latent factors from the model.

Usage

```
getFactors(object, factors = "all", as.data.frame = FALSE)
```

Arguments

`object`	a trained `MOFAmodel` object.
`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

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

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
# get factors as matrix
getFactors(MOFAobject, factors = 1:3)
# get factors as data.frame
head(getFactors(MOFAobject, factors = 1:5, as.data.frame = TRUE))
```

getImputedData *getImputedData*

Description

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

Usage

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

Arguments

`object`	a trained `MOFAmodel` object.
`views`	character vector with the view name(s), or numeric vector with the view 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`.

Value

By default returns a list where each element is a matrix with dimensionality (D,N), where D is the number of features in this view and N is the number of samples.
Alternatively, if `as.data.frame` is `TRUE`, returns a long-formatted data frame with columns (view,feature,sample,value).

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
# impute missing values
MOFAobject <- impute(MOFAobject)
# get imputations for a single view
imputedDrugs <- getImputedData(MOFAobject,view="Drugs")
head(imputedDrugs)
```

---

getTrainData *getTrainData*

Description

Fetch the training data

Usage

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

Arguments

object	a MOFAmodel object.
views	character vector with the view name(s), or numeric vector with the view 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 data frame instead of a list of matrices. Default is FALSE.

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).

Value

A list with one numeric matrix per view, containing the parsed data used to fit the MOFA model.

Examples

```
data("scMT_data", package = "MOFAdata")
MOFAobject <- createMOFAobject(scMT_data)
trainData_scMT <- getTrainData(MOFAobject, as.data.frame = FALSE)
(trainData_scMT[["RNA expression"]])[1:10,1:10]

data("CLL_data", package = "MOFAdata")
MOFAobject <- createMOFAobject(CLL_data)
trainData_CLL <- getTrainData(MOFAobject, as.data.frame = TRUE)
head(trainData_CLL)
```

---

getWeights *getWeights*

Description

Extract the weights from the model.

Usage

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

Arguments

object	a trained MOFAmodel 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".
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

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
# get weights as a list of matrices
weightList <- getWeights(MOFAobject, view = "all", factors = 1:4)
# get weights as a data.frame
head(getWeights(MOFAobject, view = "Mutations", as.data.frame = TRUE))
```

---

impute *Impute missing values from a fitted MOFA model*

Description

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

Usage

```
impute(object, views = "all", factors = "all", type = c("inRange",
  "response", "link"))
```

Arguments

object	a MOFAmodel object.
views	character vector with the view names, or numeric vector with view indexes.
factors	character vector with the factor names, or numeric vector with the factor indexes.
type	type of prediction returned, either: • **response**: gives the response vector, the mean for Gaussian and Poisson, and success probabilities for Bernoulli. • **link**: gives the linear predictions. • **inRange**: rounds the fitted values of integer-valued distributions (Poisson and Bernoulli) to the next integer. This is the default option.

Details

Matrix factorization models generate a denoised and condensed low-dimensional representation of the data which capture the main sources of heterogeneity of the data. Such representation can be used to do predictions via the equation `Y = WZ`.
This method fills the `ImputedData` slot by replacing the missing values in the training data with the model predictions.
For more details see the Methods section of the MOFA article.

Value

a `MOFAmodel` object with imputed data in the `ImputedData` slot

Examples

```
# Load CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
# impute missing data in all views using all factors
MOFA_CLL <- impute(MOFA_CLL)

# Load scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
# impute missing data in the RNA view using Factor 1
MOFA_scMT <- impute(MOFA_scMT, views="RNA expression", factors="LF1")
```

`ImputedData` *ImputedData: set and retrieve imputed data*

Description

Function to set and retrieve imputed data.

Usage

```
ImputedData(object)

ImputedData(object) <- value

## S4 method for signature 'MOFAmodel'
ImputedData(object)

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

Arguments

`object`	a `MOFAmodel` object.
`value`	a list of matrices containing the imputed data

Value

list of matrices containing the imputed data

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
MOFAobject <- impute(MOFAobject)
str(ImputedData(MOFAobject))
```

`InputData` *InputData: set and retrieve input data*

Description

Function to set and retrieve input data.

Usage

```
InputData(object)

InputData(object) <- value

## S4 method for signature 'MOFAmodel'
InputData(object)

## S4 replacement method for signature 'MOFAmodel,MultiAssayExperiment'
InputData(object) <- value
```

Arguments

`object`	a `MOFAmodel` object.
`value`	a multi-assay experiment

Value

input data as a multi-assay experiment

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
InputData(MOFAobject)
```

loadModel *loading a trained MOFA model*

Description

Method to load a trained MOFA model.
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

```
loadModel(file, object = NULL, sortFactors = TRUE, minR2 = 0.01)
```

Arguments

file	an hdf5 file saved by the MOFA python framework.
object	either NULL (default) or an an existing untrained MOFA object. If NULL, the MOFAmodel object is created from the scratch.
sortFactors	boolean indicating whether factors should be sorted by variance explained (default is TRUE)
minR2	minimum R2 threshold to call 'active' factors (default is 0.01).

Value

a MOFAmodel model.

Examples

```
# path to the hdf5 file
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
# load the model
MOFAobject <- loadModel(filepath)
```

makeExampleData *make an example multi-view data set for illustration of MOFA*

Description

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

Usage

```
makeExampleData(n_views = 3, n_features = 100, n_samples = 50,
  n_factors = 5, likelihood = "gaussian")
```

Arguments

n_views	number of views
n_features	number of features in each view
n_samples	number of samples
n_factors	number of factors
likelihood	likelihood for each view, one of "gaussian", "bernoulli", "poisson", or a character vector of length n_views

Value

Returns an untrained MOFAmodel containing simulated data as training data.

Examples

```
# Generate a data set
MOFAexample <- makeExampleData()
str(MOFAexample)
```

ModelOptions *ModelOptions: set and retrieve model options*

Description

Function to set and retrieve model options.

Usage

```
ModelOptions(object)

ModelOptions(object) <- value

## S4 method for signature 'MOFAmodel'
ModelOptions(object)

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

Arguments

object	a MOFAmodel object.
value	a list with model options

Value

list of model options

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
ModelOptions(MOFAobject)
```

`MOFA` *Multi-Omics Factor Analysis (MOFA)*

Description

MOFA provides an unsupervised framework for the integration of multi-omics data sets. Given several data matrices with measurements of multiple 'omics data types on the same or on overlapping sets of samples, MOFA infers an interpretable low-dimensional data representation in terms of (hidden) factors. These learnt factors represent the driving sources of variation across data modalities, thus facilitating the identification of cellular states or disease subgroups.

The package contains all function required for training MOFA on a multi-omics data set as well as for different downstream analyes, such as visualisation of samples in factor space, annotation of factors to molecular markers or gene sets, outlier identification and imputation of missing values.

Details

Please have a look at the vignette "MOFA" for a in-depth introduction to the package.

`MOFAmodel` *Class to store a Multi-Omics Factor Analysis (MOFA) model*

Description

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

Slots

`InputData` the input data before being parsed to Training Data. Either a MultiAssayExperiment object or a list of matrices, one per view.

`TrainData` the parsed data used to fit the MOFA model A list with one matrix per view.

`ImputedData` the parsed data with the missing values imputed using the MOFA model. A list with one matrix per view.

`Expectations` expected values of the different variables of the model. A list of matrices, one per variable. The most relevant are "W" for weights and "Z" for factors.

`TrainStats` list with training statistics such as evidence lower bound (ELBO), number of active factors, etc.

`DataOptions` list with the data processing options such as whether to center or scale the data.

`TrainOptions` list with the training options such as maximum number of iterations, tolerance for convergence, etc.

`ModelOptions` list with the model options such as likelihoods, number of factors, etc.

`FeatureIntercepts` list with the feature-wise intercepts. Only used internally.

`Dimensions` list with the relevant dimensionalities of the model. N for the number of samples, M for the number of views, D for the number of features of each view and K for the number of infered latent factors.

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

`plotDataHeatmap` *Plot heatmap of relevant features*

Description

Function to plot a heatmap of the input data for relevant features, usually the ones with highest loadings in a given factor.

Usage

```
plotDataHeatmap(object, view, factor, features = 50,
  includeWeights = FALSE, transpose = FALSE, imputed = FALSE, ...)
```

Arguments

`object`	a `MOFAmodel` object.
`view`	character vector with the view name, or numeric vector with the index of the view.
`factor`	character vector with the factor name, or numeric vector with the index of the factor.
`features`	if an integer, the total number of top features to plot, based on the absolute value of the loading. If a character vector, a set of manually-defined features. Default is 50.
`includeWeights`	logical indicating whether to include the weight of each feature as an extra annotation in the heatmap. Default is FALSE.
`transpose`	logical indicating whether to transpose the output 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.
`...`	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 loadings, using for example `plotWeights` or `plotTopWeights`. These functions display the top features that are driving the heterogeneity captured by a factor.
However, one might also be interested in visualising the coordinated heterogeneity in the input data, rather than looking at "abstract" weights.
This function extracts the top features for a given factor and view, and generates a heatmap with dimensions (samples,features). This should reveal the underlying heterogeneity that is captured by the latent factor.
A similar function for doing scatterplots rather than heatmaps is `plotDataScatter`.

Value

plots a heatmap of the data for the top features for a given factor and views

Examples

```
# Load CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
# plot top 30 features on Factor 1 in the mRNA view
plotDataHeatmap(MOFA_CLL, view="mRNA", factor=1, features=30)
# without column names (extra arguments passed to pheatmap)
plotDataHeatmap(MOFA_CLL, view="mRNA", factor=1, features=30, show_colnames = FALSE)
# transpose the heatmap
plotDataHeatmap(MOFA_CLL, view="mRNA", factor=1, features=30, transpose=TRUE)
# do not cluster rows (extra arguments passed to pheatmap)
plotDataHeatmap(MOFA_CLL, view="mRNA", factor=1, features=30, cluster_rows=FALSE)
```

---

plotDataOverview *Plot overview of the input data*

Description

Function to do a tile plot showing the dimensionality and the missing value structure of the multi-omics data.

Usage

```
plotDataOverview(object, colors = NULL)
```

Arguments

object	a MOFAmodel object.
colors	a character vector specifying the colors per view. NULL (default) uses an internal palette.

Details

This function is helpful to get an overview of the dimensionality and the missing value structure of the training data.
It shows the number of samples, the number of views, the number of features, and the structure of missing values.
It is particularly useful to visualise incomplete data sets, where some samples are missing subsets of assays.

Value

a tile plot of the training data

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotDataOverview(MOFA_CLL)

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
# using customized colors
plotDataOverview(MOFA_scMT, colors= c("red", "red", "red", "blue"))
```

`plotDataScatter` *Scatterplots of feature values against latent factors*

Description

Function to do a scatterplot of the feature(s) values against the latent factor values.

Usage

```
plotDataScatter(object, view, factor, features = 10, color_by = NULL,
  name_color = "", shape_by = NULL, name_shape = "",
  showMissing = TRUE)
```

Arguments

`object`	a `MOFAmodel` object.
`view`	character vector with a view name, or numeric vector with the index of the view.
`factor`	character vector with a factor name, or numeric vector with the index of the factor.
`features`	if an integer, the total number of features to plot (10 by default). If a character vector, a set of manually-defined features.
`color_by`	specifies groups or values used to color the samples. This can be either: (a) a character giving the name of a feature, (b) a character giving the name of a covariate (only if using MultiAssayExperiment as input), or (c) a vector of the same length as the number of samples specifying discrete groups or continuous numeric values.
`name_color`	name for the color legend
`shape_by`	specifies groups or values used to shape the samples. This can be either: (a) a character giving the name of a feature present in the training data, (b) a character giving the name of a covariate (only if using MultiAssayExperiment as input), or (c) a vector of the same length as the number of samples specifying discrete groups.
`name_shape`	name for the shape legend
`showMissing`	logical indicating whether to show samples with missing values for the color or the shape. Default is TRUE.

Details

One of the first steps for the annotation of a given factor is to visualise the corresponding loadings, using for example plotWeights or plotTopWeights. These functions display the top features that are driving the heterogeneity captured by a factor.
However, one might also be interested in visualising the coordinated heterogeneity in the input data, rather than looking at "abstract" weights.
This function generates scatterplots of features against factors (each dot is a sample), so that you can observe the association between them.
A similar function for doing heatmaps rather than scatterplots is plotDataHeatmap.

Value

a scatterplot of featurea against a factor

Examples

```
# Load CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
# plot scatter for top 5 features on factor 1 in the view mRNA:
plotDataScatter(MOFA_CLL, view="mRNA", factor=1, features=5)
# coloring by the IGHV status (features in Mutations view), not showing samples with missing IGHV:
plotDataScatter(MOFA_CLL, view="mRNA", factor=1, features=5, color_by="IGHV", showMissing=FALSE)
```

---

plotEnrichment *Plot output of Feature Set Enrichment Analysis*

Description

Method to plot the results of the Feature Set Enrichment Analyisis (FSEA)

Usage

```
plotEnrichment(object, fsea.results, factor, alpha = 0.1,
  max.pathways = 25, adjust = TRUE)
```

Arguments

object	MOFAmodel object on which FSEA was performed
fsea.results	output of runEnrichmentAnalysis function
factor	string with factor name or numeric with factor index
alpha	p.value threshold to filter out feature sets
max.pathways	maximum number of enriched pathways to display
adjust	use adjusted p-values?

Value

a ggplot2 object

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)

# perform Enrichment Analysis on mRNA data using pre-build Reactome gene sets
data("reactomeGS", package = "MOFAdata")
fsea.results <- runEnrichmentAnalysis(MOFAobject, view="mRNA", feature.sets=reactomeGS)

# Plot top 10 enriched pathwyas on factor 5:
plotEnrichment(MOFAobject, fsea.results, factor=5,  max.pathways=10)
```

---

plotEnrichmentBars *Barplot of Feature Set Enrichment Analysis results*

Description

Method to generate a barplot with the number of enriched feature sets per factor

Usage

```
plotEnrichmentBars(fsea.results, alpha = 0.05)
```

Arguments

fsea.results	output of runEnrichmentAnalysis function
alpha	FDR threshold for calling enriched feature sets. Default is 0.05

Value

a ggplot2 object

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)

# perform Enrichment Analysis on mRNA data using pre-build Reactome gene sets
data("reactomeGS", package = "MOFAdata")
fsea.results <- runEnrichmentAnalysis(MOFAobject, view="mRNA", feature.sets=reactomeGS)

# Plot overview of number of enriched pathways per factor at an FDR of 1%
plotEnrichmentBars(fsea.results, alpha=0.01)
```

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

Description

Method to plot a detailed output of the Feature Set Enrichment Analysis (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 loading) The top genes with the highest statistic (max.genes argument) are displayed and labeled in black. The remaining genes are colored in grey.

Usage

```
plotEnrichmentDetailed(object, factor, feature.sets, fsea.results,
  adjust = TRUE, alpha = 0.1, max.genes = 5, max.pathways = 10,
  text_size = 3)
```

Arguments

`object`	`MOFAmodel` object on which FSEA was performed
`factor`	string with factor name or numeric with factor index
`feature.sets`	data structure that holds feature set membership information, as used in the runEnrichmentAnalysis function.
`fsea.results`	output of runEnrichmentAnalysis function
`adjust`	use adjusted p-values?
`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

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)

# perform Enrichment Analysis on mRNA data using pre-build Reactome gene sets
data("reactomeGS", package = "MOFAdata")
fsea.results <- runEnrichmentAnalysis(MOFAobject, view="mRNA", feature.sets=reactomeGS)

# Plot detailed output of the enrichment analysis results
plotEnrichmentDetailed(
object = MOFAobject,
factor = 5,
```

```
feature.sets = reactomeGS,
fsea.results = fsea.results,

)
```

`plotEnrichmentHeatmap` *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

```
plotEnrichmentHeatmap(fsea.results, alpha = 0.05, logScale = TRUE, ...)
```

Arguments

`fsea.results`	output of runEnrichmentAnalysis function
`alpha`	FDR threshold to filter out unsignificant feature sets which are not represented in the heatmap. Default is 0.05.
`logScale`	boolean indicating whether to plot the log of the p.values.
`...`	extra arguments to be passed to pheatmap function

Value

produces a heatmap

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)

# perform Enrichment Analysis on mRNA data using pre-build Reactome gene sets
data("reactomeGS", package = "MOFAdata")
fsea.results <- runEnrichmentAnalysis(MOFAobject, view="mRNA", feature.sets=reactomeGS)

# overview of enriched pathways per factor at an FDR of 1%
plotEnrichmentHeatmap(fsea.results, alpha=0.01)
```

`plotFactorBeeswarm` *Beeswarm plot of latent factors*

Description

Beeswarm plot of the latent factor values.

Usage

```
plotFactorBeeswarm(object, factors = "all", color_by = NULL,
  shape_by = NULL, name_color = "", name_shape = "",
  showMissing = FALSE)
```

Arguments

`object`	a trained `MOFAmodel` object.
`factors`	character vector with the factor name(s), or numeric vector with the index of the factor(s) to use. Default is 'all'
`color_by`	specifies groups or values used to color the samples. This can be either: a character giving the name of a feature, a character giving the same of a covariate (only if using `MultiAssayExperiment` as input), or 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 for the shape of samples. See color_by for how this can be specified. A maximum of 6 different values can be specified.
`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)
`showMissing`	logical indicating whether to remove samples for which `shape_by` or `color_by` is missing.

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 `plotFactorScatter` for doing scatter plots and `plotFactorHist` for doing histogram plots

Value

Returns a `ggplot2` object

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotFactorBeeswarm(MOFA_CLL, factors=1:3)
plotFactorBeeswarm(MOFA_CLL, factors=1:2, color_by= "IGHV")
```

```
# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotFactorBeeswarm(MOFA_scMT)
```

`plotFactorCor` *Plot correlation matrix between latent factors*

Description

Function to plot the correlation matrix between the latent factors.

Usage

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

Arguments

`object`	a trained `MOFAmodel` 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

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotFactorCor(MOFA_CLL)

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotFactorCor(MOFA_scMT)
```

plotFactorHist *Plot histogram of latent factor values*

Description

Plot a histogram of latent factor values.

Usage

```
plotFactorHist(object, factor, group_by = NULL, group_names = "",
  alpha = 0.5, binwidth = NULL, showMissing = FALSE)
```

Arguments

object	a trained MOFAmodel object.
factor	character vector with the factor name or numeric vector with the index of the factor.
group_by	specifies groups used to color the samples of the histogram. This can be either: a character giving the name of a feature, the name of a covariate (only if using a MultiAssayExperiment as input), or a vector of the same length as the number of samples.
group_names	names for the groups.
alpha	transparency parameter. Default is 0.5
binwidth	binwidth for histogram. Default is NULL, which uses ggplot2 default calculation.
showMissing	boolean indicating whether to remove sample for which group_by is missing (default is FALSE)

Details

One of the first steps for the annotation of factors is to visualise and color them using known covariates such as phenotypic or clinical data.
This method generates a histogram of the sample values in a given latent factor.
Similar functions are plotFactorScatter for doing scatter plots between pairs of factors and plotFactorBeeswarm for doing Beeswarm plots of single factors.

Value

Returns a ggplot2 object

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotFactorHist(MOFA_CLL, factor=1)
plotFactorHist(MOFA_CLL, factor=1, group_by= "IGHV")

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotFactorHist(MOFA_scMT, factor=2)
```

plotFactorScatter *Scatterplot of two latent factors*

Description

Scatterplot of the values of two latent factors.

Usage

```
plotFactorScatter(object, factors, color_by = NULL, shape_by = NULL,
  name_color = "", name_shape = "", dot_size = 1.5, dot_alpha = 1,
  showMissing = TRUE)
```

Arguments

object	a trained MOFAmodel 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
color_by	specifies groups or values used to color the samples. This can be either a character giving the name of a feature present in the training data, a character giving the same of a covariate (only if using MultiAssayExperiment as input), or 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 a character giving the name of a feature present in the training data, a character giving the same of a covariate (only if using MultiAssayExperiment as input), or a vector of the same length as the number of samples specifying discrete groups.
name_color	name for color legend.
name_shape	name for shape legend.
dot_size	dot size (default is 1.5).
dot_alpha	dot transparency (default is 1.0, no transparency).
showMissing	logical indicating whether to include samples for which shape_by or color_by is missing.

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. Similar functions are plotFactorScatters for doing multiple scatter plots and plotFactorBeeswarm for doing Beeswarm plots for single factors.

Value

Returns a ggplot2 object

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotFactorScatter(MOFA_CLL, factors=1:2)
plotFactorScatter(MOFA_CLL, factors=1:2, color_by= "IGHV", shape_by="trisomy12", showMissing=FALSE)

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotFactorScatter(MOFA_scMT, factors=c(1,3))
```

---

plotFactorScatters *Pairwise scatterplots of multiple latent factors*

Description

Scatterplots of the sample values for pair-wise combinations of multiple latent factors.

Usage

```
plotFactorScatters(object, factors = "all", showMissing = TRUE,
  color_by = NULL, name_color = "", shape_by = NULL,
  name_shape = "")
```

Arguments

object	a MOFAmodel object.
factors	character vector with the factor name(s), or numeric vector with the index of the factor(s) to use. Default is 'all'
showMissing	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: a character giving the name of a feature present in the training data, a character giving the same of a covariate (only if using MultiAssayExperiment as input), or a vector of the same length as the number of samples specifying discrete groups or continuous numeric values.
name_color	name for color legend (usually only used if color_by is not a character itself)
shape_by	specifies groups or values used to shape the samples. This can be either: a character giving the name of a feature present in the training data, a character giving the same of a covariate (only if using MultiAssayExperiment as input), or a vector of the same length as the number of samples specifying discrete groups.
name_shape	name for shape legend (usually only used if shape_by is not a character itself)

Details

One of the first steps for the annotation of factors is to visualise and overlap them with known covariates such as phenotypic or clinical data. This method generates multiple scatterplots for pair-wise combinations of several latent factors. Similar functions are plotFactorScatter for doing single scatter plots and plotFactorBeeswarm for doing Beeswarm plots for single factors.

Value

ggplot2 object

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotFactorScatters(MOFA_CLL, factors=1:3)
plotFactorScatters(MOFA_CLL, factors=1:3, color_by= "IGHV")

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotFactorScatters(MOFA_scMT)
```

---

plotTopWeights *Plot top weights*

Description

Plot top weights for a given latent factor in a given view.

Usage

```
plotTopWeights(object, view, factor, nfeatures = 10, abs = TRUE,
  scale = TRUE, sign = "both")
```

Arguments

object	a trained MOFAmodel object.
view	character vector with the view name, or numeric vector with the index of the view to use.
factor	character vector with the factor name, or numeric vector with the index of the factor to use.
nfeatures	number of top features to display. Default is 10.
abs	logical indicating whether to use the absolute value of the weights. Default is TRUE.
scale	logical indicating whether to scale all loadings from 0 to 1. Default is TRUE.
sign	can be 'positive', 'negative' or 'both' to show only positive, negative or all weigths, respectively. Default is 'both'.

Details

The weights, or the loadings, provide the mapping between the high-dimensional space (the genes) and the low-dimensional space (the factors).
They define a score for each gene on each factor, such that genes with no association with the factor are expected to have values close to zero, whereas genes with strong association with the factor are expected to have large absolute values.
The sign of the loading indicates the direction of the effect: A positive loading indicates that the feature is more active in the cells with positive factor values, while a negative loading indicates that the feature is more active in the cells with negative factor values.

Value

Returns a ggplot2 object

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotTopWeights(MOFA_CLL, view="Mutations", factor=1, nfeatures=3)
plotTopWeights(MOFA_CLL, view="Mutations", factor=1, nfeatures=3, sign = "positive")
plotTopWeights(MOFA_CLL, view="Mutations", factor=1, nfeatures=3, sign = "negative")

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotTopWeights(MOFA_scMT, view="RNA expression", factor=1)
```

---

plotVarianceExplained *Plot variance explained by the model*

Description

Method to plot variance explained (R-squared) by the MOFA model for each view and latent factor. As a measure of variance explained for gaussian data we adopt the coefficient of determination (R2).
For details on the computation see the help of the calculateVarianceExplained function

Usage

```
plotVarianceExplained(object, cluster = TRUE, ...)
```

Arguments

object	a MOFAmodel object.
cluster	logical indicating whether to do hierarchical clustering on the plot
...	extra arguments to be passed to calculateVarianceExplained

Value

ggplot object

Examples

```
# Using an existing trained model on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotVarianceExplained(MOFA_CLL)

# Using an existing trained model on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotVarianceExplained(MOFA_scMT)
```

plotWeights *Plot Weights*

Description

An important step to annotate factors is to visualise the feature loadings.
This function plots all loadings for a given latent factor and view, optionally labeling the top ones.
A similar function is plotTopWeights, which displays only the top features with highest loading.

Usage

```
plotWeights(object, view, factor, nfeatures = 10, abs = FALSE,
  manual = NULL, color_manual = NULL, scale = TRUE)
```

Arguments

object	a MOFAmodel object.
view	character vector with the view name, or numeric vector with the index of the view to use.
factor	character vector with the factor name, or numeric vector with the index of the factor to use.
nfeatures	number of top features to label.
abs	logical indicating whether to use the absolute value of the weights.
manual	A list of character vectors with features to be manually labelled (i.e. list(c("feature1","feature2"), c("feature3","feature4")). Each character vector specifies a set of features that will be highlighted with the same color, as specified in color_manual.
color_manual	a character vector with colors, one for each character vector of manual
scale	logical indicating whether to scale all loadings from 0 to 1.

Details

The weights, or the loadings, provide the mapping between the high-dimensional space (the genes) and the low-dimensional space (the factors).
They define a score for each gene on each factor, such that genes with no association with the factor are expected to have values close to zero, whereas genes with strong association with the factor are expected to have large absolute values.
The sign of the loading indicates the direction of the effect: A positive loading indicates that the feature is more active in the cells with positive factor values, while a negative loading indicates that the feature is more active in the cells with negative factor values.

Value

a plot of all feature loadings for the given factor

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotWeights(MOFA_CLL, view="Mutations", factor=1)

# Highlight specific features
plotWeights(MOFA_CLL, view="Mutations", factor=1,
  manual=list(c("IGHV"), c("TP53", "del17p13")),
  color_manual=c("blue","red")
 )

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotWeights(MOFA_scMT, view="RNA expression", factor=1, nfeatures=15)
```

---

plotWeightsHeatmap *Plot heatmap of the weights*

Description

Function to visualize the loadings 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 loadings of individual factors, use the functions plotWeights and plotTopWeights

Usage

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

Arguments

object	a trained MOFAmodel object.
view	character vector with the view name(s), or numeric vector with the index of the view(s) to use. Default is 'all'
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 loadings with a magnitude below this threshold (in all factors) are removed
...	extra arguments passed to pheatmap.

Details

The weights, or the loadings, provide the mapping between the high-dimensional space (the genes) and the low-dimensional space (the factors).
They define a score for each gene on each factor, such that genes with no association with the factor are expected to have values close to zero, whereas genes with strong association with the factor are expected to have large absolute values.
The sign of the loading indicates the direction of the effect: A positive loading indicates that the feature is more active in the cells with positive factor values, while a negative loading indicates that the feature is more active in the cells with negative factor values.

Value

produces a heatmap of feature weights for all factors

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
plotWeightsHeatmap(MOFA_CLL, view="Mutations")
plotWeightsHeatmap(MOFA_CLL, view="Mutations", factors=1:3)

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
plotWeightsHeatmap(MOFA_scMT, view="RNA expression")
```

---

predict *Do predictions using a fitted MOFA model*

Description

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

Usage

```
predict(object, views = "all", factors = "all", type = c("inRange",
  "response", "link"))
```

Arguments

object	a MOFAmodel 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".
type	type of prediction returned, either: • **response**: gives the response vector, the mean for Gaussian and Poisson, and success probabilities for Bernoulli. • **link**: gives the linear predictions. • **inRange**: rounds the fitted values of integer-valued distributions (Poisson and Bernoulli) to the next integer. This is the default option.

Details

Matrix factorization models generate a denoised and condensed low-dimensional representation of the data which capture the main sources of heterogeneity of the data. Such representation can be used to do predictions (data reconstruction) and imputation (see impute).
For mathematical details, see the Methods section of the MOFA article.

Value

Returns a list with data predictions.

Examples

```
library(ggplot2)

# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)

# predict drug response data using all factors
predictedDrugs <- predict(MOFA_CLL, view="Drugs")

# predict all views using all factors (default)
predictedAll <- predict(MOFA_CLL)

# predict Mutation data using all factors, returning Bernoulli probabilities
predictedMutations <- predict(MOFA_CLL, view="Mutations", type="response")

# predict Mutation data using all factors, returning binary classes
predictedMutationsBinary <- predict(MOFA_CLL, view="Mutations", type="inRange")

# Compare the predictions with the true data
pred <- as.numeric(predictedAll$Drugs)
true <- as.numeric(getTrainData(MOFA_CLL)$Drugs)
qplot(pred,true) + geom_hex(bins=100) + coord_equal() +
   geom_abline(intercept=0, slope=1, col="red")

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)

# Predict all views using all factors (default)
predictedAll <- predict(MOFA_scMT)

# Compare the predictions with the true data
view <- "RNA expression"
pred <- as.numeric(predictedAll[[view]])
true <- as.numeric(getTrainData(MOFA_scMT)[[view]])
qplot(pred,true) + geom_hex(bins=100) + coord_equal() +
   geom_abline(intercept=0, slope=1, col="red")
```

---

prepareMOFA *prepare a MOFAobject for training*

Description

Function to prepare a MOFAmodel object for training. Here, data, input/output option are specified and data, model and training options can be set.

Usage

```
prepareMOFA(object, DataOptions = NULL, ModelOptions = NULL,
  TrainOptions = NULL)
```

Arguments

object	an untrained MOFAmodel
DataOptions	list of DataOptions (see getDefaultDataOptions details). If NULL, default data options are used.
ModelOptions	list of ModelOptions (see getDefaultModelOptions for details). If NULL, default model options are used.
TrainOptions	list of TrainOptions (see getDefaultTrainOptions for details). If NULL, default training options are used.

Value

Returns an untrained MOFAmodel with specified data, model and training options. Next step is to train the model with runMOFA

Examples

```
# load data
data("CLL_data", package = "MOFAdata")
#create a MOFAmodel object
MOFAobject <- createMOFAobject(CLL_data)
# set options
TrainOptions <- getDefaultTrainOptions()
ModelOptions <- getDefaultModelOptions(MOFAobject)
DataOptions <- getDefaultDataOptions()
# prepare MOFAmodel object for training
MOFAobject <- prepareMOFA(MOFAobject,
DataOptions = DataOptions,
ModelOptions = ModelOptions,
TrainOptions = TrainOptions
)
MOFAobject
```

---

qualityControl *qualityControl*

Description

Function to do quality control on a MOFAmodel object.

Usage

```
qualityControl(object, verbose = FALSE)
```

Arguments

object	a trained MOFAmodel object.
verbose	logical indicating whether to generate a verbose output.

Value

none

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
# do quality control
qualityControl(MOFAobject, verbose=TRUE)
```

---

regressCovariates *regress out a covariate from the training data*

Description

Function to regress out a covariate from the training data.
If you have technical sources of variability (i.e. batch effects) that you do not want to be captured by factors in the model, you should regress them out before fitting MOFA. This function performs a simple linear regression model, extracts the residuals, and replaces the original data in the TrainingData slot.
Why is this important? If big technical factors exist, the model will "focus" on capturing the variability driven by these factors, and smaller sources of variability could be missed.
But... can we not simply add those covariates to the model? Technically yes, but we extensively tested this functionality and it was not yielding good results.
The reason is that covariates are usually discrete labels that do not reflect the underlying molecular biology. For example, if you introduce age as a covariate, but the actual age is different from the "molecular age", the model will simply learn a new factor that corresponds to this "latent" molecular age, and it will drop the covariate from the model.
We recommend factors to be learnt in a completely unsupervised manner and subsequently relate them to the covariates via visualisation or via a simple correlation analysis (see our vignettes for more details).

Usage

```
regressCovariates(object, views, covariates, min_observations = 5)
```

Arguments

object	an untrained MOFAmodel
views	the view(s) to regress out the covariates.
covariates	a vector (one covariate) or a data.frame (for multiple covariates) where each row corresponds to one sample, sorted in the same order as in the input data matrices. You can check the order by doing sampleNames(MOFAobject). If required, fill missing values with NA, which will be ignored when fitting the linear model.
min_observations	number of non-missing observations required

Value

Returns an untrained MOFAmodel where the specified covariates have been regressed out in the training data.

Examples

```
data("CLL_data", package = "MOFAdata")
data("CLL_covariates", package = "MOFAdata")
library(MultiAssayExperiment)
mae_CLL <- MultiAssayExperiment(
experiments = CLL_data,
colData = CLL_covariates
)
MOFAobject <- createMOFAobject(mae_CLL)
MOFAobject <- prepareMOFA(MOFAobject)
MOFAobject_reg <- regressCovariates(
object = MOFAobject,
views = c("Drugs","Methylation","mRNA"),
covariates = InputData(MOFAobject)$Gender
)
# MOFA object with training data after regressing out the specified covariate
MOFAobject_reg
```

---

runEnrichmentAnalysis *Feature Set Enrichment Analysis*

Description

Method to perform feature set enrichment analysis on the feature loadings.
The input is a data structure containing the feature set membership, usually relating biological pathways to genes.
The output is a matrix of dimensions (number_gene_sets,number_factors) with p-values and other statistics.

Usage

```
runEnrichmentAnalysis(object, view, feature.sets, factors = "all",
  local.statistic = c("loading", "cor", "z"),
  global.statistic = c("mean.diff", "rank.sum"),
  statistical.test = c("parametric", "cor.adj.parametric",
  "permutation"), transformation = c("abs.value", "none"),
  min.size = 10, nperm = 1000, cores = 1, p.adj.method = "BH",
  alpha = 0.1)
```

Arguments

object	a MOFAmodel object.
view	name of the view to perform enrichment on. Make sure that the feature names of the feature set file match the feature names in the MOFA model.
feature.sets	data structure that holds feature set membership information. Must be either a binary membership matrix (rows are feature sets and columns are features) or a list of feature set indexes (see vignette for details).

`factors` character vector with the factor names to perform enrichment on. Alternatively, a numeric vector with the index of the factors. Default is all factors.

`local.statistic` the feature statistic used to quantify the association between each feature and each factor. Must be one of the following: loading (the output from MOFA, default), cor (the correlation coefficient between the factor and each feature), z (a z-scored derived from the correlation coefficient).

`global.statistic` the feature set statisic computed from the feature statistics. Must be one of the following: "mean.diff" (difference in means between the foreground set and the background set, default) or "rank.sum" (difference in rank sums between the foreground set and the background set).

`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" (very liberal, default), "cor.adj.parametric" (very conservative, adjusts for the inter-gene correlation), "permutation" (non-parametric, the recommended one if you can do sufficient number of permutations)

`transformation` optional transformation to apply to the feature-level statistics. Must be one of the following "none" or "abs.value" (default).

`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.

`cores` number of cores to run the permutation analysis in parallel. Only relevant if statistical.test is set to "permutation". Default is 1.

`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

Details

This function relates the factors to pre-defined biological pathways by performing a gene set enrichment analysis on the loadings. The general idea is to compute an activity score for every pathway in each factor based on its corresponding gene loadings.
This function is particularly useful when a factor is difficult to characterise based only on the genes with the highest loading.
We provide several pre-build 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 the following elements:

`feature.statistics` feature statistics

`set.statistics` feature-set statistics

`pval` raw p-values

`pval.adj` adjusted p-values

`sigPathways` a list with enriched pathways

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)

# perform Enrichment Analysis on mRNA data using pre-build Reactome gene sets
data("reactomeGS", package = "MOFAdata")
fsea.results <- runEnrichmentAnalysis(MOFAobject, view="mRNA", feature.sets=reactomeGS)

# heatmap of enriched pathways per factor at 1% FDR
plotEnrichmentHeatmap(fsea.results, alpha=0.01)

# plot number of enriched pathways per factor at 1% FDR
plotEnrichmentBars(fsea.results, alpha=0.01)

# plot top 10 enriched pathways on factor 5:
plotEnrichment(MOFAobject, fsea.results, factor=5,  max.pathways=10)
```

---

runMOFA *train a MOFA model*

Description

Function to train an untrained MOFAmodel object.

Usage

```
runMOFA(object, outfile = NULL)
```

Arguments

object	an untrained MOFAmodel object
outfile	output .hdf5 file

Details

In this step the R package is calling the mofapy Python package, where the the training is performed. The interface with Python is done with the reticulate package. If you have several versions of Python installed and Rstudio is not detecting the correct one, you can change it using reticulate::use_python.

Value

a trained MOFAmodel object

Examples

```
data <- makeExampleData()
# create and prepare the MOFAmodel
MOFAobject <- createMOFAobject(data)
MOFAobject <- prepareMOFA(MOFAobject)
# fit the model (takes some time)
## Not run:
# MOFAobject <- runMOFA(MOFAobject)
```

```
# MOFAobject

## End(Not run)
```

`sampleNames` *sampleNames: set and retrieve sample names*

Description

Function to set and retrieve sample names.

Usage

```
sampleNames(object)

sampleNames(object) <- value

## S4 method for signature 'MOFAmodel'
sampleNames(object)

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

Arguments

`object`	a `MOFAmodel` object.
`value`	a character vector of sample names

Value

character vector with the sample names

Examples

```
data("CLL_data", package = "MOFAdata")
MOFAobject  <- createMOFAobject(CLL_data)
head(sampleNames(MOFAobject))
```

`selectModel` *Select the best model from a list of trained `MOFAmodel` objects*

Description

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

Usage

```
selectModel(models, plotit = TRUE)
```

Arguments

models	a list containing MOFAmodel objects.
plotit	show a plot of the characteristics of the compared MOFAmodel objects (ELBO value and number of inferred factors)?

Value

a single MOFAmodel with the best ELBO statistics from the provided list

Examples

```
# Simulate Data
data <- makeExampleData()
# Create MOFA model
MOFAobject <- createMOFAobject(data)
# Prepare MOFA model
MOFAobject <- prepareMOFA(MOFAobject)
# Train several instances of MOFA models
n_inits <- 3
MOFAlist <- lapply(seq_len(n_inits), function(i) runMOFA(MOFAobject, outfile=tempfile()))
selectModel(MOFAlist)
```

Status *Status: set and retrieve training status*

Description

Function to set and retrieve taining status.

Usage

```
Status(object)

Status(object) <- value

## S4 method for signature 'MOFAmodel'
Status(object)

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

Arguments

object	a MOFAmodel object.
value	character indicating whether the object is trained or untrained

Value

character indicating whether the object is trained or untrained

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
Status(MOFAobject)
```

`subsetFactors` *Subset factors*

Description

Method to subset (or sort) factors.
Some factors might not be interesting for the downstream analysis and the user can choose to remove them. This has no effect on the values of the other factors. For example, this could be done if the model contains factors which are inactive in all views.

Usage

```
subsetFactors(object, factors)
```

Arguments

`object`	a `MOFAmodel` object.
`factors`	character vector with the factor names (LF1,LF2,...), or numeric vector with the index of the factors.

Value

`MOFAmodel` object with a subset of factors

Examples

```
# Using an existing trained model on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
MOFA_CLL_small <- subsetFactors(MOFA_CLL, factors=c(1,2,3))
MOFA_CLL_small
MOFA_CLL_small <- subsetFactors(MOFA_CLL, factors=c("LF1","LF2","LF3"))
MOFA_CLL_small
```

subsetSamples *Subset samples*

Description

Method to subset (or sort) samples.
This function can remove samples from the model. For example, you might want to observe the effect of Factor 1 on a subset of samples. You can create a new MOFAmodel excluding some samples and then visualise the effect of Factor 1 on the remaining ones, for instance via plotDataHeatmap or plotFactorScatter.
This functionality is only for exploratory purposes. In the case of outliers, we strongly recommend removing them before training the model.

Usage

```
subsetSamples(object, samples)
```

Arguments

object	a MOFAmodel object.
samples	character vector with the sample names, numeric vector with the sample indices or logical vector with the samples to be kept as TRUE.

Value

MOFAmodel object with a subset of samples

Examples

```
# Using an existing trained model on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
# Subset samples via character vector
MOFA_CLL_small <- subsetSamples(MOFA_CLL, samples=c("H045","H109","H024","H056"))
MOFA_CLL_small
# Subset samples via numeric vector
MOFA_CLL_small <- subsetSamples(MOFA_CLL, samples=1:10)
MOFA_CLL_small
```

subsetViews *Subset views*

Description

Method to subset (or sort) views. This function can remove entire views from the model. For example, you might want to generate the plotVarianceExplained plot excluding a particular view.
This functionality is only for exploratory purposes. If some view(s) are not of interest we strongly recommend removing them before training the model.

Usage

```
subsetViews(object, views)
```

Arguments

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

Value

MOFAmodel object with a subset of views

Examples

```
# Using an existing trained model on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
# Subset views via character vector
MOFA_CLL_small <- subsetViews(MOFA_CLL, views=c("Drugs","Methylation"))
MOFA_CLL_small
# Subset views via numeric vector
MOFA_CLL_small <- subsetViews(MOFA_CLL, views=2:3)
MOFA_CLL_small
```

---

trainCurveELBO *Training curve for Evidence Lower Bound (ELBO)*

Description

MOFA inference is done using the variational Bayes algorithm, which maximises a quantity called the Evidence Lower Bound (ELBO). The ELBO is supposed to increase monotonically up to convergence, but it can decrease substantially when dropping inactive factors. For more details read the supplementary methods. The frequency of ELBO computation as well as the convergence criteria are defined as hyperparameters in prepareMOFA.
All Training statistics, including the ELBO, can be fetch from the TrainStats slot of MOFAmodel .

Usage

```
trainCurveELBO(object, logScale = FALSE)
```

Arguments

object	a MOFAmodel object.
logScale	boolean indicating whether to apply log transform

Value

plot of ELBO values during training

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
trainCurveELBO(MOFA_CLL)
trainCurveELBO(MOFA_CLL, logScale= TRUE)

# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
trainCurveELBO(MOFA_scMT)
```

---

trainCurveFactors *Training curve for the number of active factors*

Description

The MOFA model starts with an initial number of factors and inactive factors can be dropped during training if they explain small amounts of variation (as defined in `getDefaultModelOptions`). This allows the model to automatically infer the dimensionality of the latent space. The corresponding hyperparameters are defined in `prepareMOFA`.
All training statistics, including the number of active factors, can be fetch from the TrainStats slot of `MOFAmodel` .

Usage

```
trainCurveFactors(object)
```

Arguments

`object`	a `MOFAmodel` object.

Value

plot of number of active factors during training

Examples

```
# Example on the CLL data
filepath <- system.file("extdata", "CLL_model.hdf5", package = "MOFAdata")
MOFA_CLL <- loadModel(filepath)
trainCurveFactors(MOFA_CLL)
# Example on the scMT data
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFA_scMT <- loadModel(filepath)
trainCurveFactors(MOFA_scMT)
```

TrainData *TrainData: set and retrieve training data*

Description

Function to set and retrieve taining data.

Usage

```
TrainData(object)

TrainData(object) <- value

## S4 method for signature 'MOFAmodel'
TrainData(object)

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

Arguments

object	a MOFAmodel object.
value	a list of matrices containing the training data

Value

list of matrices containing the training data

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
str(TrainData(MOFAobject))
```

TrainOptions *TrainOptions: set and retrieve training options*

Description

Function to set and retrieve training options.

Usage

```
TrainOptions(object)

TrainOptions(object) <- value

## S4 method for signature 'MOFAmodel'
TrainOptions(object)

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

Arguments

object	a MOFAmodel object.
value	a list with training options

Value

list of training options

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
TrainOptions(MOFAobject)
```

TrainStats *TrainStats: set and retrieve training statistcs*

Description

Function to set and retrieve taining statistics

Usage

```
TrainStats(object)

TrainStats(object) <- value

## S4 method for signature 'MOFAmodel'
TrainStats(object)

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

Arguments

object	a MOFAmodel object.
value	list of different training statistics (EBLO, number of factors)

Value

list of different training statistics (EBLO, number of factors)

Examples

```
# load a trained MOFAmodel object
filepath <- system.file("extdata", "scMT_model.hdf5", package = "MOFAdata")
MOFAobject <- loadModel(filepath)
str(TrainStats(MOFAobject))
```

`viewNames` *viewNames: set and retrieve view names*

Description

Function to set and retrieve view names.

Usage

```
viewNames(object)

viewNames(object) <- value

## S4 method for signature 'MOFAmodel'
viewNames(object)

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

Arguments

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

Value

character vector with the names for each view

Examples

```
data("CLL_data", package = "MOFAdata")
MOFAobject  <- createMOFAobject(CLL_data)
viewNames(MOFAobject)
viewNames(MOFAobject) <- c("DrugResponses", viewNames(MOFAobject)[2:4])
viewNames(MOFAobject)
```

Index