Package 'NMF'

April 15, 2010

Type Package

Title Algorithms and framework for Nonnegative Matrix Factorization (NMF).

Version 0.4

Date 2010-03-30

Author Renaud Gaujoux

Maintainer Renaud Gaujoux <renaud@cbio.uct.ac.za>

Description This package provides a framework to perform Non-negative Matrix Factorization (NMF). It implements a set of already published algorithms and seeding methods, and provides a framework to test, develop and plug new/custom algorithms.

License GPL (>=2)

URL http://nmf.r-forge.r-project.org

LazyLoad yes

Depends R (>= 2.10), methods, stats, graphics

Suggests Biobase, RColorBrewer, fastICA, bigmemory, doMC

Collate utils.R options.R NMF-class.R NMFstd-class.R NMFOffset-class.R NMFns-class.R NMFfit-class.R NMFSet-class.R registry.R NMFStrategy-class.R NMFStrategyFunction-class.R NMFStrategyIterative-class.R snmf.R lnmf.R pe-nmf.R ica.R nndsvd.R seed.R nmf.R Bioc-layer.R nmf-package.R

R topics documented:

`NMF-package` *NMF Package Overview*

Description

The NMF package provides methods to perform Nonnegative Matrix Factorization (NMF) , as well as a framework to develop and test new NMF algorithms.

A number of standard algorithms and seeding methods are implemented. Tuned visualisation and post-analysis methods help in the evaluation of the algorithms' performances or in the interpretation of the results.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Definition of Nonnegative Matrix Factorization in its modern formulation:

Lee D.D. and Seung H.S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, **401**, 788–791.

Historical first definition and algorithms:

Paatero, P., Tapper, U. (1994). Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, **2**, 111–126 , doi:10.1002/env.3170050203.

See Also

`NMF-class`, `nmf`, `Biobase`

Examples

```
# run default NMF algorithm on a random matrix
V <- matrix(runif(10000), 500, 20)
res <- nmf(V, 3)
res

# compute some quality measures
summary(res)

# Visualize the results as heatmaps
## Not run: metaheatmap(res) # mixture coefficients
## Not run: metaheatmap(res, 'features') # basis vectors

# run default NMF algorithm on a random matrix with actual patterns
set.seed(123456)
V <- syntheticNMF(500, 3, 20, noise=TRUE)
res <- nmf(V, 3)
res

# compute some quality measures
summary(res)

# Visualize the results as heatmaps
## Not run: metaheatmap(res) # mixture coefficients
## Not run: metaheatmap(res, 'features') # basis vectors
```

`advanced` *Advanced usage of package NMF*

Description

Allow the user to get/set/define package NMF specific options in the same way as with base functions `options` and `getOption`.

Usage

```
nmf.options(..., runtime = FALSE)
nmf.getOption(name)
nmf.options.reset()
nmf.options.runtime()
```

Arguments

`...`	any options can be defined, using 'name = value' or by passing a list of such tagged values. However, only the ones below are used in package NMF. Further, 'nmf.options('name') == nmf.options()['name']', see the example.
`name`	a character string holding an option name.
`runtime`	a boolean used to specify if main interface function `nmf` should store the option into the initial `NMF` object before performing the computation.

Details

nmf.getOption get the value of a single option.

nmf.options gets/sets/defines options in the way of base function `options`. Invoking 'nmf.options()' with no arguments returns a list with the current values of the options. To access the value of a single option, one should use `nmf.getOption("error.track")`, e.g., rather than `nmf.options("error.track")` which is a `list` of length one.

nmf.options.reset Reset all *built-in options* to their default values. Note that only built-in are reset. The options defined by the user during the current session will keep their values.

Value

For `nmf.getOption`, the current value set for option `name`, or `NULL` if the option is unset.

For `nmf.options()`, a list of all set options sorted by name. For `options(name)`, a list of length one containing the set value, or `NULL` if it is unset. For uses setting one or more options, a list with the previous values of the options changed (returned invisibly).

Options set in package NMF

debug logical. Similar to option `'verbose'` (see below), but reports more information.

default.algorithm `character`. The default NMF algorithm used by the `nmf` method when called without argument `method`.

default.seed `character`. The default seeding method used by the `nmf` method when called without argument `seed`.

error.track logical. Should the estimation error be tracked during the computations? If set to `TRUE` then the error track can be plotted using method `plot`. The step size of the error track is set via option `track.interval` (see below).

parallel.backend `character` or `numeric`. The default parallel backend used by the `nmf` method when called with argument `nrun` greater than 1.

Currently it accepts the following values: `'mc'` or a number that specifies the number of cores to use, `'seq'` or `NULL` to use sequential backend for `foreach`, and the empty string `''` to completely disable the parallel computation.

track.interval numeric. The number of iterations performed between two consecutive error points. For performance reason, this value should be too small, as the computation of the estimation error can be time consuming (Default value is 30).

verbose logical. Should R report extra information about the computations?

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

See Also

options

Examples

```
# save all options value
op <- nmf.options();
utils::str(op) # op may contain functions.

    nmf.getOption("track.interval") == nmf.options()$track.interval # the latter needs more

    x <- matrix(runif(50*10), 50, 10) # create a random target matrix
    # or define a synthetic data with a hidden pattern using function syntheticNMF (see ?syn
    ## Not run: x <- syntheticNMF(50, 5, 10, noise=TRUE)

    # perform default NMF computation
    res <- nmf(x, 3)

    # Toogle on verbose mode
    nmf.options(verbose = TRUE)
    res <- nmf(x, 3)

# Toogle on debug mode
    nmf.options(debug = TRUE)
    res <- nmf(x, 3)

    # set the error track step size, and save previous value
    old.o <- nmf.options(track.interval = 5)
    old.o

    # check options
    utils::str(nmf.options())
    # reset to default values
    nmf.options.reset()
    utils::str(nmf.options())
```

---

basis-methods *Extract/Set the matrix of basis vectors from a NMF model*

Description

basis and basis<- are S4 generic functions which respectively extract and set the matrix of basis vectors (i.e. the first matrix factor) of a NMF model. For example, in the case of the standard NMF model $V \equiv WH$, method basis will return matrix W.

`coef` and `coef<-` are S4 methods defined for the associated generic functions from package `stats` (See coef). They respectively extract and set the matrix of mixture coefficients (i.e. the second matrix factor) of a NMF model. For example, in the case of the standard NMF model $V \equiv WH$, method `coef` will return matrix H.

Methods `coefficients` and `coefficients` are simple aliases for methods `coef` and `coef<-` respectively.

Methods

basis `signature(object = "NMF")`: Extracts the matrix of basis vectors from NMF model `object`.

Note that it is implemented as a pure virtual method, that must be overloaded by sub-classes that implement concrete NMF models. It throws an error if directly called. See `NMF` for more details.

basis<- `signature(object = "NMF", value = "matrix"`: Sets the matrix of basis vectors from NMF model `object`.

Note that it is implemented as a pure virtual method, that must be overloaded by sub-classes that implement concrete NMF models. It throws an error if directly called. See `NMF` for more details.

coef `signature(object = "NMF")`: Extracts the matrix of mixture coefficients from NMF model `object`.

Note that it is implemented as a pure virtual method, that must be overloaded by sub-classes that implement concrete NMF models. It throws an error if directly called. See `NMF` for more details.

coef<- `signature(object = "NMF", value = "matrix"`: Sets the matrix of mixture coefficients from NMF model `object`.

Note that it is implemented as a pure virtual method, that must be overloaded by sub-classes that implement concrete NMF models. It throws an error if directly called. See `NMF` for more details.

See Also

NMF, NMFstd, NMFfit

`Comparing the results of different NMF runs`

Comparing Results from Different NMF Runs

Description

This functions allow to compare the results of different NMF runs. The results do not need to be from the same algorithm, nor even of the same dimension.

Usage

```
## S4 method for signature 'ANY':
as.NMFList(..., unlist=FALSE)

## S4 method for signature 'list':
compare(object, ..., unlist=FALSE)

## S4 method for signature 'NMFList':
plot(x, ...)

## S4 method for signature 'NMFList':
summary(object, sort.by=NULL, select=NULL, ...)
```

Arguments

`object`	A `list` or an object of class `NMFList`.
`select`	the columns to be output in the result `data.frame`. The column are given by their names (partially matched). The column names are the names of the summary measures returned by the `summary` methods of the corresponding NMF results.
`sort.by`	the sorting criteria, i.e. a partial match of a column name, by which the result `data.frame` is sorted. The sorting direction (increasing or decreasing) is computed internally depending on the chosen criteria (e.g. decreasing for the cophenetic coefficient, increasing for the residuals).
`unlist`	boolean to specify if the arguments should be unlisted before wrapping them into a `NMFList` object or comparing them.
`x`	An object of class `NMFList`.
`...`	Used to pass extra arguments to subsequent calls: • in `as.NMFList` the list of NMF results to wrap into a `NMFList` object. • in `plot`: graphical parameters passed to the `plot` function. • in `compare` and `summary`: extra arguments passed to the `summary` method of each result object (cf. `summary,NMF-method`).

Details

as.NMFList : wrap the arguments into a `NMFList` object.

compare : shortcut for `summary(as.NMFList(object)` (cf. `summary` method below).

plot : plot on a single graph the residuals tracks for each element in `x`. See function `nmf` for details on how to enable the tracking of residuals.

runtime : returns the computational time used to compute all the results in the list, as stored in slot `runtime` of `object`.

The time is computed using the function `system.time` which returns object of class `proc_time`.

Note that argument `...` is not used.

summary : `summary` method for objects of class `NMFList`.

It compute summary measures for each NMF result in the list and return them in rows in a `data.frame`. By default all the measures are included in the result, and `NA` values are used where no data is available or the measure does not apply to the result object (e.g. the dispersion for single NMF runs is not meaningful). This method is very useful to compare and evaluate the performance of different algorithms.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Metagenes and molecular pattern discovery using matrix factorization Brunet, J.~P., Tamayo, P., Golub, T.~R., and Mesirov, J.~P. (2004) Proc Natl Acad Sci U S A 101(12), 4164–4169.

Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis Kim, H. & Park, H. (2007) Bioinformatics. http://dx.doi.org/10.1093/bioinformatics/btm134.

See Also

NMFfitX1, NMFfitXn, summary

Examples

```
# generate a synthetic dataset with known classes: 50 features, 18 samples (5+5+8)
n <- 50; counts <- c(5, 5, 8);
V <- syntheticNMF(n, counts, noise=TRUE)
## Not run: metaHeatmap(V)

# build the class factor
groups <- as.factor(do.call('c', lapply(seq(3), function(x) rep(x, counts[x]))))

# perform multiple runs of NMF (keep best only)
res <- nmf(V, 3, nrun=5)
res

# compute summary measures
summary(res)

# compute more summary measures
summary(res, target=V, class=groups)

# plot a heatmap of the consensus matrix with extra annotations
## Not run: metaHeatmap(res, class=groups)

# retrieve the predicted clusters of samples
predict(res)
```

```
# perform multiple runs of NMF and keep all the runs
res <- nmf(V, 3, nrun=5, .options='k')
res

# extract best fit
fit(res)

# compute/show computational times
runtime.all(res)
seqtime(res)
```

---

esGolub *Golub ExpressionSet from Brunet et al. Paper*

Description

The original data is related to Golub et al., and this version is the one used and referenced in Brunet et al. The samples are from 27 patients with acute lymphoblastic leukemia (ALL) and 11 patients with acute myeloid leukemia (AML).

The samples were assayed using Affymetrix Hgu6800 chips and the original data on the expression of 7129 genes (Affymetrix probes) are available on the Broad Institute web site (see references below).

The data in `esGolub` were obtained from the web site related to Brunet et al.'s publication on an application of Nonnegative Matrix Factorization (see link in section *Source*).

They contain the 5,000 most highly varying genes according to their coefficient of variation, and were installed in an object of class `ExpressionSet-class`.

Usage

```
data(esGolub)
```

Format

There are 3 covariates listed.

- Samples: The original sample labels.
- ALL.AML: Whether the patient had AML or ALL. It is a `factor` with levels `c('ALL', 'AML')`.
- Cell: ALL arises from two different types of lymphocytes (T-cell and B-cell). This specifies which for the ALL patients; There is no such information for the AML samples. It is a `factor` with levels `c('T-cell', 'B-cell', NA)`.

Source

http://www.broadinstitute.org/publications/broad872

References

Metagenes and molecular pattern discovery using matrix factorization Brunet, J.~P., Tamayo, P., Golub, T.~R., and Mesirov, J.~P. (2004) Proc Natl Acad Sci U S A 101(12), 4164–4169.

Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring , Science, 531-537, 1999 , T. R. Golub and D. K. Slonim and P. Tamayo and C. Huard and M. Gaasenbeek and J. P. Mesirov and H. Coller and M.L. Loh and J. R. Downing and M. A. Caligiuri and C. D. Bloomfield and E. S. Lander

Original data from Golub et al.: http://www-genome.wi.mit.edu/mpr/data_set_ALL_AML.html

Examples

```
data(esGolub)
esGolub
## Not run: pData(esGolub)
```

Handling the results of multiple NMF runs

Handling Results from Multiple NMF Runs

Description

The NMF package provides an easy way to perform multiple runs of a given NMF algorithm on a target matrix.

The result from the nmf method is a NMFfitX object that holds either all or only the best run, depending on the running options:

```
# keep only the best run object <- nmf(X, r, nrun=20) # keep all the runs object <- nmf(X, r, nrun=20, .options='k')
```

The methods documented here are used to handle such results. They are usually independent of the type of result and can be used without change in either situation (all runs kept or only the best one).

Note that when only the best result is kept, the result object conveniently inherits from all the methods available for single runs. Therefore it can be handled as if it had been computed by a single NMF run and all the methods defined for such results can be used (cf. NMFfit and NMF-utils).

See NMFfitXn and NMFfitX1 for details on the classes that implement respectively the case where all the runs are kept and only the best run is kept.

Usage

```
consensus(object, ...)

cophcor(object, ...)
```

```
dispersion(object, ...)

## S4 method for signature 'NMFfitX':
fit(object)

nrun(object, ...)

## S4 method for signature 'NMFfitX':
metaHeatmap(object, ...)

## S4 method for signature 'NMFfitXn':
predict(object, ...)

## S4 method for signature 'NMFfitX':
runtime.all(object)

## S4 method for signature 'NMFfitXn':
runtime.all(object, null=FALSE, warning=TRUE)

seqtime(object, ...)

## S4 method for signature 'NMFfitX':
summary(object, ...)
```

Arguments

`null`	used in method `runtime.all` for `NMFfitXn` objects to specify if the result should be `NULL` when the object has no time data is stored the total computation time. In this case, if `null=FALSE` (default), the method returns the sequential time (cf. `seqtime` below) instead of `NULL`. It also emits a warning which can be toggle with argument `warning`.
`object`	A `matrix` or an object that inherits from class `NMFfitX` or `NMFfit` – depending on the method.
`warning`	used in method `runtime.all` for `NMFfitXn` objects to specify if a warning should be emitted when the object has no time data the total computation time and the sequential time is returned instead of `NULL` (cf. argument `null`).
`...`	Used to pass extra arguments to subsequent calls: • in `metaHeatmap`: graphical parameters passed to function `heatmap.2` • in `predict`: extra arguments passed to function `predict,NMF-method` • in `summary`: extra arguments like `target` or `class` passed to the method `summary,NMFfit-method`.

Details

consensus :

Computes the consensus matrix associated to the multiple NMF runs described by `object`. It is computed as the mean connectivity matrix of all the runs.

It's been proposed by *Brunet et al. (2004)* to help visualising and measuring the stability of the clusters obtained by NMF approaches.

For objects of class `NMF` (e.g. results of a single NMF run, or NMF models), the consensus matrix reduces to the connectivity matrix.

Note that argument `...` is not used.

cophcor Computes the cophenetic correlation coefficient of consensus matrix `object`, generally obtained from multiple NMF runs.

The cophenetic correlation coeffificient is based on the consensus matrix (i.e. the average of connectivity matrices) and was proposed by *Brunet et al. (2004)* to measure the stability of the clusters obtained from NMF.

It is defined as the Pearson correlation between the samples' distances induced by the consensus matrix (seen as a similarity matrix) and their cophenetic distances from a hierachical clustering based on these very distances (by default an average linkage is used). See *Brunet et al. (2004)*.

Note that argument `...` is not used.

dispersion Computes the dispersion coefficient of consensus matrix `object`, generally obtained from multiple NMF runs.

The dispersion coefficient is based on the consensus matrix (i.e. the average of connectivity matrices) and was proposed by *Kim and Park (2007)* to measure the reproducibility of the clusters obtained from NMF . It is defined as:

$$\rho = \sum_{i,j=1}^{n} 4(C_{ij} - \frac{1}{2})^2.$$

, where n is the total number of samples.

We have $0 \leq \rho \leq 1$ and $\rho = 1$ only for a perfect consensus matrix, where all entries 0 or 1. A perfect consensus matrix is obtained only when all the connectivity matrices are the same, meaning that the algorithm gave the same clusters at each run. See *Kim and Park (2007)*

Note that argument `...` is not used.

fit : returns the element that achieves the lowest residual approximation error across the runs.

For `NMFfitX1` objects it coerces `object` into a `NMFfit` object. For `NMFfitXn` objects it builds and searches the vector of residuals of all the fits and returns the one with the minimum value.

Note that argument `...` is not used.

metaHeatmap Produces a heatmap of the consensus matrix using a `heatmap`-like custom function, with parameters tuned for displaying such result.

The function used to draw the heatmap is a mixture of the function `heatmap.2` from the `gplots` package, and the function `heatmap.plus` from the `heatmap.plus` package. It allows to add extra annotation rows using the `ColSideColor` argument. See `heatmap.2` and `heatmap.plus`.

nrun returns the number of NMF runs performed to compute `object`.

In the case of a `NMFfitXn` object it returns its length – as it is also a list. In the case of a `NMFfitX1` object it returns the value of its slot `nrun`. In the case of a `NMFfit` object it always returns 1 (this method exists to create a uniform access interface to NMF results).

Note that argument `...` is not used.

predict returns a `factor` that gives the predicted cluster index for each sample (resp. for each feature) based on the *best NMF factorization* stored in `object`.

The index correspond to the basis vector that most contributes to the sample (resp. to which the feature contributes the most). See `predict` for more details.

runtime.all : returns the computational time used to compute all the runs and create `object`. The time is computed using base function `system.time` which returns object of class `proc_time`.

For `NMFfitXn` objects, there is also another time measure returned by the `seqtime` method, which computes the sequential computational time, that is the sum of the computational time used by each run.

Note that argument `...` is not used.

seqtime : returns the sequential CPU time spent of all the runs in the `object` – which must be an instance of class `NMFfitXn`. It is the sum of the CPU time used to compute each run. It returns `NULL` if the `object` is empty.

Note that argument `...` is not used.

summary : `summary` method for objects of class `NMFfitX`.

It computes a set of measures to help evaluate the quality of the *best factorization* of the set. The result is similar to the result from the `summary` method of `NMFfit` objects. See `NMF` for details on the computed measures. In addition, the cophenetic correlation coefficient and the dispersion coefficient of the consensus matrix are returned, as well as the total computational time. See the related methods above.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Metagenes and molecular pattern discovery using matrix factorization Brunet, J.~P., Tamayo, P., Golub, T.~R., and Mesirov, J.~P. (2004) Proc Natl Acad Sci U S A 101(12), 4164–4169.

Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis Kim, H. & Park, H. (2007) Bioinformatics. http://dx.doi.org/10.1093/bioinformatics/btm134.

See Also

NMFfitX1, NMFfitXn, summary

Examples

```
# generate a synthetic dataset with known classes: 50 features, 18 samples (5+5+8)
n <- 50; counts <- c(5, 5, 8);
V <- syntheticNMF(n, counts, noise=TRUE)
## Not run: metaHeatmap(V)

# build the class factor
```

```
groups <- as.factor(do.call('c', lapply(seq(3), function(x) rep(x, counts[x]))))

# perform multiple runs of NMF (keep best only)
res <- nmf(V, 3, nrun=5)
res

# compute summary measures
summary(res)

# compute more summary measures
summary(res, target=V, class=groups)

# plot a heatmap of the consensus matrix with extra annotations
## Not run: metaHeatmap(res, class=groups)

# retrieve the predicted clusters of samples
predict(res)

# perform multiple runs of NMF and keep all the runs
res <- nmf(V, 3, nrun=5, .options='k')
res

# extract best fit
fit(res)

# compute/show computational times
runtime.all(res)
seqtime(res)
```

NMF - integration with Bioconductor

Layer to use the NMF package within Bioconductor

Description

The package NMF provides an optional layer for working with common objects defined in the Bioconductor platform.

It provides:

- computation functions that support `ExpressionSet` objects as inputs.
- alias functions whose names are more intuitive when NMF is applied to bioinformatics data.
- specialized vizualization methods that adapt the titles and legend using bioinformatics terminology.
- functions to link the results with annotations, etc...

Methods

distance `signature(target = "ExpressionSet", x = "NMF", method, ...)`: returns the distance between the expression matrix and a NMF model, according to a given measure. If both argument `target` and `x` are missing, this function returns the `function` defined by argument `method`. The later can either be a `function` or a `character` string that correspond to a registered distance metric. For the moment only the metrics 'KL' and 'euclidean' are defined.

metagenes `signature(object = "NMF")`: returns the metagenes matrix according to the model defined in `object`. It is a simple alias to method `basis`.

metagenes<- `signature(object = "NMF", value = "matrix")`: sets the metagenes matrix in `object`, and returns the updated object. It is a simple alias to method `basis<-`.

metaprofiles `signature(object = "NMF")`: returns the metaprofiles matrix according to the model defined in `object`. It is a simple alias to method `coef`.

metaprofiles<- `signature(object = "NMF", value = "matrix")`: sets the metaprofiles matrix in `object`, and returns the updated object. It is a simple alias to method `coef<-`.

nmeta `signature(object = "NMF")`: returns the number of metagenes use in NMF model `object`. It's an alias to `nbasis`.

See Also

NMF, NMF-utils

`NMF-class` *Interface Class for Nonnegative Matrix Factorisation Models*

Description

This is a *virtual class* that defines a common interface to handle Nonnegative Matrix Factorisation models (NMF models) in a generic way.

It provides the definition for a minimum set of generic methods that are used in common computations and tasks in the context of Nonnegative Matrix Factorisations.

Class `NMF` makes it easy to develop new models that integrates well into the general framework implemented by the *NMF* package.

Following a few simple guidelines, new models benefit from all the functionalities available to built-in NMF models – that derive themselves from class `NMF`. See section *Defining new NMF models* below.

See section `NMFstd`, references and links therein for details on the standard NMF model and its – built-in – extensions.

Slots

This class contains a single slot, that is used internally during the computations.

`misc`**:** A list that is used internally to temporarily store algorithm parameters during the computation.

The purpose of this class is to define a common interface for NMF models as a collection of generic methods. Classes that inherits from class `NMF` are responsible for the management of data storage and the implementation of the interface's pure virtual methods.

Defining new NMF models

The minimum requirement to define a new NMF model that integrates into the framework of the *NMF* package are the followings:

- Define a class that inherits from class `NMF` and implements the new model. Say class `myNMF`.
- Implement the following S4 methods for the new class `myNMF`:

  fitted `signature(object = "myNMF", value = "matrix")`: Must return the estimation of the target matrix as fitted by the NMF model `object`.

  basis `signature(object = "myNMF")`: Must return the matrix of basis vectors (e.g. the first matrix factor in the standard NMF model).

  basis<- `signature(object = "myNMF", value = "matrix")`: Must return `object` with the matrix of basis vectors set to `value`.

  coef `signature(object = "myNMF")`: Must return the matrix of mixture coefficients (e.g. the second matrix factor in the standard NMF model).

  coef<- `signature(object = "myNMF", value = "matrix")`: Must return `object` with the matrix of mixture coefficients set to `value`.

  The *NMF* package ensures these methods are defined for classes that inherits from class `NMF`, as the methods defined for signatures `(object='NMF', ...)` and `(object='NMF', value='matrix')` throw an error when called.

- Optionally, implement method `rnmf`(signature(object="myNMF", target="numeric")). This method should fill model `object` (of class `myNMF`) with random values to fit a target matrix, whose dimension is given by the 2-length numeric vector `target`.

For concrete examples of NMF models implementations, see class `NMFstd` and its extensions (e.g. classes `NMFOffset` or `NMFns`).

Objects from the Class

Strictly speaking, because class `NMF` is virtual, no object of class `NMF` can be instantiated, only objects from its sub-classes. However, those objects are sometimes shortly referred in the documentation as "`NMF` objects" instead of "objects that inherits from class `NMF`".

For built-in models or for models that inherit from the standard model class `NMFstd`, the factory method `nmfModel` enables to easily create valid `NMF` objects in a variety of common situations. See `nmfModel` for more details.

Methods

[`signature(object = "NMF")`: sub-setting method for object of class `NMF`. Row subsets are applied to the basis matrix rows, while column subsets are applied to the mixture coefficient matrix. It returns an object of class `NMF` whose basis matrix and/or mixture coefficient matrix have been subset accordingly.

basis `signature(object = "NMF")`: returns the matrix of basis vectors according to the model defined in `object`. This is a *pure virtual* method that needs to be defined for the sub-classes of class `NMF` that implements concrete models. See also `basis`.

basis<- `signature(object = "NMF", value = "matrix")`: sets the matrix of basis vectors in `object`, and returns the updated object. This is a *pure virtual* method that needs to be defined for the sub-classes of class `NMF` that implements concrete models. See also `basis`.

coef `signature(object = "NMF")`: returns the matrix of mixture coefficients according to the model defined in `object`. This is a *pure virtual* method that needs to be defined for the sub-classes of class `NMF` that implements concrete models. See also `coef`.

coef<- `signature(object = "NMF", value = "matrix")`: sets the matrix of mixture coefficients in `object`, and returns the updated object. This is a *pure virtual* method that needs to be defined for the sub-classes of class `NMF` that implements concrete models. See also `coef<-`.

coefficients `signature(object = "NMF")`: This is a simple alias to method `coef`. See also `coefficients`.

coefficients<- `signature(object = "NMF", value = "matrix")`: This is a simple alias to method `coef<-`. See also `coef`.

connectivity `signature(x = "NMF")`: returns the connectivity matrix associated to the clusters based on NMF factorization `x`. The connectivity matrix C of a clustering is the symmetric matrix that shows the shared membership of the samples: entry C_{ij} is 1 if samples i and j belong to the same cluster, 0 otherwise.

consensus `signature(object = "NMF")`: returns the consensus matrix associated with multiple runs of NMF, as the the mean connectivity matrix across the runs. For objects of class `NMF`, it reduces to the connectivity matrix, and the method is defined to create a uniform access interface to NMF results.

dim `signature(x = "NMF")`: returns a 3-length vector containing the dimension of the target matrix together with the NMF factorization rank. For example `c(2000,30,3)` for a `NMF` object that fits a 2000x30 target matrix using 3 basis vectors.

dimnames `signature(x = "NMF")`: returns the dimension names of the NMF model `x`. It returns a 3-length list containing the row names of the basis matrix, the column names of the mixture coefficient matrix, and the column names of the basis matrix (i.e. the basis vector names).

dimnames<- `signature(x = "NMF", value)`: sets the dimension names of the NMF model `x`. `value` can be a 2 or 3-length list providing names at least for the rows of the basis vector matrix and the columns of the mixture coefficient matrix. If present, the optional third element of `value` is used to set both the names of the columns of the basis vector matrix and the rows of the mixture coefficient matrix.

distance `signature(target = "matrix", x = "NMF", method, ...)`: returns the distance between a target matrix and a NMF model, according to a given measure. If both argument `target` and `x` are missing, this function returns the `function` defined by argument

method. The later can either be a function or a character string that correspond to a registered distance metric. For the moment only the metric 'KL' and 'euclidean' are defined.

entropy signature(x = "NMF", class = "factor"): computes the entropy of NMF model x given a priori known groups of samples. See generic function entropy for more details.

evar signature(object = "NMF", target): computes the explained variance of NMF model object that approximates target. See generic function evar for more details.

featureNames signature(object = "NMF"): returns the row names of the basis matrix. If BioConductor is installed this method is defined for the generic function featureNames from the Biobase package.

featureNames<- signature(object = "NMF", value = "ANY"): sets the row names of the basis matrix. Argument value must be in a format accepted by the rownames method defined for matrices. If BioConductor is installed this method is defined for the generic function featureNames<- from the Biobase package.

fitted signature(object = "NMF"): computes the target matrix estimated by NMF model object. This is a *pure virtual* method that needs to be implemented by the sub-classes of class NMF that implements concrete models.

featureScore signature(x = "NMF"): Computes a score for each feature that reflects its specificity to one of the basis vector. The definition of the score follows *Kim and Park (2007)*. See references for more details.

extractFeatures signature(x = "NMF"): extract the features that are the most specific to each basis vector. It follows *Kim and Park (2007)*'s methodology. See references for more details.

is.empty.nmf signature(object = "NMF"): Tells if object is an empty the NMF model, that is it contains no data. It returns TRUE if the matrices of basis vectors and mixture coefficients have respectively zero rows and zero columns. It returns FALSE otherwise. This means that an empty model can still have a non-zero number of basis vectors. For example, this happens in the case of NMF models created using factory method nmfModel with no initialisation for any factor matrices.

metaHeatmap Produces a heatmap of the basis or mixture matrix using function heatmap.2 with parameters tuned for displaying NMF results. See metaHeatmap for more details.

modelname**:** returns the name of the model fitted by the object. It corresponds to the name of the S4 class of the object.

nbasis signature(x = "NMF"): returns the number of basis vectors used in NMF model x. It is the number of columns of the matrix of basis vectors.

predict signature(object = "NMF"): returns a factor that gives the predicted cluster index for each sample (resp. for each feature) based on NMF factorization object. The index correspond to the basis vector that most contribute to the sample (resp. to which the feature contribute the most). See predict details on extra arguments.

purity signature(x = "NMF", class = "factor"): computes the purity of NMF model x given a priori known groups of samples. The purity definition can be found in *Kim and Park (2007)*. See references for more details.

rnmf signature(x = "NMF", target): seeds NMF model x with random values drawn from a uniform distribution. The result is a NMF model of the same class as x with basis and mixture matrices filled with random values.

Argument target can be either:

`numeric` it must be of length 2 (resp. 1), and give the dimension of the target matrix (resp. symmetric matrix) to fit. The result is a random NMF model.

`missing` it returns `rnmf(x, c(nrow(x), ncol(x)))`, that is a random NMF model with the same dimensions as defined in model `x`.

`matrix` it returns `rnmf(x, dim(target))`, that is a random NMF model that fits a matrix of the same dimension as `target`. The values are drawn within the interval `[0, max(target)]`.

This method's version with signature `(object='NMF', target='numeric')` might need to be overloaded if the initialisation of the specific NMF model requires setting values for data other than the basis and mixture matrices. The overloading methods must call the generic version using function `callNextMethod`.

rss `signature(object = "NMF")`: returns the Residual Sum of Squares (RSS) between the target matrix and its estimation by the NMF model `object`. *Hutchins et al. (2008)* used the variation of the RSS in combination with *Lee and Seung*'s algorithm to estimate the correct number of basis vectors. See `rss` for details on its usage.

sampleNames `signature(object = "NMF")`: returns the column names of the mixture coefficient matrix. If BioConductor is installed this method is defined for the generic function `sampleNames` from the `Biobase` package.

sampleNames<- `signature(object = "NMF", value = "ANY")`: sets the columns names of the basis matrix. Argument `value` must be in a format accepted by the `colnames` method defined for matrices. If BioConductor is installed this method is defined for the generic function `sampleNames<-` from the `Biobase` package.

show `signature(object = "NMF")`: standard generic `show` method for objects of class `NMF`. It displays the model class (i.e. the name of the sub-class that implements the concrete model), the dimension of the target matrix, and the number of basis vectors.

sparseness `signature(x = "NMF")`: compute the average sparseness of the basis vectors and mixture coefficients. See *Hoyer (2004)* for more details.

summary `signature(x = "NMF")`: standard generic `summary` method for objects of class `NMF`. It computes a set of measures to evaluate the quality of the factorization.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Definition of Nonnegative Matrix Factorization in its modern formulation:

Lee D.D. and Seung H.S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, **401**, 788–791.

Historical first definition and algorithms:

Paatero, P., Tapper, U. (1994). Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, **2**, 111–126 , doi:10.1002/env.3170050203.

Reference for some utility functions:

Kim, H. and Park, H. (2007). Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*.

Hoyer (2004). Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, **5**, 1457-1469.

See Also

Main interface to perform NMF in `nmf-methods`.

Built-in NMF models and factory method in `nmfModel`.

Method `seed` to set NMF objects with values suitable to start algorithms with.

Examples

```
# show all the NMF models available (i.e. the classes that inherit from class NMF)
nmfModel()
# show all the built-in NMF models available
nmfModel(builtin.only=TRUE)

# class NMF is a virtual class so cannot be instantiated:
# the following generates an error
## Not run: new('NMF')

# To instantiate a NMF model, use factory method nmfModel. see ?nmfModel
nmfModel(3)
nmfModel(3, model='NMFns')
```

`nmf-methods` *Main Interface to run NMF algorithms*

Description

This method implements the main interface to launch NMF algorithms within the framework defined in package `NMF`. It allows to combine NMF algorithms with seeding methods. The returned object can be directly passed to visualisation or comparison methods.

For a tutorial on how to use the interface, please see the package's vignette: `vignette('NMF')`

Usage

```
## S4 method for signature 'matrix,numeric,function':
nmf(x, rank, method, name, objective='euclidean', model='NMFstd'
, mixed=FALSE, ...)

## S4 method for signature 'matrix,numeric,character':
nmf(x, rank, method, ...)

## S4 method for signature 'matrix,numeric,NMFStrategy':
```

```
nmf(x, rank, method, seed=nmf.getOption('default.seed')
, nrun=1, model=NULL, .options=list()
, .pbackend = nmf.getOption("parallel.backend")
, ...)
```

Arguments

method	The algorithm to use to perform NMF on x. Different formats are allowed: character, function. If missing, the method to use is retrieved from the NMF package's specific options by nmf.getOption("default.algorithm") (the default built-in option is 'brunet'). See section *Methods* for more details on how each format is used.
mixed	Boolean that states if the algorithm requires a nonnegative input matrix (mixed=FALSE which is the default value) or accepts mixed sign input matrices (mixed=TRUE). An error is thrown if the sign required is not fulfilled. This parameter is useful to plug-in algorithms such as semi-NMF, that typically does not impose nonnegativity constraints on both the input and the basis component matrices.If NULL then the NMF model is used
model	When method is a function, argument model must be either a single character string (default to 'NMFstd') or a list that specifies values for slots in the NMF model. The NMF model to be instantiated can optionally be given by its class name in the first element of the list [note: A NMF model is defined by a S4 class that extends class NMF]. If no class name is specified then the default model is used, see NMFstd. When method is a single character string or a NMFStrategy object, argument model must be NULL (default) or a list. Note that in this case the NMF model is defined by the NMF strategy itself and cannot be changed. • If a single character string, argument model must be the name of the class that defines the NMF model to be instantiated. Arguments in ... are handled in the same way as when model is NULL, see below. • If a list all and only its elements are used to initialise the NMF model's slots. • if NULL, the arguments in ... that have the same name as slots in the NMF model associated with the NMF strategy of name method are used to initialise these slots. **Important:** Values to initialise the NMF model's slots can be passed in .... However, if argument model is a list – even empty – then all and only its elements are used to initialise the model, those in ... are directly passed to the algorithm. So to pass a parameter to the NMF algorithm, that has the same name as a slot in the NMF model, argument model MUST be a list – possibly empty – and contains all the values one wants to use for the NMF model slots. If a variable appears in both argument model and ..., the former will be used to initialise the NMF model, the latter will be passed to the NMF algorithm. See code examples for an illustration of this situation.
name	A character string to be used as a name for the custom NMF algorithm.

`nrun` Used to perform multiple runs of the algorithm. It specifies the number of runs to perform . This argument is useful to achieve stability when using a random seeding method.

`objective` Used when `method` is a `function`. It must be A `character` string giving the name of a built-in distance method or a `function` to be used as the objective function. It is used to compute the residuals between the target matrix and its NMF estimate.

`.options` this argument is used to set some runtime options. It can be `list` containing the named options and their values, or, in the case only boolean options need to be set, a character string that specifies which options are turned on or off. The string must be composed of characters that correspond to a given option. Characters '+' and '-' are used to explicitly specify on and off respectively. E.g. `.options='tv'` will toggle on options `track` and `verbose`, while `.options='t-v'` will toggle on option `track` and off option `verbose`. Note that '+' and '-' apply to all option character found after them. The default behaviour is to assume that `.options` starts with a '+'.

The following options are available (note the characters that correspond to each option, to be used when `.options` is passed as a string):

debug - d Toggle debug mode. Like option `verbose` but with more information displayed.

keep.all - k used when performing multiple runs (`nrun`>1): if toggled on, all factorizations are saved and returned, otherwise only the factorization achieving the minimum residuals is returned.

parallel - p this option is useful on multicore *nix or Mac machine only, when performing multiple runs (`nrun` > 1). If toggled on, the runs are performed using the parallel backend defined in argument `.pbackend`. If this is set to `'mc'` then one tried to perform the runs using multiple cores with package `link[package:doMC]{doMC}` – which therefore needs to be installed.

Unlike option 'P' (capital 'P'), if the computation cannot be performed in parallel, then it will still be carried on sequentially.

IMPORTANT NOTE FOR MAC OS X USERS: The parallel computation is based on the `doMC` and `multicore` packages, so the same care should be taken as stated in the vignette of `doMC`: *"it is not safe to use doMC from R.app on Mac OS X. Instead, you should use doMC from a terminal session, starting R from the command line."*

parallel.required - P Same as `p`, but an error is thrown if the computation cannot be performed in parallel.

restore.seed - r used when seeding the NMF computation with a numeric seed. When `TRUE` (default) the random seed (`.Random.seed`) is restored to its value as before the call to the `nmf` function.

track - t enables (resp. disables) error tracking. When `TRUE`, the returned object's slot `residuals` contains the trajectory of the objective values. This tracking functionality is available for all built-in algorithms.

verbose - v Toggle verbosity. If on, messages about the configuration and the state of the current run(s) are displayed.

`.pbackend` define the parallel backend (from the `foreach` package) to use when running in parallel mode. See options `p` and `P` in argument `.options`. Currently it accepts the following values: `'mc'` or a number that specifies the number of cores to use, `'seq'` or `NULL` to use sequential backend.

`rank` The factorization rank to achieve [i.e a single positive `numeric`]

`seed` The seeding method to use to compute the starting point passed to the algorithm. See section *Seeding methods* for more details on the possible classes and types for argument `seed`.

`x` The target object to estimate. It can be a `matrix`, a `data.frame`, an `ExpressionSet` object (this requires the `Biobase` package to be installed). See section *Methods* for more details.

`...` Extra parameters passed to the NMF algorithm's `run` method or used to initialise the NMF model slots. If argument `model` is not supplied as a `list`, ANY of the arguments in `...` that have the same name as slots in the NMF model to be instantiated will be used to initialise these slots. See also the *Important* paragraph in argument `model`.

Value

The returned value depends on the run mode:

`Single run:` An object that inherits from class `NMF`.

`Multiple runs, single method:`
When `nrun > 1` and `method` is NOT a `list`, this method returns an object of class `NMFfitX`.

`Multiple runs, multiple methods:`
When `nrun > 1` and `method` is a `list`, this method returns an object of class `NMFList`.

Methods

x = "matrix", rank = "numeric", method = "list" Performs NMF on matrix `x` for each algorithm defined in the list `method`.

x = "data.frame", rank = "ANY", method = "ANY" Performs NMF on a `data.frame`: the target matrix is the converted data.frame `as.matrix(x)`

x = "ExpressionSet", rank = "ANY", method = "ANY" Performs NMF on an `ExpressionSet`: the target matrix is the expression matrix `exprs(x)`.
This method requires the `Biobase` package to be installed. Special methods for bioinformatics are provided in an optional layer, which is automatically loaded when the `Biobase` is installed. See `NMF-bioc`.

x = "matrix", rank = "numeric", method = "character" Performs NMF on a `matrix` using an algorithm whose name is given by parameter `method`. The name provided must partially match the name of a registered algorithm. See section *Algorithms* below or the package's vignette.

x = "matrix", rank = "numeric", method = "function" Performs NMF using a custom algorithm defined by a `function`. It must have signature `(x=matrix, start=NMF, ...)` and return an object that inherits from class `NMF`. It should use its argument `start` as a starting point.

NMF Algorithms

The following algorithms are available:

`lee` Standard NMF. Based on euclidean distance, it uses simple multiplicative updates. See *Lee and Seung (2000)*.

`brunet` Standard NMF. Based on Kullbach-Leibler divergence, it uses simple multiplicative updates from *Lee and Seung (2000)*, enhanced to avoid numerical underflow. See *Brunet et al. (2004)*.

`lnmf` Local Nonnegative Matrix Factorization. Based on a regularized Kullbach-Leibler divergence, it uses a modified version of Lee and Seung's multiplicative updates. See *Li et al. (2001)*.

`nsNMF` Nonsmooth NMF. Uses a modified version of Lee and Seung's multiplicative updates for Kullbach-Leibler divergence to fit a extension of the standard NMF model. It is meant to give sparser results. See *Pascual-Montatno et al. (2006)*.

`offset` Uses a modified version of Lee and Seung's multiplicative updates for euclidean distance, to fit a NMF model that includes an intercept. See *Badea (2008)*.

`pe-nmf` Pattern-Expression NMF. Uses multiplicative updates to minimize an objective function based on the Euclidean distance and regularized for effective expression of patterns with basis vectors. See *Zhang et al. (2008)*.

`snmf/r`, `snmf/l` Alternating Least Square (ALS) approach from *Kim and Park (2007)*.

Seeding methods

The purpose of seeding methods is to compute initial values for the factor matrices in a given NMF model. This initial guess will be used as a starting point by the chosen NMF algorithm.

The seeding method to use in combination with the algorithm can be passed to interface `nmf` through argument `seed`. Detailed examples of how to specify the seeding method and its parameters can be found in the *Examples* section of this man page and in the package's vignette.

Argument `seed` accepts the following formats:

a `character` **string:** giving the name of a *registered* seeding method. The corresponding method will be called to compute the starting point.

a `list`**:** giving the name of a *registered* seeding method and, optionally, extra parameters to pass to it.

a single `numeric`**:** that is used to seed the random number generator. The value will be used in a call to the `set.seed` function before computing a starting point with the 'random' seeding method.

an object that inherits from `NMF`**:** it should contain the data of an initialised NMF model, that is it must contain valid basis and mixture coefficient matrices. It will be directly passed to the algorithm's method – via its argument `seed`.

a `function`**:** that computes the starting point. It must have signature `(object=NMF, target=matrix, ...)` and return an object that inherits from class `NMF`. Argument `object` should be used as a template for the returned object.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Lee, D.~D. and Seung, H.~S. (2000). Algorithms for non-negative matrix factorization. In *NIPS*, 556–562.

Brunet, J.~P., Tamayo, P., Golub, T.~R., and Mesirov, J.~P. (2004). Metagenes and molecular pattern discovery using matrix factorization. *Proc Natl Acad Sci U S A*, **101**(12), 4164–4169.

Pascual-Montano, A., Carazo, J.~M., Kochi, K., Lehmann, D., and Pascual-Marqui, R.~D. (2006). Nonsmooth nonnegative matrix factorization (nsnmf). *IEEE transactions on pattern analysis and machine intelligence*, **8**(3), 403–415.

Kim, H. and Park, H. (2007). Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*.

Liviu Badea (2008). Extracting Gene Expression Profiles Common To Colon And Pancreatic Adenocaricinoma Using Simultaneous Nonnegative Matrix Factorization. In *Pacific Symposium on Biocomputing*, **13**, 279–290

S. Li, X. Hou, and H. Zhang (2001). Learning spatially localized, parts-based representation. *In Proc. CVPR*, 2001.

Zhang J, Wei L, Feng X, Ma Z, Wang Y (2008). Pattern expression nonnegative matrix factorization: algorithm and applications to blind source separation. *Computational intelligence and neuroscience*

See Also

class NMF, NMF-utils, package's vignette

Examples

```
## DATA
# generate a synthetic dataset with known classes: 100 features, 23 samples (10+5+8)
n <- 100; counts <- c(10, 5, 8); p <- sum(counts)
V <- syntheticNMF(n, counts, noise=TRUE)
dim(V)

# build the class factor
groups <- as.factor(do.call('c', lapply(seq(3), function(x) rep(x, counts[x]))))

## RUN NMF ALGORITHMS

# run default algorithm
res <- nmf(V, 3)
res
summary(res, class=groups)

# run default algorithm multiple times (only keep the best fit)
res <- nmf(V, 3, nrun=10)
```

```
res
summary(res, class=groups)

# run default algorithm multiple times keeping all the fits
res <- nmf(V, 3, nrun=10, .options='k')
res
summary(res, class=groups)

## Not run:
## Note: one could have equivalently done
res <- nmf(V, 3, nrun=10, .options=list(keep.all=TRUE))

## End(Not run)

# run nonsmooth NMF algorithm
res <- nmf(V, 3, 'nsNMF')
res
summary(res, class=groups)

## Not run:
## Note: partial match also works
nmf(V, 3, 'ns')

## End(Not run)

## Not run:
# Non default values for the algorithm's parameters can be specified in '...'
res <- nmf(V, 3, 'nsNMF', theta=0.8)

## End(Not run)

# compare some NMF algorithms (tracking the residual error)
res <- nmf(V, 3, list('brunet', 'lee', 'nsNMF'), seed=123456, .opt='t')
res
summary(res, class=groups)
# plot the track of the residual errors
## Not run: plot(res)

# run on an ExpressionSet (requires package Biobase)
## Not run:
data(esGolub)
nmf(esGolub, 3)

## End(Not run)

## USING SEEDING METHODS

# run default algorithm with the Non-negative Double SVD seeding method ('nndsvd')
nmf(V, 3, seed='nndsvd')

## Not run:
## Note: partial match also works
nmf(V, 3, seed='nn')
```

```
## End(Not run)

# run nsNMF algorithm, fixing the seed of the random number generator
nmf(V, 3, 'nsNMF', seed=123456)

# run default algorithm specifying the starting point following the NMF standard model
start.std <- nmfModel(W=matrix(0.5, n, 3), H=matrix(0.2, 3, p))
nmf(V, seed=start.std)

# to run nsNMF algorithm with an explicit starting point, this one
# needs to follow the 'NMFns' model:
start.ns <- nmfModel(model='NMFns', W=matrix(0.5, n, 3), H=matrix(0.2, 3, p))
nmf(V, seed=start.ns)
# Note: the method name does not need to be specified as it is infered from the
# when there is only one algorithm defined for the model.

# if the model is not appropriate (as defined by the algorihtm) an error is thrown
# [cf. the standard model doesn't include a smoothing parameter used in nsNMF]
## Not run: nmf(V, method='ns', seed=start.std)
```

NMF-utils *Class and Utility Methods for NMF objects*

Description

Define generic interface methods for class NMF, which is the base – virtual – class of the results from any NMF algorithms implemented within package NMF's framework.

Usage

```
## S4 method for signature 'NMF':
connectivity(x, ...)

## S4 method for signature 'NMF,factor':
entropy(x, class, ...)

## S4 method for signature 'NMF':
evar(object, target)

## S4 method for signature 'NMFfit':
residuals(object, track=FALSE, ...)

rss(object, ...)
## S4 method for signature 'NMF':
rss(object, target)
```

```
## S4 method for signature 'NMF':
featureScore(object, method=c('kim', 'max'))

## S4 method for signature 'NMF':
extractFeatures(object, method=c('kim', 'max')
, format=c('list', 'combine', 'subset'))

## S4 method for signature 'NMF':
metaHeatmap(object, what=c('samples', 'features'), filter=FALSE, ...)

## S4 method for signature 'NMF':
nmfApply(object, MARGIN, FUN, ...)

## S4 method for signature 'NMFfit':
plot(x, ...)

## S4 method for signature 'NMF':
predict(object, what = c('samples', 'features'), prob=FALSE)

## S4 method for signature 'NMF,factor':
purity(x, class, ...)

randomize(x, ...)

## S4 method for signature 'NMF':
sparseness(x)

syntheticNMF(n, r, p, offset=NULL, noise=FALSE, return.factors=FALSE)
```

Arguments

class	A factor giving a known class membership for each sample. In methods entropy and purity, argument class is coerce to a factor if necessary.
filter	if TRUE, only the features that are basis-specific are used. Those features are those returned by function extractFeatures.
format	the output format of the extracted features. Possible values are: • list (default) a list with one element per basis vector, each containing the indices of the basis-specific features. • combine a single integer vector containing the indices of the basis-specific features for ALL the basis. • subset the object object subset to contain only the basis-specific features.
FUN	the function to be applied: see 'Details'. In the case of functions like +, %*%, etc., the function name must be backquoted or quoted. See link[base]{apply} for more details.

MARGIN — a vector giving the subscripts which the function will be applied over. 1 indicates rows, 2' indicates columns, c(1,2) indicates rows and columns. See link[base]{apply} for more details.

method — Method used to compute the feature scores and selecting the features.

Possible values are:

- kim (default) to use Kim and Park (2007) scoring schema and feature selection method. The features are first scored using the function featureScore. Then only the features that fulfil both following criteria are retained:
  - score greater than $\hat{\mu} + 3\hat{\sigma}$, where $\hat{\mu}$ and $\hat{\sigma}$ are the median and the median absolute deviation (MAD) of the scores respectively;
  - the maximum contribution to a basis component is greater than the median of all contributions (i.e. of all elements of W)
  See *Kim and Park (2007)*.
- max where the score is the maximum contribution of each feature to the basis vectors and the selection method is the one described in *Carmona-Saez (2006)*. Briefly, for each basis vector, the features are first sorted in descending order by their contribution to the basis vector. Then, one selects only the first consecutive features from the sorted list whose highest contribution in the basis matrix is found in the considered basis (see section *References*).

n — Number of rows of the synthetic target matrix.

noise — if TRUE, a random noise is added the target matrix.

object — A matrix or an object that inherits from class NMF or NMFfit – depending on the method.

offset — a vector giving the offset to add to the synthetic target matrix. Its length should be equal to the number of rows n.

prob — Should the probability associated with each cluster prediction be computed and returned.

p — Number of columns of the synthetic target matrix. Not used if parameter r is a vector (see description of argument r).

r — Underlying factorization rank. If a single numeric is given, the classes are randomly generated from a multinomial distribution. If a numerical vector is given, then it should contain the counts in the different classes (i.e integers). In such a case argument p is not used and the number of columns is forced to be the sum of the counts.

return.factors — If TRUE, the underlying matrices W and H are also returned.

target — the target object estimated by model object. It can be a matrix or an ExpressionSet.

track — if TRUE, the whole residuals track is returned. Otherwise only the last residuals computed is returned.

what — Specifies on which matrix (basis components or mixture coefficients) the computation should be performed.

`x` for `randomize`: the `matrix` or `ExpressionSet` object whose entries will be randomised.
for `plot`: An object that inherits from class `NMFfit`.
otherwise: An object that inherits from class `NMF`.

`...` Used to pass extra parameters to subsequent calls:

- in `metaHeatmap`: Graphical parameters passed to function `heatmap.2`
- in `nmfApply`: optional arguments to function `FUN`.
- in `randomize`: passed to the `sample` function.
- in `residuals`: not used.

Details

connectivity Computes the connectivity matrix for the samples based on their mixture coefficients.
The connectivity matrix of a clustering is a matrix C containing only 0 or 1 entries such that:

$$C_{ij} = \begin{cases} 1 \text{ if sample } i \text{ belongs to the same cluster as sample } j \\ 0 \text{ otherwise} \end{cases} .$$

entropy The entropy is a measure of performance of a clustering method, in recovering classes defined by factor a priori known (i.e. one knows the true class labels). Suppose we are given l categories, while the clustering method generates k clusters. Entropy is given by:

$$Entropy = -\frac{1}{n \log_2 l} \sum_{q=1}^{k} \sum_{j=1}^{l} n_q^j \log_2 \frac{n_q^j}{n_q}$$

, where:
- n is the total number of samples;
- n is the total number of samples in cluster q;
- n_q^j is the number of samples in cluster q that belongs to original class j ($1 \leq j \leq l$).

The smaller the entropy, the better the clustering performance.
See *Kim and Park (2007)*.

evar Computes the explained variance of the NMF model `object`.
For a target V It is defined as:

$$evar = 1 - \frac{RSS}{\sum_{i,j} v_{ij}^2}$$

,
where RSS is the residual sum of squares.
It is usefull to compare the performance of different models and their ability to accurately reproduce the original target matrix. Note that a possible caveat is that some methods explicitly aim at minimizing the RSS (i.e. maximizing the explained variance), while others do not.

extractFeatures Identify the most basis-specific features, using different methods. See details of argument `method`.

featureScore Computes the feature scores as suggested in *Kim and Park (2007)*.
The score for feature i is defined as:

$$S_i = 1 + \frac{1}{\log_2 k} \sum_{q=1}^{k} p(i,q) \log_2 p(i,q),$$

where $p(i,q)$ is the probability that the i-th feature contributes to basis q:

$$p(i,q) = \frac{W(i,q)}{\sum_{r=1}^{k} W(i,r)}$$

The feature scores are real values within the range [0,1]. The higher the feature score the more basis-specific the corresponding feature.

metaHeatmap Produces a heatmap of the basis components or mixture coefficients using a `heatmap`-like custom function, with parameters tuned for displaying NMF results.

The function used to draw the heatmap is a mixture of the function `heatmap.2` from the `gplots` package, and the function `heatmap.plus` from the `heatmap.plus` package. It allows to add extra annotation rows using the `ColSideColor` argument. See `heatmap.2` and `heatmap.plus`.

nmfApply `apply`-like method for objects of class `NMF`.

When argument `MARGIN=1`, it calls the base method `apply` to apply function `FUN` to the *rows* of the basis component matrix.

When `MARGIN=2`, it calls the base method `apply` to apply function `FUN` on the *columns* of the mixture coefficient matrix.

See `apply` for more details on the output format.

plot plots the residuals track of the run that computed object `x`. See function `nmf` for details on how to enable the tracking of residuals.

predict Computes the dominant basis component for each sample (resp. feature) based on its associated entries in the mixture coefficient matrix (i.e in H) (resp. basis component matrix (i.e in W)).

When `what='samples'` the computation is performed on the mixture coefficient matrix, or on the transposed basis matrix when `what='features'`.

For each column, the dominant basis component is computed as the row index for which the entry is the maximum within the column.

If argument `prob=FALSE` (default), the result is a `factor`. Otherwise it returns a list with two elements: element `predict` contains the computed indexes (as a `factor`) and element `prob` contains the vector of the associated probabilities, that is the relative contribution of the maximum entry within each column.

purity Computes the purity of a clustering given a known factor.

The purity is a measure of performance of a clustering method, in recovering the classes defined by a factor a priori known (i.e. one knows the true class labels). Suppose we are given l categories, while the clustering method generates k clusters. Purity is given by:

$$Purity = \frac{1}{n} \sum_{q=1}^{k} \max_{1 \leq j \leq l} n_q^j$$

, where:

- n is the total number of samples;

- n_q^j is the number of samples in cluster q that belongs to original class j ($1 \leq j \leq l$).

The purity is therefore a real number in $[0, 1]$. The larger the purity, the better the clustering performance.

See *Kim and Park (2007)*.

randomize permute the row entries within each column of `x`, using a different permutation for each column.

The extra arguments in `...` are passed to the `sample` function, and will be used for each column.

The result is a `matrix` of the same dimension as `x` (or `exprs(x)` in the case `x` is an `ExpressionSet` object.).

residuals returns the – final – residuals between the target matrix and the NMF result `object`. They are computed using the objective function associated to the NMF algorithm that returned `object`. When called with `track=TRUE`, the whole residuals track is returned, if available. Note that method `nmf` does not compute the residuals track, unless explicitly required.

It is a S4 methods defined for the associated generic functions from package `stats` (See residuals)

See `nmf` and `NMFfit`.

rss returns the Residual Sum of Squares (RSS) between the target object `target` and its estimation by the `object`. *Hutchins et al. (2008)* used the variation of the RSS in combination with *Lee and Seung*'s algorithm to estimate the correct number of basis vectors. The optimal rank is chosen where the graph of the RSS first shows an inflexion point. See references.

Note that this way of estimation may not be suitable for all models. Indeed, if the NMF optimization problem is not based on the Frobenius norm, the RSS is not directly linked to the quality of approximation of the NMF model.

sparseness Generic mathod that computes the sparseness of an object as defined in *Hoyer (2004)*.

This sparseness measure quantifies how much energy of a vector is packed into only few components. It is defined by:

$$Sparseness(x) = \frac{\sqrt{n} - \frac{\sum |x_i|}{\sqrt{\sum x_i^2}}}{\sqrt{n} - 1}$$

, where n is the length of `x`.

The sparseness is a real number in $[0, 1]$. It is equal to 1 if and only if `x` contains a single nonzero component, and is equal to 0 if and only if all components of `x` are equal. It interpolates smoothly between these two extreme values. The closer to 1 is the sparseness the sparser is the vector.

The basic definition is for a `numeric` vector. The sparseness of a `matrix` is the mean sparseness of its column vectors. The sparseness of an object of class `NMF`, is the a 2-length vector that contains the sparseness of the basis and mixture coefficient matrices.

syntheticNMF Generate a synthetic matrix according to an underlying NMF model. It can be used to quickly test NMF algorithms.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Metagenes and molecular pattern discovery using matrix factorization Brunet, J.~P., Tamayo, P., Golub, T.~R., and Mesirov, J.~P. (2004) Proc Natl Acad Sci U S A 101(12), 4164–4169.

Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis Kim, H. & Park, H. (2007) Bioinformatics. http://dx.doi.org/10.1093/bioinformatics/btm134.

Non-negative Matrix Factorization with Sparseness Constraints Hoyer, P. O. (2004) *Journal of Machine Learning Research* 5 (2004) 1457–1469

Biclustering of gene expression data by non-smooth non-negative matrix factorization Carmona-Saez, Pedro and Pascual-Marqui, Roberto and Tirado, F and Carazo, Jose and Pascual-Montano, Alberto (2006) BMC Bioinformatics 7(1), 78

See Also

NMF, summary

Examples

```
# generate a synthetic dataset with known classes: 50 features, 18 samples (5+5+8)
n <- 50; counts <- c(5, 5, 8);
V <- syntheticNMF(n, counts, noise=TRUE)
## Not run: metaHeatmap(V)

# build the class factor
groups <- as.factor(do.call('c', lapply(seq(3), function(x) rep(x, counts[x]))))

# perform default NMF
res <- nmf(V, 2)
res

## Not run: metaHeatmap(res, class=groups)
## Not run: metaHeatmap(res, 'features')
# see the predicted clusters of samples
predict(res)
# compute entropy and purity
entropy(res, class=groups)
purity(res, class=groups)

# perform NMF with the right number of basis components
res <- nmf(V, 3)

## Not run: metaHeatmap(res)
## Not run: metaHeatmap(res, 'features')
entropy(res, class=groups)
purity(res, class=groups)
```

nmfEstimateRank *Estimate optimal rank for Nonnegative Matrix Factorization (NMF) models*

Description

A critical parameter in NMF algorithms is the factorization rank r. It defines the number of basis effects used to approximate the target matrix. Function nmfEstimateRank helps in choosing an optimal rank by implementing simple approaches proposed in the litterature.

Usage

```
nmfEstimateRank(x, range, method = nmf.getOption("default.algorithm"), nrun = 30, v
plot.NMF.rank(x, what = c('all', 'cophenetic', 'rss', 'residuals'
, 'dispersion', 'evar', 'sparseness'
, 'sparseness.basis', 'sparseness.coef')
, ref=NULL, ...)
```

Arguments

method	A single NMF algorithm, in one of the format accepted by interface nmf.
nrun	a numeric giving the number of run to perform for each value in range.
range	a numeric vector containing the ranks of factorization to try.
ref	reference object of class NMF.rank, as returned by function nmfEstimateRank. The measures contained in ref are used and plotted as a reference. The associated curves are drawn in *red*, while those from x are drawn in *blue*.
verbose	toggle verbosity.
what	a character string that partially matches one of the following item: 'all', 'cophenetic', 'rss', 'residuals' , 'dispersion'. It specifies which measure must be plotted (what='all' plots all the measures).
x	For nmfEstimateRank a target object to be estimated, in one of the format accepted by interface nmf. For plot.NMF.rank an object of class NMF.rank as returned by function nmfEstimateRank.
...	For nmfEstimateRank, these are extra parameters passed to interface nmf. Note that the same parameters are used for each value of the rank. See nmf. For plot.NMF.rank, these are extra graphical parameter passed to the standard function plot. See plot.

Details

Given a NMF algorithm and the target matrix, a common way of estimating r is to try different values, compute some quality measures of the results, and choose the best value according to this quality criteria. See *Brunet et al. (2004)* and *Hutchins et al. (2008)*.

The function `nmfEstimateRank` allow to launch this estimation procedure. It performs multiple NMF runs for a range of rank of factorization and, for each, returns a set of quality measures together with the associated consensus matrice.

Value

A S3 object (i.e. a list) of class `NMF.rank` with the following slots:

`measures`	a `data.frame` containing the quality measures for each rank of factorizations in `range`. Each row correspond to a measure, each column to a rank.
`consensus`	a `list` of consensus matrices, indexed by the rank of factorization (as a character string).

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Metagenes and molecular pattern discovery using matrix factorization Brunet, J.~P., Tamayo, P., Golub, T.~R., and Mesirov, J.~P. (2004) Proc Natl Acad Sci U S A 101(12), 4164–4169.

See Also

nmf

Examples

```
set.seed(123456)
n <- 50; r <- 3; m <- 20
V <- syntheticNMF(n, r, m, noise=TRUE)

# Use a seed that will be set before each first run
## Not run: res.estimate <- nmfEstimateRank(V, seq(2,5), method='brunet', nrun=10, seed=1234

# plot all the measures
## Not run: plot(res.estimate)
# or only one: e.g. the cophenetic correlation coefficient
## Not run: plot(res.estimate, 'cophenetic')
```

NMFfit-class *Base Class for to store Nonnegative Matrix Factorisation results*

Description

Base class to handle the results of general **Non-negative Matrix Factorisation** algorithms (NMF).

It provides a general structure and generic functions to manage the results of NMF algorithms. It contains a slot with the fitted NMF model (see slot `fit`) as well as data about the methods and parameters used to compute the factorization.

Details

The purpose of this class is to handle in a generic way the results of NMF algorithms. Its slot `fit` contains the fitted NMF model as an object of class `NMF`.

Other slots contains data about how the factorization has been computed, such as the algorithm and seeding method, the computation time, the final residuals, etc...

Class `NMFfit` acts as a wrapper class for its slot `fit`. It inherits from interface class `NMF` defined for generic NMF models. Therefore, all the methods defined by this interface can be called directly on objects of class `NMFfit`. The calls are simply dispatched on slot `fit`, i.e. the results are the same as if calling the methods directly on slot `fit`.

Slots

`fit`**:** An object that inherits from class `"NMF"`. It contains the fitted NMF model.
Note that class `"NMF"` is a virtual class. The default class for this slot is `NMFstd`, that implements the standard NMF model.

`residuals`**:** A `"numeric"` vector that contains the final residuals or the residuals track between the target matrix and its NMF estimate(s). Default value is `numeric()`.
See method `residuals` for details on accessor methods and main interface `nmf` for details on how to compute NMF with residuals tracking.

`method`**:** A single `"character"` string that contains the name of the algorithm used to fit the model. Default value is `''`.

`seed`**:** A single `"character"` string that contains the name of the seeding method used to seed the algorithm that computed the NMF. Default value is `''`. See `nmf-methods` for more details.

`distance`**:** Either a single `"character"` string that contains the name of the built-in objective function, or a `function` that measures the residuals between the target matrix and its NMF estimate.

`parameters`**:** A `"list"` that contains the extra parameters specific to the algorithm used to fit the model.

`runtime`**:** Object of class `"proc_time"` that contains various measures of the time spent to fit the model. See `system.time`

`options`**:** A `"list"` that contains the options used to compute the object.

`extra`**:** A `"list"` that contains extra miscellaneous data set by the algorithm. For example it can be used to store extra parameters, without the need to extend the standard NMF model.

Validity checks

The validity method for class NMFfit checks

- slot fit calling the suitable validity function on this object of class NMF (see NMF for more details).
- the validity of slot objective that must be either a function definition or a *non-empty* character string.

Objects from the Class

Object of class NMFfit using the standard operator new.

However, there is usually no need to directly create such an object, as interface methods nmf and seed take care of it.

Methods

Class-specific methods:

algorithm signature(object = "NMFfit"): Access slot method.

algorithm<- signature(object = "NMFfit", value="ANY"): Set slot method.

basis signature(object = "NMFfit"): Extract the matrix of basis vectors from the fitted NMF model. It returns basis(fit(object)).
Note that this method is part of the minimum interface for NMF models, as defined by class NMF. See NMF.

basis<- signature(object = "NMFfit", value = "matrix": Sets the matrix of basis vectors of the fitted NMF model. It calls basis(fit(object), value).
Note that this method is part of the minimum interface for NMF models, as defined by class NMF. See NMF.

coef signature(object = "NMFfit"): Extract the matrix of mixture coefficients from the fitted NMF model. It returns coef(fit(object)).
Note that this method is part of the minimum interface for NMF models, as defined by class NMF. See NMF.

coef<- signature(object = "NMFfit", value = "matrix": Sets the matrix of mixture coefficients of the fitted NMF model. It calls coef(fit(object), value).
Note that this method is part of the minimum interface for NMF models, as defined by class NMF. See NMF.

distance signature(target = "matrix", x = "NMFfit"): return the value of the loss function given a target matrix and a NMF fit. It calls method distance on slot fit. If a distance method is NOT supplied in argument method, it uses the objective function from slot objective.

plot signature(x = "NMFfit"): plot the residuals track of the run that computed object x. See function nmf for details on how to enable the tracking of residuals.

fitted signature(object = "NMFfit"): compute the estimated target matrix according to the NMF model stored in slot fit. It actually dispatches the call to slot fit, returning fitted(fit(object)).
Note that this method is part of the minimum interface for NMF models, as defined by class NMF. See NMF.

fit `signature(object = "NMFfit")`: return the `NMF` object stored in slot `fit`.

fit<- `signature(object = "NMFfit", value = "NMF")`: set the value of the `NMF` object stored in slot `fit`.

`modelname`**:** returns the name of the model fitted by the object. It corresponds to the name of the S4 class of slot `fit`.

nrun `signature(object = "NMFfit")`: return the number of NMF runs performed to compute `object`. In the case of a `NMFfit` object, this method always returns 1, as it is the result of a single NMF run and the method exists to create a uniform access interface to NMF results.

objective `signature(object = "NMFfit")`: return slot `distance` or compute the objective value if a target is passed in argument `x`.

objective<- `signature(object = "NMFfit", value = "character")`: sets slot `distance` to a built-in distance metric identified by a `character` string.

objective<- `signature(object = "NMFfit", value = "function")`: sets slot `distance` to a custom `function`. The function should return a single positive `numeric` and must take the target (a `matrix`) as its first parameter, and an object that inherits from class `NMF` as its second parameter. Extra parameters are passed.

residuals `signature(object = "NMFfit")`: Access slot `residuals`. See function `residuals` for details on extra parameters.

residuals<- `signature(object = "NMFfit", value="numeric")`: Set slot `residuals` to value `value`.

runtime `signature(object = "NMFfit")`: return the time spent to fit NMF model `object`. The time is computed using base function `system.time` which returns object of class `proc_time`.

runtime.all `signature(object = "NMFfit")`: return the CPU time used to perform all the runs to compute the object. In the case of a `NMFfit` object, this method is an alias to method `runtime` (see above), as the object is the result of a single NMF run. The method exists to create a uniform access interface to NMF results.

seeding `signature(object = "NMFfit")`: returns the seeding method used to seed the algorithm that fitted NMF model `object`. See section `nmf-methods`.

seeding<- `signature(object = "NMFfit")`: sets the seeding method used to seed the algorithm that fitted NMF model `object`.

show `signature(object = "NMFfit")`: standard generic `show` method for objects of class `NMF`.

summary `signature(x = "NMFfit")`: standard generic `summary` method for objects of class `NMF`. It returns a `numeric` vector that contains the summary of the fitted NMF model (slot `fit`), plus the computation time and the final residuals.

$ `signature(x = "NMFfit", name)`: Access element `name` in slot `extra` (which is a `list`) partially matching argument `name`. It is equivalent to `x@extra[[name, exact=FALSE]]`.

$<- `signature(x = "NMFfit", name, value)`: Set the value in of element `name` in slot `extra` (which is a `list`). Note that it does not partially match argument `name`.

Class `NMFfit` inherits from all the methods defined on class `NMF` to manipulate and interpret NMF models. For those methods, class `NMFfit` act as a wrapper class, dispatching the calls to slot `fit`. Some useful methods are: `dim`, `nbasis`, `predict`, `sparseness`. See `NMF` for more details.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

See Also

Main interface to perform NMF in nmf-methods.

Method seed to set NMF objects with values suitable to start algorithms with.

Examples

```
# run default NMF algorithm on a random matrix
n <- 50; r <- 3; p <- 20
V <- matrix(runif(n*p), n, p)
res <- nmf(V, r)

# result class is NMFfit
class(res)

# show result
res

# compute summary measures
summary(res)
```

NMFfitX-class *Virtual Class to Handle Results from Multiple Runs of a NMF Algorithms*

Description

This class defines a common interface to handle the results from multiple runs of a single NMF algorithm, performed with the nmf method.

Currently, this interface is implemented by two classes, NMFfitX1 and NMFfitXn, which respectively handle the case where only the best fit is kept, and the case where the list of all the fits is returned.

See nmf-multiple for more details on the method arguments.

Slots

runtime.all: Object of class "proc_time" that contains various measures of the time spent to perform all the runs.

Methods

consensus :

Computes the consensus matrix associated to the multiple NMF runs described by `object`. It's been proposed by *Brunet et al. (2004)* to help visualising and measuring the stability of the clusters obtained by NMF approaches. See `consensus`.

Technical note: this method is defined as a pure virtual method is the sense that an error is thrown if it is not overloaded by the classes that implement the interface (i.e. that extends class `NMFfitX`*).*

cophcor :

Computes the cophenetic correlation coefficient of the consensus matrix associated to the multiple NMF runs described by the object. It's been proposed by *Brunet et al. (2004)* to measure the stability of the clusters obtained by NMF approaches. See `cophcor` for more details.

dispersion :

Computes the dispersion coefficient of the consensus matrix associated to the multiple NMF runs described by the object. It's been proposed by *Kim and Park (2007)* to measure the reproducibility of the clusters. See `dispersion` for more details.

featureNames : returns the row names of the basis matrix from the best fit of the set of results.

If BioConductor is installed this method is defined for the generic function `featureNames` from the `Biobase` package.

fit : returns the element that achieves the lowest residual approximation error across all the runs. See `fit` for more details.

Technical note: this method is defined as a pure virtual method is the sense that an error is thrown if it is not overloaded by the classes that implement the interface (i.e. that extends class `NMFfitX`*).*

nrun : returns the number of runs performed to create the object.

Note that because the `nmf` method allows to run the NMF computation keeping only the best fit, `nrun` may return a value greater than one, while only the result of the best run is stored in the object (cf. option `'k'` in method `nmf`).

See `nmf` and `NMFfitX1`.

Technical note: this method is defined as a pure virtual method is the sense that an error is thrown if it is not overloaded by the classes that implement the interface (i.e. that extends class `NMFfitX`*).*

metaHeatmap Produces a heatmap of the consensus matrix using function `heatmap.2`. See `metaHeatmap`.

runtime.all : returns the total time spent to compute all the runs. See `runtime.all` for more details.

sampleNames : returns the column names of the mixture coefficient matrix from the best fit of the set of results. If BioConductor is installed this method is defined for the generic function `featureNames` from the `Biobase` package.

show : `show` method for objects of class `NMFfitX`.

summary : standard generic `summary` method for objects of class `NMFfitX`. It computes a set of measures to evaluate the quality of the *best factorization* of the set. The result is similar to the result from the `summary` method of `NMFfit` objects. See `NMFfit` for details on

the computed measures. In addition, the cophenetic correlation coefficient and the dispersion coefficient of the consensus matrix are returned. See methods `cophcor` and `dispersion` above.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Metagenes and molecular pattern discovery using matrix factorization Brunet, J.~P., Tamayo, P., Golub, T.~R., and Mesirov, J.~P. (2004) Proc Natl Acad Sci U S A 101(12), 4164–4169.

Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis Kim, H. & Park, H. (2007) Bioinformatics. http://dx.doi.org/10.1093/bioinformatics/btm134.

See Also

nmf-methods, nmf-multiple, NMFfitX1, NMFfitXn

Examples

```
# generate a synthetic dataset with known classes
n <- 50; counts <- c(5, 5, 8);
V <- syntheticNMF(n, counts, noise=TRUE)

# perform multiple runs of one algorithm (default is to keep only best fit)
res <- nmf(V, 3, nrun=5)
str(res)

# perform multiple runs of one algorithm (keep all the fits)
res <- nmf(V, 3, nrun=5, .options='k')
str(res)
```

NMFfitX1-class *Class to Store the Result from Multiple Runs of a NMF Algorithm when Only the Best Fit is Kept*

Description

This class is used to return the result from a multiple run of a single NMF algorithm performed with function `nmf` with the – default – option `keep.all=FALSE` (cf. nmf).

It extends both classes NMFfitX and NMFfit, and stores a the result of the best fit in its `NMFfit` structure.

Beside the best fit, this class allows to hold data about the computation of the multiple runs, such as the number of runs, the CPU time used to perform all the runs, as well as the consensus matrix.

Due to the inheritance from class NMFfit, objects of class NMFfitX1 can be handled exactly as the results of single NMF run – as if only the best run had been performed.

Slots

consensus: object of class "matrix" used to store the consensus matrix based on all the runs.

nrun: an integer that contains the number of runs performed to compute the object.

runtime.all: object of class "proc_time" that contains various measures of the time spent to perform all the runs (inherited from NMFfitX)

Extends

Class "NMFfitX", directly. Class "NMFfit", directly. Class "NMF", by class "NMFfit", distance 2.

Validity

There is currently no validity check for this class.

Methods

consensus : returns the pre-computed consensus matrix associated with the runs. It is calculated during the NMF computations itself, and is stored in slot consensus. If this slot is of length zero, then it returns NULL.

It's been proposed by *Brunet et al. (2004)* to help visualising and measuring the stability of the clusters obtained by NMF approaches. See consensus.

fit : returns the best fit as an NMFfit object.

nrun : returns the number of NMF runs performed to compute the object, as stored in slot nrun.

runtime.all returns the CPU time used to compute all the runs in the list, as stored in slot runtime.all (inherited from class NMFfitX).

show : show method for class NMFfitX1.

Besides these above methods, class NMFfitX1 inherits from all the methods from class NMFfit and as such can be handled exactly as the result of a single NMF run.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

See Also

NMFfitX, nmf-methods, nmf-multiple

Examples

```
# generate a synthetic dataset with known classes
n <- 50; counts <- c(5, 5, 8);
V <- syntheticNMF(n, counts, noise=TRUE)

# build the class factor
groups <- as.factor(do.call('c', lapply(seq(3), function(x) rep(x, counts[x]))))

# perform multiple runs of one algorithm, keeping only the best fit (default)
res <- nmf(V, 3, nrun=5)
res
#NOTE: the implicit nmf options are .options=list(keep.all=FALSE) or .options='-k'

# compute summary measures
summary(res)
# get more info
summary(res, target=V, class=groups)

# show computational time
runtime.all(res)
```

NMFfitXn-class *Class to Store the Result from Multiple Runs of a NMF Algorithm when Keeping All the Fits*

Description

This class is used to return the result from a multiple run of a single NMF algorithm performed with function `nmf` with option `keep.all=TRUE` (cf. `nmf`).

It extends both classes `NMFfitX` and `list`, and stores the result of each run (i.e. a `NMFfit` object) in its `list` structure.

IMPORTANT NOTE: This class is designed to be **read-only**, even though all the `list`-methods can be used on its instances. Adding or removing elements would most probably lead to incorrect results in subsequent calls. Capability for concatenating and merging NMF results is for the moment only used internally, and should be included and supported in the next release of the package.

Slots

`.Data`**:** standard slot that contains the S3 `list` object data. See R documentation on S4 classes for more details.

Extends

Class "`NMFfitX`", directly. Class "`list`", from data part. Class "`vector`", by class "list", distance 2. Class "`AssayData`", by class "list", distance 2.

Validity

NMFfitXn objects are designed to store results of single runs of the same NMF algorithm, that have the same dimensions. The following checks are performed in the class validity method:

- All elements must be the result of a single NMF run. That is they must be of class NMFfit, and objects of class NMFfitX are not allowed.
- All elements must be the result of the same NMF algorithm.
- The dimension of the fitted problem must be the same for all elements: same dimension of the target matrix, and same factorisation rank.

Methods

algorithm : returns the name of the common algorithm used to compute all the runs.

Since all elements in the list are results from the same algorithm, the returned name is taken from the first element. The method returns NULL if the list is empty.

consensus : Computes the consensus matrix associated with the list of runs, i.e. the mean connectivity matrix of all the fits in the list. It's been proposed by *Brunet et al. (2004)* to help visualising and measuring the stability of the clusters obtained by NMF approaches. See consensus.

dim : returns the common dimension of the NMF problem fitted by all the runs – based on the dimension of the first fit in the list. See dim,NMF-method for more details on the returned value.

entropy : computes the mean (resp. the best) entropy of the list of NMF fit. Which value is computed depends on argument method. See entropy for more details.

fit : returns the element of the list that achieves the lowest residual approximation error. See residuals.

nrun : returns the number of NMF runs performed to compute the object, i.e. its length – as a list.

purity : computes the mean (resp. the best) purity of the list of NMF fit. Which value is computed depends on argument method. See purity for details.

residuals : computes the mean (resp. the best) residual error of the list of NMF fit. Which value is computed depends on argument method. See residuals for details.

predict : returns the predicted cluster index based on the *best* NMF factorization in the list. See predict for more details.

runtime.all signature(object = "NMFfitXn", null=FALSE, warning=TRUE): returns the CPU time used to compute all the runs in the list, as stored in slot runtime.all (inherited from class NMFfitX).

When the computation is performed in parallel , the result may be very different from the sequential computation time returned by the seqtime method (see below).

When no time data is available in slot runtime.all, the runtime.all method for class NMFfitXn differs from the one defined in its parent class NMFfitX. Indeed, in the case no time data is stored in slot runtime.all, setting the extra argument null to FALSE (default) forces the method to return the sequential computation time instead and a warning is thrown unless argument warning is FALSE. Otherwise, in such a case, a call with null=TRUE would return NULL.

seqtime : returns the sequential CPU time spent of all the runs in the list. It is the sum of the CPU time used to compute each run. It returns `NULL` if the list is empty.

show : `show` method for class `NMFfitXn`.

Besides these above methods, class `NMFfitXn` inherits all the methods from class `NMFfitX` like: `summary`, `metaHeatmap`.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

See Also

`NMFfitX`, `nmf-methods`, `nmf-multiple`

Examples

```
# generate a synthetic dataset with known classes
n <- 50; counts <- c(5, 5, 8);
V <- syntheticNMF(n, counts, noise=TRUE)

# build the class factor
groups <- as.factor(do.call('c', lapply(seq(3), function(x) rep(x, counts[x]))))

# perform multiple runs of one algorithm, keeping all the fits
res <- nmf(V, 3, nrun=5, .options='k') # .options=list(keep.all=TRUE) also works
res
summary(res)
# get more info
summary(res, target=V, class=groups)

# compute/show computational times
runtime.all(res)
seqtime(res)
```

NMFList-class *Class "NMFList" to Handle the Comparison of NMF Results*

Description

Class "NMFList" is used to wrap into a list the results of different NMF runs with the objective to compare them.

While it handles indifferently any kind of NMF result, it is usually used to compare NMF results from different algorithms.

Objects from the Class

Objects can be created by calls of the form `as.NMFList(...)`.

Slots

`runtime`: Object of class `"proc_time"` that stores the CPU time used to compute all the results in the list.

`.Data`: Object of class `"list"`, inherited from class `list`.

Extends

Class `"list"`, from data part. Class `"vector"`, by class "list", distance 2. Class `"AssayData"`, by class "list", distance 2.

Methods

algorithm : returns the character vector of the names of the algorithms of the results in the list.

plot plots the error track of each result in the list onto a single plot. To work properly, the results must be computed with argument `.options='t'` or `.options=list(track=TRUE, ...)`. See `nmf-compare` for more details.

runtime : returns the CPU time used to compute all the results in the list, as stored in slot `runtime`. If no time data is available then it returns `NULL`.

show : show method for class `NMFList`.

summary returns a `data.frame` whose lines are the summary measures of each result in the list.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

See Also

`nmf-compare`, `nmf-multiple`, `nmf`

Examples

```
showClass("NMFList")
```

---

nmfModel - NMF Model Factory

Factory Method for NMF Models

Description

`nmfModel` is a generic function which provides a convenient way to build NMF models.

It provides a unique interface to create `NMF` objects that can follow different NMF models, and is designed to resolve potential inconsistencies in the matrices dimensions.

Details

NMF models are defined by S4 classes that inherit from class NMF.

nmfModel methods act as factory methods to help in the creation of NMF model objects in common situations: creating an empty model, a model with given dimensions, a model with dimensions compatible with a given target matrix, ...

All methods return an object that inherits from class NMF, except for the call with no argument, which lists the NMF models defined in the session (built-in and user-defined).

The returned NMF objects are suitable for seeding NMF algorithms via argument seed of the nmf method. In this case the factorisation rank is implicitly set by the number of columns of the basis vector matrix.

Methods

signature(rank = "missing", target = "missing", builtin.only=FALSE, ...)
: When called with no argument or argument builtin.only only, the method returns a character vector that contains the name of all the NMF models currently defined. These are the classes that inherits from class NMF, but not from NMFfit. If argument builtin.only=TRUE, only the models provided by the package itself are returned, discarding the user-defined models.

When called with extra arguments in ..., then the argument builtin.only is discarded and the method is equivalent to the call nmfModel(0, 0, ...). See the description of the appropriate method below.

signature(rank = "numeric", target = "numeric" , model='NMFstd', W, H, ...)
: This call creates an object of class model, using the extra parameters ... to initialise slots that are specific to the given model. All NMF models implement get/set methods to access the matrix factors, which can be initialised via arguments W and H. For example, all the built-in models derive from class NMFstd, which has two slots, W and H, to hold the two factors.

If only argument rank is provided, the method creates a NMF model of dimension 0xrankx0. That is that the basis and mixture coefficient matrices, W and H, have dimension 0xrank and rankx0 respectively.

If target dimensions are also provided in argument target, then the method creates a NMF object compatible to fit a target matrix of dimension target[1]xtarget[2] That is that the basis and mixture coefficient matrices, W and H, have dimension target[1]xrank and rankxtarget[2] respectively. If no other argument is provided, these matrices are filled with NAs.

If arguments W and/or H are provided, the method creates a NMF model where the basis and mixture coefficient matrices, W and H, are initialised using the values of W and/or H.

The dimensions given by target, W and H, must be compatible. However, whenever possible, the method will reduce the dimensions to the achieve dimension compatibility.

When W and H are both provided, the NMF object created is suitable to seed a NMF algorithm in a call to the nmf method. Note that in this case the factorisation rank is implicitly set by the number of basis vectors.

signature(rank = "numeric", target = "matrix") This call is equivalent to nmfModel(rank, dim(target), ...). That is that the returned NMF object is fits a target matrix of the same dimension as target.

`signature(rank = "missing", target = "ANY", ...)` This call is equivalent to `nmfModel(0, target, ...)`.

`signature(rank = "NULL", target = "ANY")` This call is equivalent to `nmfModel(0, target, ...)`.

`signature(rank = "numeric", target = "missing")` This call is equivalent to `nmfModel(rank, 0, ...)`.

Examples

```
# List all NMF models
nmfModel()
# or list only the built-in models
nmfModel(builtin.only=TRUE)

# create a NMF object based on one random matrix: the missing matrix is deduced
# Note this only works when using factory method NMF
n <- 50; r <- 3;
w <- matrix(runif(n*r), n, r)
nmfModel(W=w)

# create a NMF object based on random (compatible) matrices
p <- 20
h <- matrix(runif(r*p), r, p)
nmfModel(W=w, H=h)

# create a NMF object based on incompatible matrices: generate an error
h <- matrix(runif((r+1)*p), r+1, p)
## Not run: new('NMFstd', W=w, H=h)

# same thing using the factory method: dimensions are corrected and a warning
# is thrown saying that the dimensions used are reduced
nmfModel(W=w, H=h)

# apply default NMF algorithm to a random target matrix
V <- matrix(runif(n*p), n, p)
## Not run: nmf(V, r)
```

NMFns-class *Nonsmooth Nonnegative Matrix Factorization*

Description

Class that implements the *Nonsmooth Nonnegative Matrix Factorization* (nsNMF) model, required by the Nonsmooth NMF algorithm.

The Nonsmooth NMF algorithm is defined by Pascual-Montano et al. (2006) as a modification of the standard divergence based NMF algorithm (see section Details and references below). It aims at obtaining sparser factor matrices, by the introduction of a smoothing matrix.

Usage

```
## S4 method for signature 'NMFns':
fitted(object, W, H, S, ...)
## S4 method for signature 'NMFns':
smoothing(x, theta)
```

Arguments

x	an object of class NMFns
object	an object of class NMFns
W	the matrix of basis vectors, i.e. the first matrix factor in the non-smooth NMF model.
H	the matrix of mixture coefficients, i.e. the third matrix factor the non-smooth NMF model.
S	the smoothing matrix, i.e. the middle matrix factor in the non-smooth NMF model.
theta	a single numericto be used as smoothing parameter (see section Details).
...	extra parameters passed to method smoothing. So typically used to pass a value for theta.

Details

The Nonsmooth NMF algorithm is a modification of the standard divergence based NMF algorithm (see NMF). Given a non-negative $n \times p$ matrix V and a factorization rank r, it fits the following model:

$$V \equiv WS(\theta)H,$$

where:

- W and H are such as in the standard model, that is non-negative matrices of dimension $n \times r$ and $r \times p$ respectively;
- S is a $r \times r$ square matrix whose entries depends on an extra parameter $0 \leq \theta \leq 1$ in the following way:

  $$S = (1 - \theta)I + \frac{\theta}{r}11^T,$$

  where I is the identity matrix and 1 is a vector of ones.

The interpretation of S as a smoothing matrix can be explained as follows: Let X be a positive, nonzero, vector. Consider the transformed vector $Y = SX$. If $\theta = 0$, then $Y = X$ and no smoothing on X has occurred. However, as $\theta \rightarrow 1$, the vector Y tends to the constant vector with all elements almost equal to the average of the elements of X. This is the smoothest possible vector in the sense of non-sparseness because all entries are equal to the same nonzero value, instead of having some values close to zero and others clearly nonzero.

Algorithm

The Nonsmooth NMF algorithm uses a modified version of the multiplicative update equations in Lee & Seung's method for Kullbach-Leibler divergence minimization. The update equations are modified to take into account the – constant – smoothing matrix. The modification reduces to using matrix WS instead of matrix W in the update of matrix H, and similarly using matrix SH instead of matrix H in the update of matrix W.

After matrix W have been updated, each of its columns is scaled so that it sums up to 1.

Objects from the Class

Object of class `NMFns` can be created using the standard way with operator `new`

However, as for all the classes that extend class `NMFstd`, objects of class `NMFns` should be created using factory method `nmfModel` :

```
new('NMFns', theta=0.8)
nmfModel(model='NMFns')
nmfModel(model='NMFns', theta=0.8)
```

See `nmfModel` for more details on how to use the factory method.

Slots

Class `NMFns` extends `NMF` adding a single slot:

`theta`**:** Single `"numeric"` that contains the smoothing parameter. Default prototype value is `0.5`.

Extends

Class `"NMF"`, directly.

Methods

fitted `signature(object = "NMFns")`: returns the estimated target matrix according to the Nonsmooth-NMF model `object`:

$$\hat{V} = \hat{V}(\theta) = WS(\theta)H$$

Note that this method is part of the minimum interface for NMF model, as defined by class `NMF`.

smoothing returns the smoothing matrix $S(\theta)$. See section *Details*.

show `signature(object = "NMFns")`: standard generic `show` method for objects of class `NMFns`. It calls the parent class `show` method (i.e. for class `NMF`) and add the value of parameter `theta` to the display.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Alberto Pascual-Montano et al. (2006). Nonsmooth Nonnegative Matrix Factorization (nsNMF). *IEEE Transactions On Pattern Analysis And Machine Intelligence* , Vol. 28, No. 3, March 2006 403

See Also

NMF , nmf-methods

Examples

```
# create a completely empty NMF object
new('NMFns')

# create a NMF object based on random (compatible) matrices
n <- 50; r <- 3; p <- 20
w <- matrix(runif(n*r), n, r)
h <- matrix(runif(r*p), r, p)
nmfModel(model='NMFns', W=w, H=h)

# apply Nonsmooth NMF algorithm to a random target matrix
V <- matrix(runif(n*p), n, p)
## Not run: nmf(V, r, 'ns')
```

NMFOffset-class *Nonnegative Matrix Factorization with Offset*

Description

Class that implements the *Nonnegative Matrix Factorization with Offset* model, required by the NMF with Offset algorithm.

The NMF with Offset algorithm is defined by Badea (2008) as a modification of Lee & Seung's euclidean based NMF algorithm (see section Details and references below). It aims at obtaining 'cleaner' factor matrices, by the introduction of an offset matrix, explicitly modelling a feature specific baseline – constant across samples.

Objects from the Class

Object of class NMFOffset can be created using the standard way with operator new

However, as for all the classes that extend class NMFstd, objects of class NMFOffset should be created using factory method nmfModel :

```
new('NMFOffset')
nmfModel(model='NMFOffset')
nmfModel(model='NMFOffset', W=w, offset=rep(1, nrow(w)))
```

See nmfModel for more details on how to use the factory method.

Slots

Class NMFOffset extends NMF adding a single slot:

offset: A "numeric" vector to handle the common baseline for each feature. Its length will always be equal to the number of features, i.e. the number of rows in slot W.

Extends

Class "NMF", directly.

Methods

fitted signature(object = "NMFOffset"): returns the estimated target matrix according to the NMF with Offset model object:

$$\hat{V} = WH + offset$$

Note that this method is part of the minimum interface for NMF model, as defined by class NMF.

initialize initialize method for class NMFOffset. It ensures consistency between slots W and slot offset.

offset signature(object = "NMFOffset"): return the value of slot offset.

rnmf returns the object with slots W, H and offset filled with random values drawn from a uniform distribution. This method first calls method rnmf for NMF object to set the entries of slots W and H, then sets the entries in slot offset within the interval [0, max(max(W),max(H))].

show signature(object = "NMFOffset"): standard generic show method for objects of class NMFOffset. It calls the parent class show method (i.e. for class NMF) and add the value of vector offset to the display (only the first five elements are displayed).

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Badea (2008). Extracting Gene Expression Profiles Common To Colon And Pancreatic Adenocaricinoma Using Simultaneous Nonnegative Matrix Factorization. In *Pacific Symposium on Biocomputing* , **13**, 279-290

See Also

NMF, nmf-methods

Examples

```
# create a completely empty NMF object
new('NMFOffset')

# create a NMF object based on random (compatible) matrices
n <- 50; r <- 3; p <- 20
w <- matrix(runif(n*r), n, r)
h <- matrix(runif(r*p), r, p)
nmfModel(model='NMFOffset', W=w, H=h, offset=rep(0.5, nrow(w)))

# apply Nonsmooth NMF algorithm to a random target matrix
V <- matrix(runif(n*p), n, p)
## Not run: nmf(V, r, 'offset')
```

NMFSet-class *Deprecated Class to store results from multiple runs of NMF algorithms*

Description

This class is deprecated and replaced by class NMFfitX and its extensions. It remains only for backward compatibility and will be defunct in the next release.

It extends the base class list to store the result from a multiple run of NMF algorithms.

The elements are of class NMF.

Slots

consensus: Object of class "matrix" used to store the consensus matrix when multiple runs have been performed with option keep.all=FALSE. In this case, only the best factorization is returned, so the object is of length 1. However the consensus matrix across all runs is still computed and stored in this slot.

nrun: an integer that contains the number of runs when NMF is performed with option keep.all=FALSE. See nmf.

runtime: Object of class "proc_time" that contains various measures of the time spent to perform all the runs.

.Data: standard slot that contains the S3 list object data. See R documentation on S4 classes for more details.

Methods

All the methods for this class have been removed from the package and are substituted by methods for NMFfitX objects.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

See Also

NMFfitX, NMF , nmf-methods, nmf-multiple

NMFstd-class *Implement of the standard NMF model*

Description

Class that implements the standard model of Nonnegative Matrix Factorisation.

It provides a general structure and generic functions to manage factorizations that follow NMF standard model.

Details

Let V be a $n \times m$ non-negative matrix and r a positive integer. In its standard form (see references below), a NMF of V is commonly defined as a pair of matrices (W, H) such that:

$$V \equiv W H,$$

where:

- W and H are $n \times r$ and $r \times m$ matrices respectively with non-negative entries;
- $\equiv$ is to be understood with respect to some loss function. Common choices of loss functions are based on Frobenius norm or Kullbach-Leibler divergence.

Integer r is called the *factorization rank*. Depending on the context of application of NMF, the columns of W and H take different names:

columns of W basis vector, metagenes, factors, source, image basis

columns of H mixture coefficients, metagenes expression profiles, weights

NMF approach has been successfully applied to several fields. Package NMF was implemented trying to use names as generic as possible for objects and methods. The following terminology is used:

samples the columns of the target matrix V

features the rows of the target matrix V

basis matrix the first matrix factor W

basis vectors the columns of first matrix factor W

mixture matrix the second matrix factor H

mixtures coefficients the columns of second matrix factor H

However, because package NMF was primilary implemented to work with gene expression microarray data, it also provides a layer to easily and intuitively work with objects from the Bioconductor base framework. See NMF-bioc for more details.

Slots

`W`: A `"matrix"` that contains the *first* matrix factor of the factorisation

`H`: A `"matrix"` that contains the *second* matrix factor of the factorisation

Validity checks

The validity method for class `NMF` checks for compatibility of slots `W` and `H`, as those matrices must be compatible with respect to the matrix product. It also checks the relevance of the factorisation, and throws a warning when the factorisation rank is greater than the number of columns in `H`.

Objects from the Class

Factory method

The more convenient way of creating `NMF` objects is to use factory method `nmfModel`:

```
nmfModel(rank=0, target=0, model='NMFstd', ...)
```

It provides a unique interface to create `NMF` objects that can follow different NMF models, and is designed to resolve potential inconsistencies in the matrices dimensions. See `nmfModel`.

For example, to build a 5-rank NMF model compatible to fit a given matrix V, one calls:

```
nmfModel(5, V)
```

If the factors W and H are already available, they can be used to initialise the model:

```
nmfModel(5, V, W=w, H=h)
```

Standard way

Objects can still be created by calls of the usual form:

```
new("NMF")
new("NMF", W=w, H=h)
```

Methods

distance `signature(target = "matrix", x = "NMF")`: return the value of the loss function given a target matrix and a NMF fit.

fitted `signature(object = "NMF")`: compute the estimated target matrix according to the standard NMF model `object`, i.e. as the matrix product of slots `W` and `H`.

Note that this method is part of the minimum interface for NMF model, as defined by class `NMF`.

basis `signature(object = "NMF")`: Returns slot `W`, the matrix of basis vectors in NMF model `object`.

Note that this method is part of the minimum interface for NMF models, as defined by class `NMF`. See `NMF`.

basis<- `signature(object = "NMF", value = "matrix")`: Sets the value of slot `W`, the matrix of basis vectors in NMF model `object`.

Note that this method is part of the minimum interface for NMF models, as defined by class `NMF`. See `NMF`.

coef `signature(object = "NMF")`: Returns slot `H`, the matrix of mixture coefficients in the NMF model `object`.

Note that this method is part of the minimum interface for NMF models, as defined by class `NMF`. See `NMF`.

coef<- `signature(object = "NMF", value = "matrix")`: Set the value of slot `H`, the matrix of mixture coefficients in the NMF model `object`.

Note that this method is part of the minimum interface for NMF models, as defined by class `NMF`. See `NMF`.

rnmf `signature(x = "NMF", target)`: seed NMF model `x` with random values drawn from a random distribution. If a target is specified as a `matrix`, then the values are drawn within the interval `[0, max(target)]`.

show `signature(object = "NMF")`: standard generic `show` method for objects of class `NMF`.

summary `signature(x = "NMF")`: standard generic `summary` method for objects of class `NMF`.

Class `NMFstd` inherits from all the methods defined on class `NMF` to manipulate and interpret NMF models. Some useful are: `dim`, `nbasis`, `predict`, `sparseness`. See `NMF` for more details.

Author(s)

Renaud Gaujoux <renaud@cbio.uct.ac.za>

References

Definition of Nonnegative Matrix Factorization in its modern formulation:

Lee D.D. and Seung H.S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, **401**, 788–791.

Historical first definition and algorithms:

Paatero, P., Tapper, U. (1994). Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, **2**, 111–126 , doi:10.1002/env.3170050203.

Reference for some utility functions:

Kim, H. and Park, H. (2007). Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*.

Hoyer (2004). Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, **5**, 1457-1469.

See Also

Main interface to perform NMF in `nmf-methods`.

Method `seed` to set NMF objects with values suitable to start algorithms with.

Examples

```
# create a completely empty NMF object (i.e. 0 features, 0 basis components, 0 samples)
new('NMFstd')

# create a NMF object based on one random matrix: the missing matrix is deduced
# Note this only works when using factory method NMF
n <- 50; r <- 3;
w <- matrix(runif(n*r), n, r)
nmfModel(W=w)

# create a NMF object based on random (compatible) matrices
p <- 20
h <- matrix(runif(r*p), r, p)
nmfModel(W=w, H=h)

# create a NMF object based on incompatible matrices: generate an error
h <- matrix(runif((r+1)*p), r+1, p)
## Not run: new('NMFstd', W=w, H=h)

# same thing using the factory method: dimensions are corrected and a warning
# is thrown saying that the dimensions used are reduced
nmfModel(W=w, H=h)

# apply default NMF algorithm to a random target matrix
V <- matrix(runif(n*p), n, p)
## Not run: nmf(V, r)
```

Index