Package: RcppML (via r-universe)

October 21, 2024

Type Package

Title Rcpp Machine Learning Library

Version 0.5.6

Date 2022-15-12

Description Fast machine learning algorithms including matrix factorization and divisive clustering for large sparse and dense matrices.

License GPL-3 + file LICENSE

Imports Rcpp, Matrix, methods, stats, knitr, utils

LinkingTo Rcpp

VignetteBuilder knitr

RoxygenNote 7.2.1

Suggests ggplot2, rmarkdown, devtools, ggrepel, uwot, cowplot, viridis, testthat (>= 3.0.0)

Config/testthat/edition 3

LazyData true

URL https://github.com/zdebruine/RcppML

BugReports https://github.com/zdebruine/RcppML/issues

Encoding UTF-8

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

RemoteUrl https://github.com/zdebruine/rcppml

RemoteRef HEAD

RemoteSha 5449a5b479908f40f56cf911f11e0a7e156d207f

Contents

`align` *Align two NMF models*

Description

Align two NMF models

Usage

```
align(object, ...)

## S4 method for signature 'nmf'
align(object, ref, method = "cosine", ...)
```

Arguments

`object`	nmf model to be aligned to `ref`
`...`	arguments passed to or from other methods
`ref`	reference nmf model to which `object` will be aligned
`method`	either `cosine` or `cor`

Details

For nmf models, factors in object are reordered to minimize the cost of bipartite matching (see bipartiteMatch) on a cosine or correlation distance matrix. The w matrix is used for matching, and must be equidimensional in object and ref.

---

align_models *Align two matrices with bipartite matching*

Description

Same as the align S4 method for the ‘nmf‘ class, but operates only on the ‘w‘ matrices.

Usage

```
align_models(w, wref, method = "cosine", ...)
```

Arguments

w	matrix with columns to be aligned to columns in wref
wref	reference matrix to which columns in w will be aligned
method	distance metric (either cor or cosine) to use for constructing the cost matrix
...	additional arguments

---

aml *Acute Myelogenous Leukemia cells*

Description

DNA methylation in ~800 regions in 123 Acute Myelogenous Leukemia (AML) samples classified by probable cell of origin (GMP, L-MPP, or MEP), and 5 samples from healthy references for each suspected cell of origin (GMP, L-MPP, MEP).

Usage

```
data(aml)
```

Format

list of dense matrix of features (methylated regions) as rows and samples ("AML" or "Control" samples, classified by putative cell of origin or reference cell type) as columns. The "metadata_h" maps to publicly available clinical metadata.

Details

AML methylation signatures differ from their cell of origin by additive methylation and subtractive methylation, in addition to tumor-specific heterogeneity. These AML tumors likely originated from one of three healthy cell types (GMP, LMPP, or MEP), the challenge is to classify the cell of origin based on these healthy cell type DMRs.

`bipartiteMatch` *Bipartite graph matching*

Description

Hungarian algorithm for matching samples in a bipartite graph from a distance ("cost") matrix

Usage

```
bipartiteMatch(x)
```

Arguments

`x`	symmetric matrix giving the cost of every possible pairing

Details

This implementation was adapted from RcppHungarian, an Rcpp wrapper for the original C++ implementation by Cong Ma (2016).

Value

List of "cost" and "pairs"

`bipartition` *Bipartition a sample set*

Description

Spectral biparitioning by rank-2 matrix factorization

Usage

```
bipartition(data, tol = 1e-05, nonneg = TRUE, ...)
```

Arguments

data	dense or sparse matrix of features in rows and samples in columns. Prefer matrix or Matrix::dgCMatrix, respectively
tol	tolerance of the fit
nonneg	enforce non-negativity of the rank-2 factorization used for bipartitioning
...	development parameters

Details

Spectral bipartitioning is a popular subroutine in divisive clustering. The sign of the difference between sample loadings in factors of a rank-2 matrix factorization gives a bipartition that is nearly identical to an SVD.

Rank-2 matrix factorization by alternating least squares is faster than rank-2-truncated SVD (i.e. *irlba*).

This function is a specialization of rank-2 nmf with support for factorization of only a subset of samples, and with additional calculations on the factorization model relevant to bipartitioning. See nmf for details regarding rank-2 factorization.

Value

A list giving the bipartition and useful statistics:

- v : vector giving difference between sample loadings between factors in a rank-2 factorization
- dist : relative cosine distance of samples within a cluster to centroids of assigned vs. not-assigned cluster
- size1 : number of samples in first cluster (positive loadings in 'v')
- size2 : number of samples in second cluster (negative loadings in 'v')
- samples1: indices of samples in first cluster
- samples2: indices of samples in second cluster
- center1 : mean feature loadings across samples in first cluster
- center2 : mean feature loadings across samples in second cluster

Advanced Parameters

Several parameters may be specified in the ... argument:

- diag = TRUE: scale factors in w and h to sum to 1 by introducing a diagonal, d. This should generally never be set to FALSE. Diagonalization enables symmetry of models in factorization of symmetric matrices, convex L1 regularization, and consistent factor scalings.
- samples = 1:ncol(A): samples to include in bipartition, numbered from 1 to ncol(A). Default is all samples.
- calc_dist = TRUE: calculate the relative cosine distance of samples within a cluster to either cluster centroid. If TRUE, centers for clusters will also be calculated.
- seed = NULL: random seed for model initialization, generally not needed for rank-2 factorizations because robust solutions are recovered when diag = TRUE
- maxit = 100: maximum number of alternating updates of w and h. Generally, rank-2 factorizations converge quickly and this should not need to be adjusted.

Author(s)

Zach DeBruine

References

Kuang, D, Park, H. (2013). "Fast rank-2 nonnegative matrix factorization for hierarchical document clustering." Proc. 19th ACM SIGKDD intl. conf. on Knowledge discovery and data mining.

See Also

nmf, dclust

Examples

```
## Not run:
library(Matrix)
data(iris)
A <- as(as.matrix(iris[,1:4]), "dgCMatrix")
bipartition(A, calc_dist = TRUE)

## End(Not run)
```

---

biplot,nmf-method *Biplot for NMF factors*

Description

Produces a biplot from the output of nmf

Usage

```
## S4 method for signature 'nmf'
biplot(x, factors = c(1, 2), matrix = "w", group_by = NULL, ...)
```

Arguments

x	an object of class "nmf"
factors	length 2 vector specifying factors to plot.
matrix	either w or h
group_by	a discrete factor giving groupings for samples or features. Must be of the same length as number of samples in object$h or number of features in object$w.
...	for consistency with biplot generic

Value

ggplot2 object

See Also

nmf

`cosine` *Cosine similarity*

Description

Column-by-column Euclidean norm cosine similarity for a matrix, pair of matrices, pair of vectors, or pair of a vector and matrix. Supports sparse matrices.

Usage

```
cosine(x, y = NULL)
```

Arguments

x	matrix or vector of, or coercible to, class "dgCMatrix" or "sparseVector"
y	(optional) matrix or vector of, or coercible to, class "dgCMatrix" or "sparseVector"

Details

This function takes advantage of extremely fast vector operations and is able to handle very large datasets.

`cosine` applies a Euclidean norm to provide very similar results to Pearson correlation. Note that negative values may be returned due to the use of Euclidean normalization when all associations are largely random.

This function adopts the sparse matrix computational strategy applied by `qlcMatrix::cosSparse`, and extends it to any combination of single and/or pair of sparse matrix and/or dense vector.

Value

dense matrix, vector, or value giving cosine distances

`crossValidate` *Cross-validation for NMF*

Description

Find an "optimal" rank for a Non-Negative Matrix Factorization using cross-validation. Returns a `data.frame` with class `nmfCrossValidate`. Plot results using the `plot` class method.

Usage

```
crossValidate(data, k, reps = 3, n = 0.05, verbose = FALSE, ...)

## S3 method for class 'nmfCrossValidate'
plot(x, ...)
```

Arguments

`data`	dense or sparse matrix of features in rows and samples in columns. Prefer `matrix` or `Matrix::dgCMatrix`, respectively
`k`	array of factorization ranks to test
`reps`	number of independent replicates to run
`n`	fraction of values to handle as missing (default is 5%, or `0.05`)
`verbose`	should updates be displayed when each factorization is completed
`...`	parameters to `RcppML::nmf`, not including `data` or `k`
`x`	`nmfCrossValidate` object, the result of `crossValidate`

Details

A random speckled pattern of values is masked off during model fitting, and the mean squared error of prediction is evaluated after the model has reached the desired tolerance. The rank at which the model achieves the lowest error (best prediction accuracy) is the optimal rank.

Value

`data.frame` with class `nmfCrossValidate` with columns `rep`, `k`, and `value`

See Also

`nmf`

`dclust` *Divisive clustering*

Description

Recursive bipartitioning by rank-2 matrix factorization with an efficient modularity-approximate stopping criteria

Usage

```
dclust(
  A,
  min_samples,
  min_dist = 0,
  tol = 1e-05,
  maxit = 100,
  nonneg = TRUE,
  seed = NULL
)
```

Arguments

`A`	matrix of features-by-samples in sparse format (preferred class is "Matrix::dgCMatrix")
`min_samples`	stopping criteria giving the minimum number of samples permitted in a cluster
`min_dist`	stopping criteria giving the minimum cosine distance of samples within a cluster to the center of their assigned vs. unassigned cluster. If `0`, neither this distance nor cluster centroids will be calculated.
`tol`	in rank-2 NMF, the correlation distance $(1 - R^2)$ between w across consecutive iterations at which to stop factorization
`maxit`	maximum number of fitting iterations
`nonneg`	in rank-2 NMF, enforce non-negativity
`seed`	random seed for rank-2 NMF model initialization

Details

Divisive clustering is a sensitive and fast method for sample classification. Samples are recursively partitioned into two groups until a stopping criteria is satisfied and prevents successful partitioning.

See `nmf` and `bipartition` for technical considerations and optimizations relevant to bipartitioning.

Stopping criteria. Two stopping criteria are used to prevent indefinite division of clusters and tune the clustering resolution to a desirable range:

- `min_samples`: Minimum number of samples permitted in a cluster
- `min_dist`: Minimum cosine distance of samples to their cluster center relative to their unassigned cluster center (an approximation of Newman-Girvan modularity)

Newman-Girvan modularity (Q) is an interpretable and widely used measure of modularity for a bipartition. However, it requires the calculation of distance between all within-cluster and between-cluster sample pairs. This is computationally intensive, especially for large sample sets.

`dclust` uses a measure which linearly approximates Newman-Girvan modularity, and simply requires the calculation of distance between all samples in a cluster and both cluster centers (the assigned and unassigned center), which is orders of magnitude faster to compute. Cosine distance is used instead of Euclidean distance since it handles outliers and sparsity well.

A bipartition is rejected if either of the two clusters contains fewer than `min_samples` or if the mean relative cosine distance of the bipartition is less than `min_dist`.

A bipartition will only be attempted if there are more than `2 * min_samples` samples in the cluster, meaning that `dist` may not be calculated for some clusters.

Reproducibility. Because rank-2 NMF is approximate and requires random initialization, results may vary slightly across restarts. Therefore, specify a `seed` to guarantee absolute reproducibility.

Other than setting the seed, reproducibility may be improved by setting `tol` to a smaller number to increase the exactness of each bipartition.

Value

A list of lists corresponding to individual clusters:

- id : character sequence of "0" and "1" giving position of clusters along splitting hierarchy
- samples : indices of samples in the cluster
- center : mean feature expression of all samples in the cluster
- dist : if applicable, relative cosine distance of samples in cluster to assigned/unassigned cluster center.
- leaf : is cluster a leaf node

Author(s)

Zach DeBruine

References

Schwartz, G. et al. "TooManyCells identifies and visualizes relationships of single-cell clades". Nature Methods (2020).

Newman, MEJ. "Modularity and community structure in networks". PNAS (2006)

Kuang, D, Park, H. (2013). "Fast rank-2 nonnegative matrix factorization for hierarchical document clustering." Proc. 19th ACM SIGKDD intl. conf. on Knowledge discovery and data mining.

See Also

`bipartition`, `nmf`

Examples

```
## Not run:
data(USArrests)
A <- Matrix::as(as.matrix(t(USArrests)), "dgCMatrix")
clusters <- dclust(A, min_samples = 2, min_dist = 0.001)
str(clusters)

## End(Not run)
```

`evaluate` *Evaluate an NMF model*

Description

Calculate mean squared error for an NMF model, accounting for any masking schemes requested during fitting.

Usage

```
evaluate(x, ...)

## S4 method for signature 'nmf'
evaluate(x, data, mask = NULL, missing_only = FALSE, ...)
```

Arguments

`x`	fitted model, class `nmf`, generally the result of calling `nmf`, with models of equal dimensions as `data`
`...`	development parameters
`data`	dense or sparse matrix of features in rows and samples in columns. Prefer `matrix` or `Matrix::dgCMatrix`, respectively
`mask`	dense or sparse matrix of values in `data` to handle as missing. Prefer `Matrix::dgCMatrix`. Alternatively, specify `"zeros"` or `"NA"` to mask either all zeros or NA values.
`missing_only`	calculate mean squared error only for missing values specified as a matrix in `mask`

hawaiibirds *Bird species frequency in Hawaii*

Description

Frequency of bird species observation within 1km-squared grids in Hawaii as recorded in the eBird project. Not all counts are complete because some grids are sampled better than others (i.e. urban areas, birding hotspots).

Usage

```
data(hawaiibirds)
```

Format

list of three components: `counts` giving mean total counts of species in each grid, `metadata_h` giving information about each grid (i.e. latitude, longitude, and island), and `metadata_w` giving information about species taxonomic classification.

Details

This dataset was obtained as follows:

- All eBird observations in Hawaii were downloaded from the eBird website as of Oct. 2021
- Only complete checklists were retained
- Only non-X counts were retained
- Species were removed with fewer than 50 observations or that were spuhs
- Grids were defined by latitude/longitude rounded to two decimal places
- Grids were removed with fewer than 10 checklists
- Mean frequency of species in each grid was calculated based on the number of times a species was observed and the number of checklists submitted in that grid.
- Grids were assigned to one of each major Hawaiian island based on geographical coordinates.

lnmf *Linked non-negative matrix factorization*

Description

Run lNMF on a list of datasets to separate shared and unique signals.

Usage

```
lnmf(
  data,
  k_wh,
  k_uv,
  tol = 1e-04,
  maxit = 100,
  L1 = c(0, 0),
  L2 = c(0, 0),
  seed = NULL,
  mask = NULL
)
```

Arguments

data	list of dense or sparse matrices giving features in rows and samples in columns. Rows in all matrices must correspond exactly. Prefer `matrix` or `Matrix::dgCMatrix`, respectively
k_wh	rank of the shared factorization
k_uv	ranks of the unique factorizations, an array corresponding to each dataset provided
tol	tolerance of the fit
maxit	maximum number of fitting iterations
L1	LASSO penalties in the range (0, 1], single value or array of length two for `c(w & u, h & v)`
L2	Ridge penalties greater than zero, single value or array of length two for `c(w & u, h & v)`
seed	single initialization seed or array of seeds. If multiple seeds are provided, the model with least mean squared error is returned.
mask	list of dense or sparse matrices indicating values in `data` to handle as missing. Prefer `Matrix::ngCMatrix`. Alternatively, specify a string `"zeros"` or `"NA"` to mask either all zeros or NA values.

Details

Detailed documentation to come

Value

an object of class `lnmf`

Author(s)

Zach DeBruine

References

DeBruine, ZJ, Melcher, K, and Triche, TJ. (2021). "High-performance non-negative matrix factorization for large single-cell data." BioRXiv.

See Also

nmf, nnls

`movielens` *Movie ratings*

Description

~250,000 Ratings of ~3800 movies across 19 genres by 610 users on a scale of 1 to 5 stars. Zeros indicate no rating.

Usage

```
data(movielens)
```

Format

list of two components: `ratings` matrix in `Matrix::dgCMatrix` format of movies (rows) vs. users (columns), and `genres` as a sparse logical matrix in `Matrix::lgCMatrix` format, giving genres (rows) to which each movie (columns) is assigned.

Details

This dataset was derived from https://grouplens.org/datasets/movielens/ (ml-latest-small.zip), and movies without a genre assignment were removed. Half-star ratings were rounded up. Movies with fewer than 5 ratings were removed.

`mse` *Mean squared error of factor model*

Description

Same as the `evaluate` S4 method for the `nmf` class, but allows one to input the ‘w‘, ‘d‘, ‘h‘, and ‘data‘ independently.

Usage

```
mse(w, d = NULL, h, data, mask = NULL, missing_only = FALSE, ...)
```

Arguments

w	feature factor matrix (features as rows)
d	scaling diagonal vector (if applicable)
h	sample factor matrix (samples as columns)
data	dense or sparse matrix of features in rows and samples in columns. Prefer matrix or Matrix::dgCMatrix, respectively
mask	dense or sparse matrix of values in data to handle as missing. Prefer Matrix::dgCMatrix. Alternatively, specify "zeros" or "NA" to mask either all zeros or NA values.
missing_only	only calculate mean squared error at masked values
...	additional arguments

---

nmf *Non-negative matrix factorization*

Description

High-performance NMF of the form $A = wdh$ for large dense or sparse matrices, returns an object of class nmf.

Usage

```
nmf(
  data,
  k,
  tol = 1e-04,
  maxit = 100,
  L1 = c(0, 0),
  L2 = c(0, 0),
  seed = NULL,
  mask = NULL,
  ...
)
```

Arguments

data	dense or sparse matrix of features in rows and samples in columns. Prefer matrix or Matrix::dgCMatrix, respectively
k	rank
tol	tolerance of the fit
maxit	maximum number of fitting iterations
L1	LASSO penalties in the range (0, 1], single value or array of length two for c(w, h)
L2	Ridge penalties greater than zero, single value or array of length two for c(w, h)

seed	single initialization seed or array, or a matrix or list of matrices giving initial w. For multiple initializations, the model with least mean squared error is returned.
mask	dense or sparse matrix of values in data to handle as missing. Prefer Matrix::dgCMatrix. Alternatively, specify "zeros" or "NA" to mask either all zeros or NA values.
...	development parameters

Details

This fast NMF implementation decomposes a matrix A into lower-rank non-negative matrices w and h, with columns of w and rows of h scaled to sum to 1 via multiplication by a diagonal, d:

$$A = wdh$$

The scaling diagonal ensures convex L1 regularization, consistent factor scalings regardless of random initialization, and model symmetry in factorizations of symmetric matrices.

The factorization model is randomly initialized. w and h are updated by alternating least squares.

RcppML achieves high performance using the Eigen C++ linear algebra library, OpenMP parallelization, a dedicated Rcpp sparse matrix class, and fast sequential coordinate descent non-negative least squares initialized by Cholesky least squares solutions.

Sparse optimization is automatically applied if the input matrix A is a sparse matrix (i.e. Matrix::dgCMatrix). There are also specialized back-ends for symmetric, rank-1, and rank-2 factorizations.

L1 penalization can be used for increasing the sparsity of factors and assisting interpretability. Penalty values should range from 0 to 1, where 1 gives complete sparsity.

Set options(RcppML.verbose = TRUE) to print model tolerances to the console after each iteration.

Parallelization is applied with OpenMP using the number of threads in getOption("RcppML.threads") and set by option(RcppML.threads = 0), for example. 0 corresponds to all threads, let OpenMP decide.

Value

object of class nmf

Slots

w feature factor matrix

d scaling diagonal vector

h sample factor matrix

misc list often containing components:

- tol : tolerance of fit
- iter : number of fitting updates
- runtime : runtime in seconds
- mse : mean squared error of model (calculated for multiple starts only)
- w_init : initial w matrix used for model fitting

Methods

S4 methods available for the `nmf` class:

- `predict`: project an NMF model (or partial model) onto new samples
- `evaluate`: calculate mean squared error loss of an NMF model
- `summary`: `data.frame` giving `fractional`, `total`, or `mean` representation of factors in samples or features grouped by some criteria
- `align`: find an ordering of factors in one `nmf` model that best matches those in another `nmf` model
- `prod`: compute the dense approximation of input data
- `sparsity`: compute the sparsity of each factor in w and h
- `subset`: subset, reorder, select, or extract factors (same as `[`)
- generics such as `dim`, `dimnames`, `t`, `show`, `head`

Author(s)

Zach DeBruine

References

DeBruine, ZJ, Melcher, K, and Triche, TJ. (2021). "High-performance non-negative matrix factorization for large single-cell data." BioRXiv.

Lin, X, and Boutros, PC (2020). "Optimization and expansion of non-negative matrix factorization." BMC Bioinformatics.

Lee, D, and Seung, HS (1999). "Learning the parts of objects by non-negative matrix factorization." Nature.

Franc, VC, Hlavac, VC, Navara, M. (2005). "Sequential Coordinate-Wise Algorithm for the Non-negative Least Squares Problem". Proc. Int'l Conf. Computer Analysis of Images and Patterns. Lecture Notes in Computer Science.

See Also

`project`, `mse`, `nnls`

Examples

```
## Not run:
# basic NMF
model <- nmf(rsparsematrix(1000, 100, 0.1), k = 10)

# compare rank-2 NMF to second left vector in an SVD
data(iris)
A <- Matrix::as(as.matrix(iris[, 1:4]), "dgCMatrix")
nmf_model <- nmf(A, 2, tol = 1e-5)
bipartitioning_vector <- apply(nmf_model$w, 1, diff)
second_left_svd_vector <- base::svd(A, 2)$u[, 2]
abs(cor(bipartitioning_vector, second_left_svd_vector))
```

```
# compare rank-1 NMF with first singular vector in an SVD
abs(cor(nmf(A, 1)$w[, 1], base::svd(A, 2)$u[, 1]))

# symmetric NMF
A <- crossprod(rsparsematrix(100, 100, 0.02))
model <- nmf(A, 10, tol = 1e-5, maxit = 1000)
plot(model$w, t(model$h))
# see package vignette for more examples

## End(Not run)
```

`nnls` *Non-negative least squares*

Description

Solves the equation a `%*%` x = b for x subject to $x > 0$.

Usage

```
nnls(a, b, cd_maxit = 100L, cd_tol = 1e-08, L1 = 0, L2 = 0, upper_bound = 0)
```

Arguments

a	symmetric positive definite matrix giving coefficients of the linear system
b	matrix giving the right-hand side(s) of the linear system
cd_maxit	maximum number of coordinate descent iterations
cd_tol	stopping criteria, difference in x across consecutive solutions over the sum of x
L1	L1/LASSO penalty to be subtracted from b
L2	Ridge penalty by which to shrink the diagonal of a
upper_bound	maximum value permitted in solution, set to `0` to impose no upper bound

Details

This is a very fast implementation of sequential coordinate descent non-negative least squares (NNLS), suitable for very small or very large systems. The algorithm begins with a zero-filled initialization of x.

Least squares by **sequential coordinate descent** is used to ensure the solution returned is exact. This algorithm was introduced by Franc et al. (2005), and our implementation is a vectorized and optimized rendition of that found in the NNLM R package by Xihui Lin (2020).

Value

vector or matrix giving solution for x

Author(s)

Zach DeBruine

References

DeBruine, ZJ, Melcher, K, and Triche, TJ. (2021). "High-performance non-negative matrix factorization for large single-cell data." BioRXiv.

Franc, VC, Hlavac, VC, and Navara, M. (2005). "Sequential Coordinate-Wise Algorithm for the Non-negative Least Squares Problem. Proc. Int'l Conf. Computer Analysis of Images and Patterns."

Lin, X, and Boutros, PC (2020). "Optimization and expansion of non-negative matrix factorization." BMC Bioinformatics.

Myre, JM, Frahm, E, Lilja DJ, and Saar, MO. (2017) "TNT-NN: A Fast Active Set Method for Solving Large Non-Negative Least Squares Problems". Proc. Computer Science.

See Also

nmf, project

Examples

```
## Not run:
# compare solution to base::solve for a random system
X <- matrix(runif(100), 10, 10)
a <- crossprod(X)
b <- crossprod(X, runif(10))
unconstrained_soln <- solve(a, b)
nonneg_soln <- nnls(a, b)
unconstrained_err <- mean((a %*% unconstrained_soln - b)^2)
nonnegative_err <- mean((a %*% nonneg_soln - b)^2)
unconstrained_err
nonnegative_err
all.equal(solve(a, b), nnls(a, b))

# example adapted from multiway::fnnls example 1
X <- matrix(1:100,50,2)
y <- matrix(101:150,50,1)
beta <- solve(crossprod(X)) %*% crossprod(X, y)
beta
beta <- nnls(crossprod(X), crossprod(X, y))
beta

# learn nmf model and do bvls projection
data(hawaiibirds)
w <- nmf(hawaiibirds$counts, 10)@w
h <- project(w, hawaiibirds$counts)
# now impose upper bound on solutions
h2 <- project(w, hawaiibirds$counts, upper_bound = 2)

## End(Not run)
```

project *Project a model onto new data*

Description

Equivalent to `predict` method for NMF, but requires only the `w` matrix to be supplied and not the entire NMF model. Use NNLS to project a basis factor model onto new samples.

Usage

```
project(w, data, L1 = 0, L2 = 0, mask = NULL, upper_bound = 0, ...)
```

Arguments

`w`	matrix of features (rows) by factors (columns), corresponding to rows in `data`
`data`	a dense or sparse matrix
`L1`	L1/LASSO penalty
`L2`	L2/Ridge penalty
`mask`	masking on data values
`upper_bound`	maximum value permitted in least squares solutions, essentially a bounded-variable least squares problem between 0 and `upper_bound`
`...`	arguments passed to `predict.nmf`

Details

See `nmf` for more info, as well as the `predict` method for NMF.

RcppML *RcppML: Rcpp Machine Learning Library*

Description

High-performance non-negative matrix factorization and linear model projection for sparse matrices, and fast non-negative least squares implementations

Author(s)

Zach DeBruine

r_matrix *Random transpose-identical dense/sparse matrix*

Description

Generate a random sparse matrix, just like `Matrix::rsparsematrix` or `(matrix(runif(nrow * ncol), nrow,))`, but much faster. Generation of transpose-identical matrices is also supported without additional computational cost.

Usage

```
r_matrix(nrow, ncol, transpose_identical = FALSE)

r_sparsematrix(
  nrow,
  ncol,
  inv_density,
  transpose_identical = FALSE,
  pattern = FALSE
)
```

Arguments

`nrow`	number of rows
`ncol`	number of columns
`transpose_identical`	should the matrix be transpose-identical?
`inv_density`	an integer giving the inverse density of the matrix (i.e. 10 percent density corresponds to `inv_density = 10`). Density is probabilistic, not exact. See examples.
`pattern`	should a pattern matrix (`Matrix::ngCMatrix`) be returned? If not, a `Matrix::dgCMatrix` with random uniform values will be returned.

See Also

`r_unif`

`r_matrix`, `r_unif`, `r_binom`

Examples

```
# generate a simple random matrix
A <- r_matrix(10, 10)

# generate two matrices that are transpose identical
set.seed(123)
A1 <- r_matrix(10, 100, transpose_identical = TRUE)
set.seed(123)
A2 <- r_matrix(100, 10, transpose_identical = TRUE)
```

```
all.equal(t(A2), A1)

# generate a transpose-identical pair of speckled matrices
set.seed(123)
A <- r_sparsematrix(10, 100, inv_density = 10, transpose_identical = TRUE)
set.seed(123)
A <- r_sparsematrix(100, 10, inv_density = 10, transpose_identical = TRUE)
all.equal(t(A), A)
Matrix::isSymmetric(A[1:10, 1:10])
heatmap(as.matrix(A), scale = "none", Rowv = NA, Colv = NA)

# note that density is probabilistic, not absolute
A <- replicate(1000, r_sparsematrix(100, 100, 10))
densities <- sapply(A, function(x) length(x@i) / prod(dim(x)))
plot(density(densities)) # normal distribution centered at 0.100
range(densities)
```

`r_sample` *Random distributions and samples*

Description

`r_sample` is just like `base::sample`, only faster. `r_sample` takes a sample of the specified size from the elements of `x` using replacement if indicated.

These functions generate random distributions (uniform, normal, or binomial) just like their base R counterparts (`runif`, `rnorm`, and `rbinom`), but faster.

Usage

```
r_sample(x, size = NULL, replace = FALSE)

r_unif(n, min = 0, max = 1)

r_binom(n, size = 1, inv_prob = 2)
```

Arguments

`x`	either a positive integer giving the number of items to choose from, or a vector of elements to shuffle or from which to choose. See 'Details'.
`size`	number of trials (one or more)
`replace`	should sampling be with replacement?
`n`	number of observations
`min`	finite lower limit of the uniform distribution
`max`	finite upper limit of the uniform distribution
`inv_prob`	inverse probability of success for each trial, must be integral (e.g. 50 percent success = 2, 10 percent success = 10)

Details

All RNGs make use of Marsaglia's xorshift method to generate random integers.

`r_unif` takes the random integer and divides it by the seed and returns the floating decimal portion of the result.

See Also

`r_matrix`, `r_sparsematrix`

Examples

```
# draw all integers from 1 to 10 in a random order
sample(10)

# shuffle a vector of values
v <- r_unif(3)
v
v_ <- sample(v)
v_

# draw values from a vector
sample(r_unif(100), 3)

# draw some integers between 1 and 1000
sample(1000, 3)

# simulate a uniform distribution
v <- r_unif(10000)
plot(density(v))

# simulate a binomial distribution
v <- r_binom(10000, 100, inv_prob = 10)
hist(v)
sum(v) / length(v)
# ~10 because 100 trials at 10 percent success odds
#   is about 10 successes per element

# get successful trials in a bernoulli distribution
v <- r_binom(100, 1, 20)
successful_trials <- slot(as(v, "nsparseVector"), "i")
successful_trials
```

`simulateNMF` *Simulate an NMF dataset*

Description

Generate a random matrix that follows some defined NMF model to test NMF factorizations. Adapts methods from `NMF::syntheticNMF`.

Usage

```
simulateNMF(nrow, ncol, k, noise = 0.5, dropout = 0.5, seed = NULL)
```

Arguments

nrow	number of rows
ncol	number of columns
k	true rank of simulated model
noise	standard deviation of Gaussian noise centered at 0 to add to input matrix. Any negative values after noise addition are set to 0.
dropout	density of dropout events
seed	seed for random number generation

Value

list of dense matrix A and true w and h models

---

sparsity *Compute the sparsity of each NMF factor*

Description

Compute the sparsity of each NMF factor

Usage

```
sparsity(object, ...)

## S4 method for signature 'nmf'
sparsity(object, ...)
```

Arguments

object	object of class nmf.
...	additional parameters

Details

For nmf models, the sparsity of each factor is computed and summarized or w and h matrices. A long data.frame with columns factor, sparsity, and model is returned.

subset,nmf-method *nmf class methods*

Description

Given an NMF model in the form $A = wdh$, `project` projects `w` onto `A` to solve for `h`.

Usage

```
## S4 method for signature 'nmf'
subset(x, i, ...)

## S4 method for signature 'nmf,ANY,ANY,ANY'
x[i]

## S4 method for signature 'nmf'
head(x, n = getOption("digits"), ...)

## S4 method for signature 'nmf'
show(object)

## S4 method for signature 'nmf'
dimnames(x)

## S4 method for signature 'nmf'
dim(x)

## S4 method for signature 'nmf'
t(x)

## S4 method for signature 'nmf'
sort(x, decreasing = TRUE, ...)

## S4 method for signature 'nmf'
prod(x, ..., na.rm = FALSE)

## S4 method for signature 'nmf'
x$name

## S4 method for signature 'nmf,list'
coerce(from, to)

## S4 method for signature 'nmf'
x[[i]]

## S4 method for signature 'nmf'
predict(object, data, L1 = 0, L2 = 0, mask = NULL, upper_bound = NULL, ...)
```

Arguments

x	object of class nmf.
i	indices
...	arguments passed to or from other methods
n	number of rows/columns to show
object	fitted model, class nmf, generally the result of calling nmf, with models of equal dimensions as data
decreasing	logical. Should the sort be increasing or decreasing?
na.rm	remove na values
name	name of nmf class slot
from	class which the coerce method should perform coercion from
to	class which the coerce method should perform coercion to
data	dense or sparse matrix of features in rows and samples in columns. Prefer matrix or Matrix::dgCMatrix, respectively
L1	a single LASSO penalty in the range (0, 1]
L2	a single Ridge penalty greater than zero
mask	dense or sparse matrix of values in data to handle as missing. Prefer Matrix::dgCMatrix. Alternatively, specify "zeros" or "NA" to mask either all zeros or NA values.
upper_bound	maximum value permitted in least squares solutions, essentially a bounded-variable least squares problem between 0 and upper_bound

Details

For the alternating least squares matrix factorization update problem $A = wh$, the updates (or projection) of h is given by the equation:

$$w^T wh = wA_j$$

which is in the form $ax = b$ where $a = w^T w$ $x = h$ and $b = wA_j$ for all columns j in A.

Any L1 penalty is subtracted from b and should generally be scaled to max(b), where $b = WA_j$ for all columns j in A. An easy way to properly scale an L1 penalty is to normalize all columns in w to sum to the same value (e.g. 1). No scaling is applied in this function. Such scaling guarantees that L1 = 1 gives a completely sparse solution.

There are specializations for dense and sparse input matrices, symmetric input matrices, and for rank-1 and rank-2 projections. See documentation for nmf for theoretical details and guidance.

Value

object of class nmf

Author(s)

Zach DeBruine

References

DeBruine, ZJ, Melcher, K, and Triche, TJ. (2021). "High-performance non-negative matrix factorization for large single-cell data." BioRXiv.

See Also

nnls, nmf

Examples

```
## Not run:
w <- matrix(runif(1000 * 10), 1000, 10)
h_true <- matrix(runif(10 * 100), 10, 100)
# A is the crossproduct of "w" and "h" with 10% signal dropout
A <- (w %*% h_true) * (r_sparsematrix(1000, 100, 10) > 0)
h <- project(w, A)
cor(as.vector(h_true), as.vector(h))

# alternating projections refine solution (like NMF)
mse_bad <- mse(w, rep(1, ncol(w)), h, A) # mse before alternating updates
h <- project(w, A)
w <- t(project(h, t(A)))
h <- project(w, A)
w <- t(project(h, t(A)))
h <- project(w, A)
w <- t(project(h, t(A)))
mse_better <- mse(w, rep(1, ncol(w)), h, A) # mse after alternating updates
mse_better < mse_bad

## End(Not run)
```

---

summary,nmf-method *Summarize NMF factors*

Description

summary method for class "nmf". Describes metadata representation in NMF factors. Returns object of class nmfSummary. Plot results using plot.

Usage

```
## S4 method for signature 'nmf'
summary(object, group_by, stat = "sum", ...)

## S3 method for class 'nmfSummary'
plot(x, ...)
```

Arguments

object	an object of class "nmf", usually, a result of a call to nmf
group_by	a discrete factor giving groupings for samples or features. Must be of the same length as number of samples in object$h or number of features in object$w.
stat	either sum (sum of factor weights falling within each group), or mean (mean factor weight falling within each group).
...	arguments passed to or from other methods
x	nmfSummary object, the result of calling summary on an nmf object

Value

data.frame with columns group, factor, and stat

Index