Package 'poismf'

September 12, 2020

Type Package

Title Factorization of Sparse Counts Matrices Through Poisson Likelihood

Version 0.2.5

Date 2020-09-12

Author David Cortes [aut, cre, cph], Jean-Sebastien Roy [cph], Stephen Nash [cph]

Maintainer David Cortes <david.cortes.rivera@gmail.com>

URL https://github.com/david-cortes/poismf

BugReports https://github.com/david-cortes/poismf/issues

Description Creates a low-rank factorization of a sparse counts matrix by maximizing Poisson likelihood with l1/l2 regularization
with all non-negative latent factors (e.g. for recommender systems or topic modeling) (Cortes, (2018) <arXiv:1811.01908>).
Similar to hierarchical Poisson factorization, but follows an optimization-based approach with regularization instead of a
hierarchical structure, and is fit through gradient-based methods instead of variational inference.

License BSD_2_clause + file LICENSE

Imports Matrix, methods

Enhances SparseM

RoxygenNote 7.1.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-09-12 11:40:02 UTC

R topics documented:

factors 2
factors.single 3
get.all.predictions 4
get.factor.matrices 5

get.model.mappings 6
poismf 6
poismf_unsafe 11
poisson.llk 12
predict.poismf 13
print.poismf 14
summary.poismf 15
topN 15
topN.new 16

Index **19**

`factors` *Determine latent factors for new rows/users*

Description

Determines the latent factors for new users (rows) given their counts for existing items (columns).

This function will use the same method and hyperparameters with which the model was fit. If using this for recommender systems, it's recommended to use instead the function factors.single, even though the new factors obtained with that function might not end up being in the same scale as the original factors in 'model$A'.

Note that, when using proximal gradient method (the default), results from this function and from 'get.factor.matrices' on the same data might differ a lot. If this is a problem, it's recommended to use conjugate gradient instead.

Usage

```
factors(model, X, add_names = TRUE)
```

Arguments

`model`	A Poisson factorization model as returned by 'poismf'.
`X`	New data for whose rows to determine latent factors. Can be passed as a 'data.frame' or as a sparse or dense matrix (see documentation of poismf for details on the data type). While other functions only accept sparse matrices in COO (triplets) format, this function will also take CSR matrices from the 'SparseM' package (recommended) and CSC matrices from the 'Matrix' package ('Matrix::dgCMatrix', but it's not recommended). Inputs will be converted to CSR regardless of their original format. If passing a 'data.frame', the first column should contain row indices or IDs, and these will be internally remapped - the mapping will be available as the row names for the matrix if passing 'add_names=TRUE', or as part of the outputs if passing 'add_names=FALSE'. The IDs passed in the first column will not be matched to the existing IDs of 'X' passed to 'poismf'. If 'X' passed to 'poismf' was a 'data.frame', 'X' here must also be passed as 'data.frame'. If 'X' passed to 'poismf' was a matrix and 'X' is a 'data.frame',

the second column of ‘X‘ here should contain column numbers (with numeration starting at 1).

`add_names` Whether to add row names to the output matrix if the indices were internally remapped - they will only be so if the ‘X‘ here is a ‘data.frame‘. Note that if the indices in passed in ‘X‘ here (first and second columns) are integers, once row names are added, subsetting ‘X‘ by an integer will give the row at that position - that is, if you want to obtain the corresponding row for ID=2 from ‘X‘ in ‘A_out‘, you need to use ‘A_out["2",]‘, not ‘A_out[2,]‘.

Details

The factors are initialized to the mean of each column in the fitted model.

Value

- If ‘X‘ was passed as a matrix, will output a matrix of dimensions (n, k) with the obtained factors. If passing ‘add_names=TRUE‘ and ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, this matrix will have row names. **Careful with subsetting with integers** (see documentation for ‘add_names‘).
- If ‘X‘ was passed as a ‘data.frame‘ and passing ‘add_names=FALSE‘ here, will output a list with an entry ‘factors‘ containing the latent factors as described above, and an entry ‘mapping‘ indicating to which row ID does each row of the output correspond.

See Also

factors.single topN.new

`factors.single` *Get latent factors for a new user given her item counts*

Description

This is similar to obtaining topics for a document in LDA. See also function factors for getting factors for multiple users/rows at a time.

This function works with one user at a time, and will use the truncated Newton-CG approach regardless of how the model was fit. Note that, since this optimization method will likely have different optimal hyperparameters than the other methods, it offers the option of varying those hyperparameters in here.

Usage

```
factors.single(
  model,
  X,
  l2_reg = model$l2_reg,
  l1_reg = model$l1_reg,
  weight_mult = model$weight_mult,
```

```
  maxupd = model$maxupd
)
```

Arguments

model	Poisson factorization model as returned by ‘poismf‘.
X	Data with the non-zero item indices and counts for this new user. Can be passed as a sparse vector from package ‘Matrix‘ (‘Matrix::dsparseVector‘, which can be created from indices and values through function ‘Matrix::sparseVector‘), or as a ‘data.frame‘, in which case will take the first column as the item/column indices (numeration starting at 1) and the second column as the counts. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, ‘X‘ here must also be a ‘data.frame‘.
l2_reg	Strength of L2 regularization to use for optimizing the new factors.
l1_reg	Strength of the L1 regularization. Not recommended.
weight_mult	Weight multiplier for the positive entries over the missing entries.
maxupd	Maximum number of Newton-CG updates to perform. You might want to increase this value depending on the use-case.

Details

The factors are initialized to the mean of each column in the fitted model.

Value

Vector of dimensionality ‘model$k‘ with the latent factors for the user, given the input data.

See Also

factors topN.new

---

get.all.predictions *Predict whole input matrix*

Description

Outputs the predictions for the whole input matrix to which the model was fit. Note that this will be a dense matrix, and in typical recommender systems scenarios will likely not fit in memory.

Usage

```
get.all.predictions(model)
```

Arguments

model	A Poisson factorization model as output by function ‘poismf‘.

Value

A matrix [dimA x dimB] with the full predictions for all rows and columns. If the ‘X‘ data passed to ‘poismf‘ was a ‘data.frame‘, the resulting output will have row and column names added to it. Be aware that, if ‘X‘ was a ‘data.frame‘ with integers as IDs, selecting columns or rows of the named matrix will output ordinal positions - e.g. ‘out[2,]‘ (second row) vs. ‘out["2",]‘ (row for ID=2 in ‘X‘). The mappings, if produced, can be obtained through function get.model.mappings.

`get.factor.matrices` *Extract Latent Factor Matrices*

Description

Extract the latent factor matrices for users (rows) and columns (items) from a Poisson factorization model object, as returned by function ‘poismf‘.

Usage

```
get.factor.matrices(model, add_names = TRUE)
```

Arguments

`model`	A Poisson factorization model, as produced by ‘poismf‘.
`add_names`	Whether to add row names to the matrices if the indices were internally remapped - they will only be so if the ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘. Note that if passing ‘X‘ as ‘data.frame‘ with integer indices to ‘poismf‘, once row names are added, subsetting such matrix by an integer will give the row at that position - that is, if you want to obtain the corresponding row for ID=2 from ‘X‘ in ‘factors$A‘, you need to use ‘factors$A["2",]‘, not ‘factors$A[2,]‘.

Details

If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, the mapping between IDs from ‘X‘ to row numbers in ‘A‘ and column numbers in ‘B‘ are avaiable under ‘model$levels_A‘ and ‘model$levels_B‘, respectively. They can also be obtained through ‘get.model.mappings‘, and will be added as row names if using ‘add_names=TRUE‘. **Be careful about subsetting with integers** (see documentation for ‘add_names‘ for details).

Value

List with entries ‘A‘ (the user factors) and ‘B‘ (the item factors).

See Also

get.model.mappings

`get.model.mappings` *Extract user/row and item/column mappings from Poisson model.*

Description

Will extract the mapping between IDs passed as ‘X‘ to function ‘poismf‘ and row/column positions in the latent factor matrices and prediction functions.

Such a mapping will only be generated if the ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, otherwise they will not be re-mapped.

Usage

```
get.model.mappings(model)
```

Arguments

`model`	A Poisson factorization model as returned by ‘poismf‘.

Value

A list with row entries:

- ‘rows‘: a vector in which each user/row ID is placed at its ordinal position in the internal data structures. If there is no mapping (e.g. if ‘X‘ passed to ‘poismf‘ was a sparse matrix), will be ‘NULL‘.
- ‘columns‘: a vector in which each item/column ID is placed at its ordinal position in the internal data structures. If there is no mapping (e.g. if ‘X‘ passed to ‘poismf‘ was a sparse matrix), will be ‘NULL‘.

See Also

get.factor.matrices

`poismf` *Factorization of Sparse Counts Matrices through Poisson Likelihood*

Description

Creates a low-rank non-negative factorization of a sparse counts matrix by maximizing Poisson likelihood minus L1/L2 regularization, using gradient-based optimization procedures.

Ideal for usage in recommender systems, in which the ‘X‘ matrix would consist of interactions (e.g. clicks, views, plays), with users representing the rows and items representing the columns.

Usage

```
poismf(
  X,
  k = 50,
  method = "tncg",
  l2_reg = "auto",
  l1_reg = 0,
  niter = "auto",
  maxupd = "auto",
  limit_step = TRUE,
  initial_step = 1e-07,
  weight_mult = 1,
  init_type = "gamma",
  seed = 1,
  nthreads = parallel::detectCores()
)
```

Arguments

X	The counts matrix to factorize. Can be: • A 'data.frame' with 3 columns, containing in this order: row index or user ID, column index or item ID, count value. The first two columns will be converted to factors to enumerate them internally, and will return those same values from 'topN'. In order to avoid this internal re-enumeration, can pass 'X' as a sparse COO matrix instead. • A sparse matrix from package 'Matrix' in triplets (COO) format (that is: 'Matrix::dgTMatrix') (recommended). Such a matrix can be created from row/column indices through function 'Matrix::sparseMatrix' (with 'giveCsparse=FALSE'). Will also accept them in CSC format ('Matrix::dgCMatrix'), but will be converted along the way (so it will be slightly slower). • A sparse matrix in COO format from the 'SparseM' package. Such a matrix can be created from row/column indices through 'new("matrix.coo", ra=values, ja=col_ix, ia=row_ix, dim=as.integer(c(m,n)))'. Will also accept them in CSR and CSC format, but will be converted along the way (so it will be slightly slower). • A full matrix (of class 'base::matrix') - this is not recommended though. Passing sparse matrices is faster as it will not need to re-enumerate the rows and columns, Full matrices will be converted to sparse.
k	Number of latent factors to use (dimensionality of the low-rank factorization). If 'k' is small (e.g. 'k=5'), it's recommended to use 'method='pg''. For large values, (e.g. 'k=100'), it's recommended to use 'method='tncg''. For medium values (e.g. 'k=50'), it's recommende to use either 'method='tncg'' or 'method='cg''.
method	Optimization method to use. Options are: • '"tncg"' : will use a truncated Newton-CG method. This is the slowest option, but tends to find better local optima, and if run for many iterations, tends to produce sparse latent factor matrices.

- ‘"cg"‘ : will use a Conjugate Gradient method, which is faster than the truncated Newton-CG, but tends not to reach the best local optima. Usually, with this method and the default hyperparameters, the latent factor matrices will not be very sparse.
- ‘"pg"‘ : will use a proximal gradient method, which is a lot faster than the other two and more memory-efficient, but tends to only work with very large regularization values, and doesn't find as good local optima, nor tends to result in sparse factors.

`l2_reg` Strength of L2 regularization. It is recommended to use small values along with ‘method='tncg'‘, very large values along with ‘method='pg'‘, and medium to large values with ‘method='cg'‘. If passing ‘"auto"‘, will set it to ‘10^3‘ for TNCG, ‘10^5‘ for CG, and ‘10^9‘ for PG.

`l1_reg` Strength of L1 regularization. Not recommended.

`niter` Number of alternating iterations to perform. One iteration denotes an update over both matrices. If passing ‘'auto'‘, will set it to 50 for TNCG, 25 for CG, and 10 for PG.

`maxupd` Maximum number of updates to each user/item vector within an iteration. Note: for 'method=TNCG', this means maximum number of **function evaluations** rather than number of updates, so it should be higher. You might also want to try decreasing this while increasing ‘niter‘. For ‘method='pg'‘, this will be taken as the actual number of updates, as it does not perform a line search like the other methods. If passing ‘"auto"‘, will set it to ‘5*k‘ for ‘method='tncg'‘, 25 for ‘method='cg'‘, and 10 for ‘method='pg'‘. If using ‘method='cg'‘, you might also try instead setting ‘maxupd=1‘ and ‘niter=100‘.

`limit_step` When passing ‘method='cg'‘, whether to limit the step sizes in each update so as to drive at most one variable to zero each time, as prescribed in [2]. If running the procedure for many iterations, it's recommended to set this to 'True'. You also might set ‘method='cg'‘ plus ‘maxupd=1‘ and ‘limit_step=FALSE‘ to reduce the algorithm to simple gradient descent with a line search.

`initial_step` Initial step size to use for proximal gradient updates. Larger step sizes reach converge faster, but are more likely to result in failed optimization. Ignored when passing ‘method='tncg'‘ or ‘method='cg'‘, as those will perform a line seach instead.

`weight_mult` Extra multiplier for the weight of the positive entries over the missing entries in the matrix to factorize. Be aware that Poisson likelihood will implicitly put more weight on the non-missing entries already. Passing larger values will make the factors have larger values (which might be desirable), and can help with instability and failed optimization cases. If passing this, it's recommended to try very large values (e.g. 10^2), and might require adjusting the other hyperparameters.

`init_type` How to initialize the model parameters. One of ‘'gamma'‘ (will initialize them ‘~ Gamma(1, 1))‘ or ‘'unif'‘ (will initialize them ‘~ Unif(0, 1))‘..

`seed` Random seed to use for starting the factorizing matrices.

`nthreads` Number of parallel threads to use.

Details

In order to obtain sparse latent factor matrices, you need to pass ‘method=’tncg’‘ and a large ‘niter‘, such as ‘niter=50‘ or ‘niter=100‘. The L1 regularization approach is not recommended, even though it might also produce relatively sparse results with the other optimization methods.

When using proximal gradient method, this model is prone to numerical instability, and can turn out to spit all NaNs or zeros in the fitted parameters. The conjugate gradient and Newton-CG methods are not prone to such failed optimizations.

Value

An object of class ‘poismf‘ with the following fields of interest:

Fields

`A` The user/document/row-factor matrix (as a vector in row-major order, has to be reshaped to (k, nrows) and then transposed to obtain an R matrix).

`B` The item/word/column-factor matrix (as a vector in row-major order, has to be reshaped to (k, ncols) and then transposed to obtain an R matrix).

`levels_A` A vector indicating which user/row ID corresponds to each row position in the ‘A‘ matrix. This will only be generated when passing ‘X‘ as a ‘data.frame‘, otherwise will not remap them.

`levels_B` A vector indicating which item/column ID corresponds to each row position in the ‘B‘ matrix. This will only be generated when passing ‘X‘ as a ‘data.frame‘, otherwise will not remap them.

References

- Cortes, David. "Fast Non-Bayesian Poisson Factorization for Implicit-Feedback Recommendations." arXiv preprint arXiv:1811.01908 (2018).
- Li, Can. "A conjugate gradient type method for the nonnegative constraints optimization problems." Journal of Applied Mathematics 2013 (2013).
- Carlsson, Christer, et al. "User’s guide for TN/TNBC: Fortran routines for nonlinear optimization." Mathematical Sciences Dept. Tech. Rep. 307, The Johns Hopkins University. 1984.

See Also

predict.poismf topN factors get.factor.matrices get.model.mappings

Examples

```
library(poismf)

### create a random sparse data frame in COO format
nrow <- 10^2 ## <- users
ncol <- 10^3 ## <- items
nnz  <- 10^4 ## <- events (agg)
set.seed(1)
```

```
X <- data.frame(
        row_ix = sample(nrow, size=nnz, replace=TRUE),
        col_ix = sample(ncol, size=nnz, replace=TRUE),
        count  = rpois(nnz, 1) + 1
     )
X <- X[!duplicated(X[, c("row_ix", "col_ix")]), ]

### can also pass X as sparse matrix - see below
### X <- Matrix::sparseMatrix(
###          i=X$row_ix, j=X$col_ix, x=X$count,
###          giveCsparse=FALSE)
### the indices can also be characters or other types:
### X$row_ix <- paste0("user", X$row_ix)
### X$col_ix <- paste0("item", X$col_ix)

### factorize the randomly-generated sparse matrix
### good speed (proximal gradient)
model <- poismf(X, k=5, method="pg", nthreads=1)

### good quality, but slower (conjugate gradient)
model <- poismf(X, k=5, method="cg", nthreads=1)

### better quality, but much slower (truncated Newton-CG)
model <- poismf(X, k=5, method="tncg", nthreads=1)


### for getting sparse factors
model <- poismf(X, k=50, method="tncg")
mean(model$A == 0.)


### predict functionality (chosen entries in X)
### predict entry [1, 10] (row 1, column 10)
predict(model, 1, 10)
### predict entries [1,4], [1,5], [1,6]
predict(model, c(1, 1, 1), c(4, 5, 6))

### ranking functionality (for recommender systems)
topN(model, user=2, n=5, exclude=X$col_ix[X$row_ix==2])
topN.new(model, X=X[X$row_ix==2, c("col_ix","count")],
    n=5, exclude=X$col_ix[X$row_ix==2])

### obtaining latent factors
a_vec  <- factors.single(model,
            X[X$row_ix==2, c("col_ix","count")])
A_full <- factors(model, X)
A_orig <- get.factor.matrices(model)$A

### (note that newly-obtained factors will differ slightly)
sqrt(mean((A_full["2",] - A_orig["2",])^2))
```

`poismf_unsafe` *Poisson factorization with no input casting*

Description

This is a faster version of poismf which will not make any checks or castings on its inputs. It is intended as a fast alternative when a model is to be fit multiple times with different hyperparameters, and for allowing custom-initialized factor matrices. **Note that since it doesn't make any checks or conversions, passing the wrong kinds of inputs or passing inputs with mismatching dimensions will crash the R process**.

For most use cases, it's recommended to use the function 'poismf' instead.

Usage

```
poismf_unsafe(A, B, Xcsr, Xcsc, k, ...)
```

Arguments

`A`	Initial values for the user-factor matrix of dimensions [dimA, k], assuming row-major order. Can be passed as a vector of dimension [dimA*k], or as a matrix of dimension [k, dimA]. Note that R matrices use column-major order, so if you want to pass an R matrix as initial values, you'll need to transpose it, hence the shape [k, dimA]. Recommended to initialize '~ Gamma(1,1)'. **Will be modified in-place**.
`B`	Initial values for the item-factor matrix of dimensions [dimB, k]. See documentation about 'A' for more details.
`Xcsr`	The **transpose** of the 'X' matrix in CSC format (so that its structure would match a CSR matrix). Should be an object of class 'Matrix::dgCMatrix'.
`Xcsc`	The 'X' matrix in CSC format. Should be an object of class 'Matrix::dgCMatrix'.
`k`	The number of latent factors. **Must match with the dimension of 'A' and 'B'**.
`...`	Other hyperparameters that can be passed to 'poismf'. See the documentation for poismf for details about possible hyperparameters. Init type and seed are ignored as the 'A' and 'B' matrices are supposed to be passed already-initialized.

Value

A 'poismf' model object. See the documentation for poismf for details.

See Also

poismf

Examples

```
library(poismf)

### create a random sparse data frame in COO format
nrow <- 10^2 ## <- users
ncol <- 10^3 ## <- items
nnz  <- 10^4 ## <- events (agg)
set.seed(1)
X <- data.frame(
        row_ix = sample(nrow, size=nnz, replace=TRUE),
        col_ix = sample(ncol, size=nnz, replace=TRUE),
        count  = rpois(nnz, 1) + 1
     )
X <- X[!duplicated(X[, c("row_ix", "col_ix")]), ]

### convert to required format
Xcsr <- Matrix::sparseMatrix(
            i=X$col_ix, j=X$row_ix, x=X$count,
            giveCsparse=TRUE
)
Xcsc <- Matrix::sparseMatrix(
            i=X$row_ix, j=X$col_ix, x=X$count,
            giveCsparse=TRUE
)

### initialize factor matrices
k <- 5L
A <- rgamma(nrow*k, 1, 1)
B <- rgamma(ncol*k, 1, 1)

### call function
model <- poismf_unsafe(A, B, Xcsr, Xcsc, k, nthreads=1)
```

`poisson.llk` *Evaluate Poisson log-likelihood for counts matrix*

Description

Calculates Poisson log-likehood plus constant for new combinations of rows and columns of 'X'. Intended to use as a test metric or for monitoring a validation set.

By default, this Poisson log-likelihood is calculated only for the combinations of users (rows) and items (columns) provided in 'X_test' here, ignoring the missing entries. This is the usual use-case for evaluating a validation or test set, but can also be used for evaluating it on the training data with all missing entries included as zeros (see parameters for details).

Note that this calculates a **sum** rather than an average.

Usage

```
poisson.llk(model, X_test, full_llk = FALSE, include_missing = FALSE)
```

Arguments

`model`	A Poisson factorization model object as returned by 'poismf'.
`X_test`	Input data on which to calculate log-likelihood, consisting of triplets. Can be passed as a 'data.frame' or as a sparse COO matrix (see documentation of poismf for details on the accepted data types). If the 'X' data passed to 'poismf' was a 'data.frame', should pass a 'data.frame' with entries corresponding to the same IDs, otherwise might pass either a 'data.frame' with the row and column indices (starting at 1), or a sparse COO matrix.
`full_llk`	Whether to add to the number a constant given by the data which doesn't depend on the fitted parameters. If passing 'False' (the default), there is some chance that the resulting log-likelihood will end up being positive - this is not an error, but is simply due to ommission of this constant. Passing 'TRUE' might result in numeric overflow and low numerical precision.
`include_missing`	If 'TRUE', will calculate the Poisson log-likelihood for all entries (i.e. all combinations of users/items, whole matrix 'X'), taking the missing ones as zero-valued. If passing 'FALSE', will calculate the Poisson log-likelihood only for the non-missing entries passed in 'X_test' - this is usually the desired behavior when evaluating a test dataset.

Details

If using more than 1 thread, the results might vary slightly between runs.

Value

Obtained Poisson log-likelihood (higher is better).

See Also

poismf

`predict.poismf` *Predict expected count for new row(user) and column(item) combinations*

Description

Predict expected count for new row(user) and column(item) combinations

Usage

```
## S3 method for class 'poismf'
predict(object, a, b = NULL, ...)
```

Arguments

`object`	A Poisson factorization model as returned by ‘poismf‘.
`a`	Can be either: • A vector of length N with the users/rows to predict - each entry will be matched to the corresponding entry at the same position in ‘b‘ - e.g. to predict value for entries (3,4), (3,5), and (3,6), should pass ‘a=c(3,3,3), b=c(3,5,6)‘. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, should match with the entries in its first column. If ‘X‘ passed to ‘poismf‘ was a matrix, should indicate the row numbers (numeration starting at 1). • A sparse matrix, ideally in COO (triplets) format from package ‘Matrix‘ (‘Matrix::dgTMatrix‘) or from package ‘SparseM‘ (‘matrix.coo‘), in which case it will make predictions for the non-zero entries in the matrix and will output another sparse matrix with the predicted entries as values. In this case, ‘b‘ should not be passed. This option is not available if the ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘.
`b`	A vector of length N with the items/columns to predict - each entry will be matched to the corresponding entry at the same position in ‘a‘ - e.g. to predict value for entries (3,4), (3,5), and (3,6), should pass ‘a=c(3,3,3), b=c(3,5,6)‘. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, should match with the entries in its second column. If ‘X‘ passed to ‘poismf‘ was a matrix, should indicate the column numbers (numeration starting at 1). If ‘a‘ is a sparse matrix, should not pass ‘b‘.
`...`	Not used.

Value

- If ‘a‘ and ‘b‘ were passed, will return a vector of length N with the predictions for the requested row/column combinations.
- If ‘b‘ was not passed, will return a sparse matrix with the same entries and shape as ‘a‘, but with the values being the predictions from the model for the non-missing entries.

See Also

poismf topN factors

`print.poismf` *Get information about poismf object*

Description

Print basic properties of a "poismf" object.

Usage

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

Arguments

x	An object of class "poismf" as returned by function "poismf".
...	Extra arguments (not used).

---

summary.poismf *Get information about poismf object*

Description

Print basic properties of a "poismf" object (same as ‘print.poismf‘ function).

Usage

```
## S3 method for class 'poismf'
summary(object, ...)
```

Arguments

object	An object of class "poismf" as returned by function "poismf".
...	Extra arguments (not used).

See Also

print.poismf

---

topN *Rank top-N highest-predicted items for an existing user*

Description

Rank top-N highest-predicted items for an existing user

Usage

```
topN(model, user, n = 10, include = NULL, exclude = NULL, output_score = FALSE)
```

Arguments

model	A Poisson factorization model as returned by ‘poismf‘.
user	User for which to rank the items. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, must match with the entries in its first column, otherwise should match with the rows of ‘X‘ (numeration starting at 1).
n	Number of top-N highest-predicted results to output.
include	List of items which will be ranked. If passing this, will only make a ranking among these items. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, must match with the entries in its second column, otherwise should match with the columns of ‘X‘ (numeration starting at 1). Can only pass one of ‘include‘ or ‘exclude.‘ Must not contain duplicated entries.
exclude	List of items to exclude from the ranking. If passing this, will rank all the items except for these. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, must match with the entries in its second column, otherwise should match with the columns of ‘X‘ (numeration starting at 1). Can only pass one of ‘include‘ or ‘exclude‘. Must not contain duplicated entries.
output_score	Whether to output the scores in addition to the IDs. If passing ‘FALSE‘, will return a single array with the item IDs, otherwise will return a list with the item IDs and the scores.

Value

- If passing ‘output_score=FALSE‘ (the default), will return a vector of size ‘n‘ with the top-N highest predicted items for this user.If the ‘X‘ data passed to ‘poismf‘ was a ‘data.frame‘, will contain the item IDs from its second column, otherwise will be integers matching to the columns of ‘X‘ (starting at 1). If ‘X‘ was passed as ‘data.frame‘, the entries in this vector might be coerced to character regardless of their original type.
- If passing ‘output_score=TRUE‘, will return a list, with the first entry being the vector described above under name ‘ix‘, and the second entry being the associated scores, as a numeric vector of size ‘n‘.

See Also

topN.new predict.poismf factors.single

---

topN.new *Rank top-N highest-predicted items for a new user*

Description

Rank top-N highest-predicted items for a new user

Usage

```
topN.new(
  model,
  X,
  n = 10,
  include = NULL,
  exclude = NULL,
  output_score = FALSE,
  l2_reg = model$l2_reg,
  l1_reg = model$l1_reg,
  weight_mult = model$weight_mult,
  maxupd = model$maxupd
)
```

Arguments

model	A Poisson factorization model as returned by ‘poismf‘.
X	Data with the non-zero item indices and counts for this new user. Can be passed as a sparse vector from package ‘Matrix‘ (‘Matrix::dsparseVector‘, which can be created from indices and values through ‘Matrix::sparseVector‘), or as a ‘data.frame‘, in which case will take the first column as the item/column indices (numeration starting at 1) and the second column as the counts. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, ‘X‘ here must also be a ‘data.frame‘.
n	Number of top-N highest-predicted results to output.
include	List of items which will be ranked. If passing this, will only make a ranking among these items. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, must match with the entries in its second column, otherwise should match with the columns of ‘X‘ (numeration starting at 1). Can only pass one of ‘include‘ or ‘exclude.‘ Must not contain duplicated entries.
exclude	List of items to exclude from the ranking. If passing this, will rank all the items except for these. If ‘X‘ passed to ‘poismf‘ was a ‘data.frame‘, must match with the entries in its second column, otherwise should match with the columns of ‘X‘ (numeration starting at 1). Can only pass one of ‘include‘ or ‘exclude‘. Must not contain duplicated entries.
output_score	Whether to output the scores in addition to the IDs. If passing ‘FALSE‘, will return a single array with the item IDs, otherwise will return a list with the item IDs and the scores.
l2_reg	Strength of L2 regularization to use for optimizing the new factors.
l1_reg	Strength of the L1 regularization. Not recommended.
weight_mult	Weight multiplier for the positive entries over the missing entries.
maxupd	Maximum number of Newton-CG updates to perform. You might want to increase this value depending on the use-case.

Details

This function calculates the latent factors in the same way as ‘factors.single‘ - see the documentation of factors.single for details.

The factors are initialized to the mean of each column in the fitted model.

Value

- If passing ‘output_score=FALSE‘ (the default), will return a vector of size ‘n‘ with the top-N highest predicted items for this user.If the ‘X‘ data passed to ‘poismf‘ was a ‘data.frame‘, will contain the item IDs from its second column, otherwise will be integers matching to the columns of ‘X‘ (starting at 1). If ‘X‘ was passed as ‘data.frame‘, the entries in this vector might be coerced to character regardless of their original type.
- If passing ‘output_score=TRUE‘, will return a list, with the first entry being the vector described above under name ‘ix‘, and the second entry being the associated scores, as a numeric vector of size ‘n‘.

See Also

factors.single topN

Index

factors, 2, *3, 4*, *9*, *14*
factors.single, *2, 3*, 3, *16–18*

get.all.predictions, 4
get.factor.matrices, 5, *6*, *9*
get.model.mappings, *5*, 6, *9*

poismf, *2*, 6, *11*, *13, 14*
poismf_unsafe, 11
poisson.llk, 12
predict.poismf, *9*, 13, *16*
print.poismf, 14, *15*

summary.poismf, 15

topN, *9*, *14*, 15, *18*
topN.new, *3, 4*, *16*, 16