Package 'rrecsys'

June 27, 2016

Type Package

Title Environment for Assessing Recommender Systems

Version 0.9.5.4

Date 2016-06-26

URL https://github.com/ludovikcoba/rrecsys

BugReports https://github.com/ludovikcoba/rrecsys/issues

Description Provides implementations of several popular recommendation systems. They can process standard recommendation datasets (user/item matrix) as input and generate rating predictions and recommendation lists. Standard algorithm implementations included in this package are: Global/Item/User-Average baselines, Item-
Based KNN, FunkSVD, BPR and weighted ALS. They can be assessed according to the standard offline evaluation methodology for recommender systems using measures such as MAE, RMSE, Precision, Recall, AUC, NDCG, RankScore and coverage measures. The package is intended for rapid prototyping of recommendation algorithms and education purposes.

Imports methods

Depends R (>= 3.1.2), registry, proxy, MASS, stats, knitr

License GPL-3

VignetteBuilder knitr

Encoding UTF-8

Repository CRAN

Suggests covr

NeedsCompilation no

Author Ludovik Çoba [aut, cre, cph],
Markus Zanker [ctb]

Maintainer Ludovik Çoba <lcoba@unishk.edu.al>

Date/Publication 2016-06-27 17:31:21

R topics documented:

`algAverageClass` *Baseline algorithms exploiting global/item and user averages.*

Description

Container for the model learned using any average(golbal, user or item) based model.

Slots

`alg`: The algorithm denominator, of class `"character"`.

`data`: the dataset used for training the model, class `"matrix"`.

`average`: average calculated either globaly, on user or item, class `"matrix"`.

Methods

`show` signature(object = "algAverageClass")

See Also

`rrecsys`.

BPRclass *Bayesian Personalized Ranking based model.*

Description

Container for the model learned using any Bayesian Personalized Ranking based model.

Slots

`alg`: The algorithm denominator, of class `"character"`.

`data`: the dataset used for training the model, class `"matrix"`.

`factors`: user(U) and items(V) factors, class `"list"`.

`parameters`: the parameters(such as number of factors `k`, learning rate `lambda`, user regularization term `regU`, positive rated item regularization term `regI`, negative rated item regularization term `regJ` and the boolean `updateJ` to decide whatever negative updates are required) used in the model, class `"list"`.

Methods

`show` signature(object = "BPRclass")

See Also

`rrecsys`.

dataSet-class *Dataset class.*

Description

Defines a structure for a dataset that distinguishes between binary and non-binary feedback datasets.

Slots

`data`: the dataset, class `"matrix"`.

`binary`: class `"logical"`, determines if the item dataset contains binary (i.e. 1/0) or non-binary ratings.

`minimum`: class `"numeric"`, defines the minimal value present in the dataset.

`maximum`: class `"numeric"`, defines the maximal value present in the dataset.

`halfStar`: object of class `"logical"`, if **TRUE** the range of ratings in the dataset contains as well half star values.

Methods

show signature(object = "dataSet")

nrow signature(object = "dataSet"): number of rows of the dataset.

ncol signature(object = "dataSet"): number of columns of the dataset.

dim signature(object = "dataSet"): returns the dimensions of the dataset.

rowRatings signature(object = "dataSet"): returns the number of ratings on each row.

colRatings signature(object = "dataSet"): returns the number of ratings on each column.

numRatings signature(object = "dataSet"): returns the total number of ratings.

[signature(x = "dataSet", i = "ANY", j = "ANY", drop = "ANY")): returns a subset of the dataset.

sparsity signature(object = "dataSet"): returns the sparsity of the dataset.

coerce signature(from = "dataSet", to = "matrix")

Examples

```
x <- matrix(sample(c(0:5), size = 100, replace = TRUE,
  prob = c(.6,.08,.08,.08,.08,.08)), nrow = 20, byrow = TRUE)

x <- defineData(x)

colRatings(x)

rowRatings(x)

numRatings(x)

sparsity(x)

a <- x[1:10,2:3]
```

`defineData` *Define dataset.*

Description

Defines your dataset, if either it is implicit or explicit.

Usage

```
defineData(data, binary = FALSE, minimum = 1, maximum = 5,
  halfStar = FALSE, goodRating = 0.5)
```

Arguments

data	the dataset, class "matrix".
binary	class "logical", defines if the item dataset consists of binary (i.e. 0/1) or non-binary ratings. Deafault value FALSE.
minimum	class "numeric", defines the minimal value present in the dataset. Deafault value 1.
maximum	class "numeric", defines the maximal value present in the dataset. Deafault value 5.
halfStar	object of class "logical", if **TRUE** the range of ratings in the dataset contains as well half star values. Deafault value FALSE.
goodRating	class "numeric", in case binary is TRUE, goodRating defines the threshold value for binarizing the dataset (i.e. any rating value >= goodRating will be transformed to 1 and all other values to 0(corresponding to a not rated item). Deafault value 0.5.

Value

Returns an object of class "dataSet".

See Also

See Also as dataSet-class.

Examples

```
data(mlLatest100k)

a <- defineData(mlLatest100k)

b <- defineData(mlLatest100k, TRUE, goodRating = 3)
```

---

evalModel *Creating the evaluation model.*

Description

Creates the dataset split for evaluation where ratings of each user are uniformly distributed over k random folds. The function returns the list of items that are assigned to each fold, such that algorithms can be compared on the same train/test splits.

Usage

```
evalModel(data, folds)
```

Arguments

data	dataset, of class dataSet.
folds	The number of folds to use in the k-fold cross validation, of class numeric, default value set to 5.

Value

An object of class evalModel-class.

See Also

evalModel-class, evalRec, dataSet.

Examples

```
x <- matrix(sample(c(0:5), size = 200, replace = TRUE,
     prob = c(.6,.08,.08,.08,.08,.08)), nrow = 20, byrow = TRUE)

x <- defineData(x)

my_10_folds <- evalModel(x, 10)             #output class evalModel.

my_6_folds <- evalModel(x, 6)

my_6_folds
#6 - fold cross validation model on the dataset with 20 users and 10 items.

my_6_folds@data                     #the dataset.
my_6_folds@folds                    #the number of folds in the model.
my_6_folds@fold_indices               #the index of each item in the fold.
```

---

evalModel-class *Evaluation model.*

Description

Class that contains the data and a distribution of the uniform distribution of ratings onto k-folds.

Details

The fold_indices list contains the indexes to access the dataset on one dimension. A matrix can be addressed as a one dimensional array, considered as an extension of each column after another. E.g: in a matrix M with 10 rows and 20 columns, M[10] == M[10, 1]; M[12] == M[2,2].

Slots

`data`: the dataset, class `"matrix"`.

`folds`: number of k - folds, class `"numeric"`.

`fold_indices`: a list with k slots, each slot represents a fold and contains the index of items assigned to that fold, class `"list"`.

`fold_indices_x_user`: a list that specifies specificly for each user the distribution of the items in the folds, class `"list"`.

Methods

`show` signature(object = "evalModel")

`evalPred` *Evaluates the requested prediction algorithm.*

Description

Evaluates the prediction task of an algorithm with a given configuration and based on the given evaluation model. RMSE and MAE are both calculated individually for each user and then averaged over all users (in this case they will be referred as RMSE and MAE) as well as determined as the average error over all predictions (in this case they are named globalRMSE and globalMAE).

Usage

```
evalPred(model, ...)
## S4 method for signature 'evalModel'
evalPred(model, alg, ... )
```

Arguments

`model`	Object of type `evalModel`. See `evalModel-class`.
`alg`	The algorithm to be used in the evaluation. Of type `character`.
`...`	other attributes specific to the algorithm to be deployed. Refer to `rrecsys`.

Value

Returns a data frame with the `RMSE`, `MAE`, `globalRMSE` and `globalMAE` for each of the k-folds definded in the evaluation model and an average over all folds.

References

F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011. ISBN 978-0-387-85819-7. URL http://www.springerlink.com/content/978-0-387-85819-7.

See Also

evalModel-class, rrecsys.

Examples

```
x <- matrix(sample(c(0,1), size = 200, replace = TRUE,
      prob = c(.8,.2)), nrow = 20, byrow = TRUE)

x <- defineData(x)

e <- evalModel(x, 5)

SVDEvaluation <- evalPred(e, "FunkSVD", k = 4)

SVDEvaluation


IBEvaluation <- evalPred(e, "IBKNN", neigh = 5)

IBEvaluation
```

---

evalRec *Evaluates the requested recommendation algorithm.*

Description

Evaluates the recommendation task of an algorithm with a given configuration and based on the given evaluation model.

Usage

```
evalRec(model, ...)
## S4 method for signature 'evalModel'
evalRec(model, alg, topN, goodRating, alpha, ... )
```

Arguments

model	Object of type evalModel. See evalModel-class.
alg	The algorithm to be used in the evaluation. Of class character.
topN	Object of class numeric, specifying the number of items to be recommended per user.
goodRating	Object of class numeric, indicating the threshold of the ratings to be considered a good. This attribute is not used when evaluating implicit feedback.
alpha	Object of class numeric, is the half-life parameter for the rankscore metric.
...	other attributes specific to the algorithm to be deployed. Refer to rrecsys.

Value

Returns a data frame with the precision, recall, F1, nDCG, RankScore, true positives(TP), false positives(FP), true negatives(TN), false negatives(FN) for each of the k-folds definded in the evaluation model and the overall average.

References

F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011. ISBN 978-0-387-85819-7. URL http://www.springerlink.com/content/978-0-387-85819-7.

See Also

evalModel-class, rrecsys.

Examples

```
x <- matrix(sample(c(0:5), size = 200, replace = TRUE,
      prob = c(.6,.8,.8,.8,.8,.8)), nrow = 20, byrow = TRUE)

x <- defineData(x)

e <- evalModel(x, 5)

SVDEvaluation <- evalRec(e, "FunkSVD", goodRating = 4, k = 4)

SVDEvaluation


IBEvaluation <- evalRec(e, "IBKNN", goodRating = 4, neigh = 5)

IBEvaluation
```

---

getAUC *Returns the Area under the ROC curve.*

Description

Computes the Area Under the ROC curve for a recommendation task of an algorithm with its given configuration and based on the given evaluation model.

Usage

```
getAUC(model, ...)
## S4 method for signature 'evalModel'
getAUC(model, alg, ... )
```

Arguments

model	Object of type evalModel. See evalModel-class.
alg	The algorithm to be used in the evaluation. Of class character.
...	other attributes specific to the algorithm to be deployed. Refer to rrecsys.

Value

Returns a data frame with the AUC for each of the k-folds definded in the evaluation model and the overall average.

References

T. Fawcett, *"ROC Graphs: Notes and Practical Considerations for Data Mining Researchers ROC Graphs : Notes and Practical Considerations for Data Mining Researchers,"*, HP Inven., p. 27, 2003.

See Also

evalModel-class, rrecsys.

Examples

```
x <- matrix(sample(c(0:5), size = 200, replace = TRUE,
      prob = c(.6,.8,.8,.8,.8,.8)), nrow = 20, byrow = TRUE)

x <- defineData(x)

e <- evalModel(x, 5)

auc <- getAUC(e, "FunkSVD", k = 4)

auc
```

IBclass *Item based model.*

Description

Container for the model learned using any k-nearest neighbor item-based collaborative filtering algorithm.

Slots

`alg`: The algorithm denominator, of class `"character"`.

`data`: the dataset used for training the model, class `"matrix"`.

`sim`: The item - item similarity matrix, class `"matrix"`.

`sim_index_kNN`: The index of the k nearest neighbors for each item, class `"matrix"`.

`neigh`: The size of the neighborhood, class `"numeric"`.

Methods

`show` signature(object = "IBclass")

See Also

`rrecsys`.

`mlLatest100k` *Movielens Latest*

Description

This dataset (ml-latest-small) is a 5-star rating dataset from [MovieLens](http://movielens.org), a movie recommendation service of the GroupLens research group at the University of Minnesota. It contains 100234 ratings across 8927 movies. The data was created by 718 users between March 26, 1996 and August 05, 2015. This dataset was generated on August 06, 2015. Users were selected at random for inclusion. All selected users had rated at least 20 movies. The data is edited and structured as a matrix and distributed as such. Below the usage license of this redistributed data is cited below.

Usage

```
data("mlLatest100k")
```

Format

The format is: num [1:718, 1:8915] 5 3 0 0 4 4 0 3 0 0 ... - attr(*, "dimnames")=List of 2 ..$: chr [1:718] "1" "2" "3" "4" ... ..$: chr [1:8915] "Toy Story (1995)" "Jumanji (1995)" "GoldenEye (1995)" "Twelve Monkeys (a.k.a. 12 Monkeys) (1995)" ...

Usage License

Neither the University of Minnesota nor any of the researchers involved can guarantee the correctness of the data, its suitability for any particular purpose, or the validity of results based on the use of the data set. The data set may be used for any research purposes under the following conditions:

1. The user may not state or imply any endorsement from the University of Minnesota or the GroupLens Research Group.

2. The user must acknowledge the use of the data set in publications resulting from the use of the data set, and must send us an electronic or paper copy of those publications.
3. The user may redistribute the data set, including transformations, so long as it is distributed under these same license conditions.
4. The user may not use this information for any commercial or revenue-bearing purposes without first obtaining permission from a faculty member of the GroupLens Research Project at the University of Minnesota.
5. The executable software scripts are provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of them is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

In no event shall the University of Minnesota, its affiliates or employees be liable to you for any damages arising out of the use or inability to use these programs (including but not limited to loss of data or data being rendered inaccurate). If you have any further questions or comments, please email <grouplens-info@cs.umn.edu>.

Source

http://grouplens.org/datasets/movielens/latest/

---

nDCG *Normalized Discounted Cumulative Gain*

Description

Metric for information retrival where positions are discounted logarithmically.

Usage

```
nDCG(recommendedIDX, testSetIDX)
```

Arguments

recommendedIDX	indices of the recommended items. Object of class numeric.
testSetIDX	indices of the items in the test set. Object of class numeric

Details

nDCG is computed as the ratio between Discounted Cumulative Gain(DCG) and idealized Discounted Cumulative Gain(IDCG):

$$DGC_{pos} = rel_1 + \sum_{i=2}^{pos} \frac{rel_i}{\log_2 i}$$

$$IDGC_{pos} = rel_1 + \sum_{i=2}^{|h|-1} \frac{rel_i}{\log_2 i}$$

$$nDCG_{pos} = \frac{DCG}{IDCG}$$

References

Asela Gunawardana, Guy Shani, Evaluating Recommender Systems.

`PPLclass` *Popularity based model.*

Description

Container for the model learned by an unpersonalized popularity-based algorithm.

Slots

`alg`: The algorithm denominator, of class `"character"`.

`data`: the dataset used for training the model, class `"matrix"`.

`indices`: the indices of items ordered by popularity, class `"integer"`.

Methods

`show` signature(object = "PPLclass")

See Also

`rrecsys`.

`predict` *Generate predictions.*

Description

Generate predictions on any of the previously trained models.

Arguments

`model`	A previosly trained model, see `rrecsys`
`Round`	object of class `"logical"`, if **TRUE** all the predictions are rounded to integer values, else values are returned as calculeted.

Value

All unrated items are predicted and the entire matrix is returned with the new ratings.

See Also

rrecsys, IBclass, SVDclass.

Examples

```
data("mlLatest100k")

smallMl <- mlLatest100k[1:50, 1:100]

exExpl <- defineData(smallMl)

model1exp <- rrecsys(exExpl, alg = "funk", k = 10, gamma = 0.1, lambda = 0.001)

pre1 <- predict(model1exp, Round = TRUE)
```

---

rankScore *Rank Score*

Description

Rank Score extends the recall metric to take the positions of correct items in a ranked list into account.

Usage

```
rankScore(recommendedIDX, testSetIDX, alpha)
```

Arguments

recommendedIDX	indices of the recommended items. Object of class numeric.
testSetIDX	indices of the items in the test set. Object of class numeric
alpha	is the ranking half life. Object of class numeric.

Details

Rank Score is defined as the ratio of the Rank Score of the correct items to best theoretical Rank Score achievable for the user:

$$rankscore_p = \sum_{i \in h} 2^{-\frac{rank(i)-1}{\alpha}}$$

$$rankscore_{max} = \sum_{i=1}^{|T|} 2^{-\frac{i-1}{\alpha}}$$

$$rankscore = \frac{rankscore_p}{rankscore_{max}}$$

`recommend` *Generate recommendation.*

Description

This method generates top-n recommendations based on a model that has been trained before.

Usage

```
recommend(model, topN = 3)
```

Arguments

`model`	the trained model of any algorithm.
`topN`	number of items to be recommended per user, class `numeric`.

Value

Returns a list with suggested items for each user.

See Also

`rrecsys`.

Examples

```
myratings <- matrix(sample(c(0:5), size = 200, replace = TRUE,
        prob = c(.6,.08,.08,.08,.08,.08)), nrow = 20, byrow = TRUE)

myratings <- defineData(myratings)

r <- rrecsys(myratings, alg = "FunkSVD", k = 2)

rec <- recommend(r)
```

recResultsClass *Results of a recommendation.*

Description

Defines a structure for the results of a recommendation algorithm.

Slots

`indices`**:** indices of the recommended items for each user, class `"list"`.

`recommended`**:** names, if aviable, of the recommended items for each user, class `"list"`.

Methods

`show` signature(object = "recResultsClass")

`[` signature(object = "dataSet", i = "ANY""): returns a subset of the recommended items.

Examples

```
x <- matrix(sample(c(0:5), size = 100, replace = TRUE,
  prob = c(.6,.08,.08,.08,.08,.08)), nrow = 20, byrow = TRUE)

colnames(x) <- paste0(rep("item-", 5), 1:5)
rownames(x) <- paste0(rep("user-", 20), 1:20)

x <- defineData(x)

model <- rrecsys(x, alg = "funk", k = 2, gamma = 0.1, lambda = 0.001)

rec <- recommend(model, topN = 1)

rec

rec[c(1,4,6)]
```

rrecsys *Create a recommender system.*

Description

Based on the specific given algorithm a recommendation model will be trained.

Usage

```
rrecsys(data, alg, ...)
```

Arguments

`data`	Training set of class `"matrix"`. The columns correspond to items and the rows correspond to users.
`alg`	A `"character"` string specifying the recommender algorithm to apply on the data.
`...`	other attributes, see details.

Details

Based on the value of *alg* the attributes will have different names and values. Possible configuration of `alg` and it's meaning:

1. **itemAverage**. When `alg = "itemAverage"` the average rating of an item is used to make predictions and recommendations.
2. **userAverage**. When `alg = "userAverage"` the average rating of a user is used to make predictions and recommendations.
3. **globalAverage**. When `alg = "globalAverage"` the overall average of all ratings is used to make predictions and recommendations.
4. **Mostpopular**. The most popular algorithm (`alg = "mostpopular"`) is the most simple algorithm for recommendations. Item will be ordered based on the number of times that they were rated. Recommendations for a particular user will be the most popular items from the data set which are not contained in the user's training set.
5. **IBKNN**. As `alg = "IBKNN"` a k-nearest neighbor item-based collaborative filtering algorithm. It determines the "adjusted cosine" distance among the items as:

$$sim(\vec{a}, \vec{b}) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u}) * (r_{u,b} - \overline{r_u})}{\sqrt{(r_{u,a} - \overline{r_u})^2} * \sqrt{(r_{u,b} - \overline{r_u})^2}}$$

   The items *a* and *b* are considered as the corresponding rating vectors $\vec{a}$ and $\vec{b}$. It extracts, based on the value of the *neigh* attribute, the number of closest neighbors for each item.
6. **FunkSVD**. It implements `alg = "funkSVD"` a stochastic gradient descent optimization technique. The U(user) and V(item) factor matrices are initialized at small values and cropped to *k* features. Each feature is trained until *convergence* (the convergence value has to be specified by the user). On each loop the algorithm predicts r'_{ui} and calculates the error as:

$$r'_{ui} = u_u * v_i^T$$

$$e_{ui} = r_{ui} - r'_{ui}$$

   The factors are updated:

$$v_{ik} \leftarrow v_{ik} + lambda * (e_{ui} * u_{uk} - gamma * v_{ik})$$

$$u_{uk} \leftarrow u_{uk} + lambda * (e_{ui} * v_{ik} - gamma * u_{uk})$$

   . The attribute *lambda* represents the learning rate, while *gamma* corresponds to the weight of the regularization term.

7. **wALS**. The `alg = "wALS"` weighted Alternated Least squares method. For a given non-negative weight matrix *W* the algorithm will perform updates on the item *V* and user *U* feature matrix as follows:

$$U_i = R_i * \widetilde{W}_i * V * (V^T * \widetilde{W}_i * V + lambda(\sum_j W_{ij})I)^{-1}$$

$$V_j = R_j^T * \widetilde{W}_j * U * (V^T * \widetilde{W}_j * u + lambda(\sum_i W_{ij})I)^{-1}$$

   Initialy the *V* matrix is initialized with Guassian random numbers with mean zero and small standard deviation. Than *U* and *V* are updated until `convergence`. The attribute `scheme` must specify the scheme(`uni, uo, io, co`) to use.

8. **BPR**. In this implementation of BPR(`alg = "BPR"`) is applied a stochastic gradient descent approach that randomly choose triples from D_R and trains the model Θ. In this implementation the BPR optimization criterion is applied on matrix factorization. If $R = U \times V^T$, where *U* and *V* are the usual feature matrix cropped to *k* features, the parameter vector of the model is $\Theta = \langle U, V \rangle$. The algorithm will use three regularization terms, `RegU` for the user features *U*, `RegI` for positive updates and `RegJ` for negative updates of the item features *V*, `lambda` is the learning rate, `autoConvergence` is a toggle to the auto convergence validation, `convergence` upper limit to the convergence, and `updateJ` if true updates negative item features.

To receive a list of all algorithms and their default configuration a call to `rrecsysRegistry` is advised.

Value

Depending on the `alg` value it will be either an object of type `SVDclass` or `IBclass`.

References

D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, New York, NY, USA, 1st edition, 2010. ISBN 978-0-521-49336-9.

Funk, S., 2006, *Netflix Update: Try This at Home,* http://sifter.org/~simon/journal/20061211.html.

Y. Koren, R. Bell, and C. Volinsky. *Matrix Factorization Techniques for Recommender Systems*. Computer, 42(8):30–37, Aug. 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. http://dx.doi.org/10.1109/MC.2009.263.

R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, and Q. Yang. *One-Class Collaborative Filtering*. In Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on, pages 502–511, Dec 2008. doi: 10.1109/ICDM.2008.16.

S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. *BPR: Bayesian Personalized Ranking from Implicit Feedback*. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press. ISBN 978-0-9749039-5-8. URL http://dl.acm.org/citation.cfm?id=1795114.1795167.

Examples

```
myratings <- matrix(sample(c(0:5), size = 200, replace = TRUE,
        prob = c(.6,.08,.08,.08,.08,.08)), nrow = 20, byrow = TRUE)

myratings <- defineData(myratings)

r <- rrecsys(myratings, alg = "funkSVD", k = 2)

r1 <- rrecsys(myratings, alg = "funkSVD", k = 3,
              gamma = 0.01, lambda = 0.002)

r2 <- rrecsys(myratings, alg = "IBKNN", neigh = 5)

rrecsysRegistry$get_entries()
```

---

setStoppingCriteria *Set stopping criteria.*

Description

Define stopping criteria for functions that need a convergence check.

Usage

```
setStoppingCriteria(autoConverge = FALSE,
  deltaErrorThreshold = 1e-05, nrLoops = NULL, minNrLoops = 10)
showStoppingCriteria()
showDeltaError()
```

Arguments

autoConverge	class "logical", turns on the autoconvergence algorithm.
deltaErrorThreshold	class "numeric", is the threshold for the autoconvergence algorithm.
nrLoops	class "numeric", number of loops that will be performed in case autoConvergence is FALSE
minNrLoops	class "numeric", the minimum number of loops to consider before verifying the deltaErrorThreshold.

Details

If autoConvergence = TRUE tells the package to monitor the difference of global RMSE on two consecutive iterations, and to see if it drops below a threshold value. Whenever it drops under the specified value the iteration is considered converged. If FALSE the limit of iterations is delimited by nrLoops

Methods

showStoppingCriteria Print on console the current configuration of the convergence algorithm.

showDeltaError Report the delta error on each iteration of the algorithm that requires an autoconvergence algorithm.

References

M. D. Ekstrand, M. Ludwig, J. Kolb, and J. T. Riedl, *"LensKit: a modular recommender framework,"*, Proc. fifth ACM Conf. Recomm. Syst. - RecSys '11, p. 349, 2011.

See Also

See Also as rrecsys, SVDclass, wALSclass, BPRclass.

Examples

```
setStoppingCriteria(autoConverge = TRUE)

setStoppingCriteria(nrLoops = 30)
```

SVDclass *SVD model.*

Description

Container for the model learned using any matrix factorization algorithm.

Slots

alg: The algorithm denominator, of class "character".

data: the dataset used for training the model, class "matrix".

factors: user(U) and items(V) factors, class "list".

parameters: the parameters(such as number of factors k, learning rate lambda, regularization term gamma and number of iterations until convergence) used in the model, class "list".

Methods

show signature(object = "SVDclass")

See Also

rrecsys.

wALSclass *Weighted Alternating Least Squares based model.*

Description

Container for the model learned using any weighted Alternating Least Squares based algorithm.

Slots

`alg`: The algorithm denominator, of class `"character"`.

`data`: the dataset used for training the model, class `"matrix"`.

`factors`: user(U) and items(V) factors, class `"list"`.

`weightScheme`: The weighting scheme used in updating the factors, class `"matrix"`.

`parameters`: the parameters(such as number of factors `k`, learning rate `lambda`, number of iterations until `convergence` and the weighting scheme) used in the model, class `"list"`.

Methods

`show` signature(object = "wALSclass")

See Also

`rrecsys`.

Index