Package 'rrecsys'

June 9, 2019

Type Package

Title Environment for Evaluating Recommender Systems

Version 0.9.7.3.1

Date 2018-02-10

URL https://rrecsys.inf.unibz.it/

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

Description Processes standard recommendation datasets (e.g., a user-item rating matrix) as input and generates rating predictions and lists of recommended items. Standard algorithm implementations which are included in this package are the following: Global/Item/User-Average baselines, Weighted Slope One, Item-Based KNN, User-Based KNN, FunkSVD, BPR and weighted ALS. They can be assessed according to the standard offline evaluation methodology (Shani, et al. (2011) <doi:10.1007/978-0-387-85820-3_8>) for recommender systems using measures such as MAE, RMSE, Precision, Recall, F1, AUC, NDCG, RankScore and coverage measures. The package (Coba, et al.(2017) <doi: 10.1007/978-3-319-60042-0_36>) is intended for rapid prototyping of recommendation algorithms and education purposes.

Imports methods, Rcpp

Depends R (>= 3.1.2), registry, MASS, stats, knitr, ggplot2

License GPL-3

VignetteBuilder knitr

Encoding UTF-8

Repository CRAN

LinkingTo Rcpp

NeedsCompilation yes

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

Maintainer Ludovik Çoba <Ludovik.Coba@inf.unibz.it>

Date/Publication 2019-06-09 18:45:49 UTC

R topics documented:

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

Description

Container for the model learned using any average(global, 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 globally, 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.

---

dataChart *Visualization of data characteristics.*

Description

This method visualizes data characteristics on a two dimensional graph, where "x" axes shows either items ordered by descending popularity, or users based on the number of ratings they have submitted. Moreover the "y" axes shows the number of ratings.

Usage

```
dataChart(data, x = "items", y = "num_of_ratings")
```

Arguments

data	the dataset, class "_ds".
x	class "character", is the variable that will be shown on the "x" axis. Possible values are: "items", "users".
y	class "character", is the variable that will be shown on the "y" axis. Possible values are: "num_of_ratings", "%_of_ratings".

Value

Plot results.

See Also

See Also as _ds-class.

Examples

```
data(mlLatest100k)

a <- defineData(mlLatest100k)

dataChart(a, x = "items", y = "num_of_ratings")
```

---

dataSet-class *Dataset class.*

Description

Container for a dense dataset that distinguishes between binary and non-binary feedback datasets. Extends _ds.

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.

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

Methods

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.

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

rowAverages signature(object = "dataSet"): returns the average rating on each row.

colAverages signature(object = "dataSet"): returns the average rating on each column.

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.

Arguments

data	the dataset, class "matrix".
sparseMatrix	class "logical". If FALSE implies that the imput is a dense two dimensional matrix. If TRUE implies that the imput is arranges as coordinate list where entries are stored as list of (row, column, value) tuples.

`binary` class "logical", defines if the item dataset consists of binary (i.e. NA/1) or non-binary ratings. Default value FALSE.

`minimum` class "numeric", defines the minimal value present in the dataset. Default value 0.5.

`maximum` class "numeric", defines the maximal value present in the dataset. Default value 5.

`intScale` object of class "logical", if **TRUE** the range of ratings in the dataset contains as well half star values. Default value FALSE.

`positiveThreshold` class "numeric", in case binary is TRUE, positiveThreshold defines the threshold value for binarizing the dataset (i.e. any rating value >= positiveThreshold will be transformed to 1 and all other values to NA(corresponding to a not rated item). Default 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,binary = TRUE ,positiveThreshold = 3)
```

---

evalChart *Visualization of data characteristics.*

Description

This method visualizes data characteristics on a two dimensional graph, where "x" axes shows either items ordered by descending popularity, or users based on the number of ratings they have submitted. Moreover the "y" axes shows the number of ratings.

Usage

```
evalChart(res, x = "items", y = "TP", x_label, y_label, y_lim)
```

Arguments

res	evaluation results, class "evalRecResults".
x	class "character", is the variable that will be shown on the "x" axis. Possible values are: "items", "users".
y	class "character", is the variable that will be shown on the "y" axis. Possible values are: "num_of_ratings", "%_of_ratings".
x_label	class "character", the label to be printed on the "x" axes.
y_label	class "character", the label to be printed on the "y" axes.
y_lim	class "numeric", scale of the "y" axes.

Value

Plot results.

See Also

See Also as evalRecResults-class.

---

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 _ds.
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, _ds.

Examples

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

d <- defineData(x)

my_2_folds <- evalModel(d, 2)             #output class evalModel.

my_2_folds
# 2 - fold cross validation model on the dataset with 20 users and 10 items.

my_2_folds@data                     #the dataset.
my_2_folds@folds                    #the number of folds in the model.
my_2_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 specifically 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 defined 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(0:5), size = 200, replace = TRUE,
        prob = c(.6,.8,.8,.8,.8,.8)), nrow = 20, byrow = TRUE)

x <- defineData(x)

e <- evalModel(x, 2)

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

SVDEvaluation
```

```
IBEvaluation <- evalPred(e, "IBKNN", simFunct = "cos", neigh = 5, coRatedThreshold = 2)

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.

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.
topNGen	Object of class character, specifying the function used to produce the recommendations. Values: "hpr" and "mf" (currently available only for IB and UB methods).
positiveThreshold	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 an object of class evalRecResults 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 defined 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, evalRecResults-class.

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, 2)

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

SVDEvaluation
```

---

evalRecResults *Evaluation results.*

Description

Defines a structure for the results obtained by evaluating an algorithm

Slots

data: class "_ds", the dataset.

alg: class "character", the name of the used algorithm.

topN: class "numeric", the number N of Top-N items recommended to each user.

topNGen: class "character", the name of the recommendation algorithm.

positiveThreshold: class "numeric", indicating the threshold of the ratings to be considered a good. This attribute is not used when evaluating implicit feedback.

alpha: class numeric, is the half-life parameter for the rankscore metric.

parameters: class "list", parameters used in the configuration of the algorithm.

TP: class "numeric", True Positives count on each fold.

FP: class "numeric", False Positives count on each fold.

TN: class "numeric", True Negatives count on each fold.

FN: class "numeric", False Negatives count on each fold.

precision: class "numeric", precision measured on each fold.

recall: class "numeric", recall measured on each fold.

F1: class "numeric", F1 measured on each fold.

nDCG: class "numeric", nDCG measured on each fold.

rankscore: class "numeric", rankscore measured on each fold.

item_coverage: class "numeric", item coverage.

`user_coverage`: class "`numeric`", user coverage.

`ex.time`: class "`numeric`", the execution time.

`TP_count`: class "`numeric`", True positives count on each item.

`rec_counts`: class "`numeric`", counts how many times an item was recommended.

`rec_popularity`: class "`numeric`", popularity of recommendations.

Methods

`show` signature(object = "evalRecResults")

`results` signature(object = "evalRecResults", metrics = "character"): returns a subset of the results based on the required metric.

`eval_nDCG` *Normalized Discounted Cumulative Gain*

Description

Metric for information retrival where positions are discounted logarithmically.

Usage

```
eval_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.

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 defined 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(NA, 1:5), 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
```

histogram *Ratings histogram.*

Description

Histogram of the ratings grouped by value.

Usage

```
histogram(data, title = "", x = "Rating values", y = "# of ratings")
```

Arguments

data	class "_ds", the dataset.
title	class "character", eventual caption of for the chart.
x	class "character", label for the x-axis.
y	class "character", label for the y-axis.

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".

parameters: the parameters used in the model, class "list".

Methods

show signature(object = "IBclass")

See Also

rrecsys.

ml100k *Movielens 100K Dataset*

Description

MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota.

This data set consists of:

1. 100,000 ratings (1-5) from 943 users on 1682 movies.
2. Each user has rated at least 20 movies.

The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up - users who had less than 20 ratings or did not have complete demographic information were removed from this data set. Detailed descriptions of the data file can be found at the end of this file.

Source

http://grouplens.org/datasets/movielens/100k/

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)" ...

Source

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

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"`.

`parameters`: the parameters used in the model, class `"list"`.

Methods

`show` signature(object = "PPLclass")

See Also

`rrecsys`.

predict *Generate predictions.*

Description

Generate predictions on any of the previously trained models.

Arguments

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

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, learningRate = 0.01, regCoef = 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. Two main methods: recommendHPR, recommendMF. The first method recommends the highest predicted ratings on a user. Instead recommendMF (currently available only for IBKNN and UBKNN), recommends the most frequent item in the user's neighborhood.

Usage

```
recommendHPR(model, topN = 3)
recommendMF(model, topN = 3, pt)
```

Arguments

model	the trained model of any algorithm.
topN	number of items to be recommended per user, class numeric.
pt	positive threshold, 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 <- recommendHPR(r)
```

`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. Given two items a and b, we consider them as rating vectors $\vec{a}$ and $\vec{b}$. If the argument *simFunct* is set to "cos" the method computes the cosine similarity as:

$$sim(\vec{a}, \vec{b}) = cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|}$$

If the argument *simFunct* is set to "adjCos" the method determines the "adjusted cosine" distance among the items as:

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

It extracts, based on the value of the *neigh* attribute, the number of closest neighbors for each item.

6. **UBKNN**. As `alg = "UBKNN"` a k-nearest neighbor user-based collaborative filtering algorithm. Given two users *u* and *u*, we consider them as rating vectors $\vec{u}$ and $\vec{v}$. If the argument *simFunct* is set to "cos" the method computes the cosine similarity as:

$$sim(\vec{u}, \vec{v}) = cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| * |\vec{v}|}$$

If the argument *simFunct* is set to "Pearson" the method determines the "Pearson correlation" among the users as:

$$sim(\vec{u}, \vec{v}) = Pearson(\vec{u}, \vec{v}) = \frac{\sum\limits_{i \in I_u \cap I_v} (R_{ui} - \overline{R_u}) * (R_{vi} - \overline{R_v})}{\sqrt{\sum\limits_{i \in I_u \cap I_v} (R_{ui} - \overline{R_u})^2 * \sum\limits_{i \in I_u \cap I_v} (R_{vi} - \overline{R_v})^2}}$$

It extracts, based on the value of the *neigh* attribute, the number of closest neighbors for each item.

7. **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, by configuring the *steps* argument). 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:

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

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

. The attribute *learningRate* represents the learning rate, while *regCoef* corresponds to the weight of the regularization term. If the argument *biases* is TRUE, the biases will be computed to update the features and generate predictions.

8. **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:

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

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

Initially the *V* matrix is initialized with Gaussian 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.

9. **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 Boolean `randomInit` parameter determines whatever the feature matrix are initialized to a random value or at a static 0.1 value. 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.
10. **SlopeOne** The Weighted Slope One (`alg = "slopeOne"`) performs prediction for a missing rating $\hat{r}_{ui}$ for user u on item i as the following average:

$$\hat{r}_{ui} = \frac{\sum_{\forall r_{uj}} (dev_{ij} + r_{uj}) c_{ij}}{\sum_{\forall r_{uj}} c_{ij}}.$$

The average deviation rating dev_ij between co-rated items is defined by:

$$dev_{ij} = \sum_{\forall u \in users} \frac{r_{ui} - r_{uj}}{c_{ij}}.$$

Where c_ij is the number of co-ratings between items i and j and r_ui is an existing rating for user u on item i. The Weighted Slope One takes into account both, information from users who rated the same item and the number of observed ratings.

To view a full list of available algorithms and their default configuration execute `rrecsysRegistry`.

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)

r2 <- rrecsys(myratings, alg = "IBKNN", simFunct = "cos", 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 auto-convergence algorithm.
deltaErrorThreshold	class "numeric", is the threshold for the auto-convergence 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 auto-convergence 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)
```

---

slopeOneClass *Slope One model.*

Description

Container for the model learned using Slope One algorithm.

Slots

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

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

devcard: Deviation and Cardinality between columns, class "list".

Methods

show signature(object = "SVDclass")

See Also

rrecsys.

`sparseDataSet-class` *Dataset class for tuples (user, item, rating).*

Description

Container for a sparse dataset that distinguishes between binary and non-binary feedback datasets. Data are stored as tuples (user, item, rating). Extends `_ds`.

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.

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

`userID`: class `"numeric"`, array containing all user IDs.

`itemID`: class `"numeric"`, array containing all item IDs.

`userPointers`: class `"list"`, pointer to all users position in the dataset.

`itemPointers`: class `"list"`, pointer to all items position in the dataset.

Methods

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

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

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

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

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

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

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

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

rowAverages signature(object = "sparseDataSet"): returns the average rating on each row.

colAverages signature(object = "sparseDataSet"): returns the average rating on each column.

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 used in the model, class `"list"`.

`baselines`: Global, user and item baselines, class `"list"`.

Methods

`show` signature(object = "SVDclass")

See Also

`rrecsys`.

UBclass *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"`.

`parameters`: the parameters used in the model, class `"list"`.

Methods

`show` signature(object = "UBclass")

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`.

_ds-class *Dataset class.*

Description

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

Slots

`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.

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

Methods

show signature(object = "_ds")

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

summary signature(object = "_ds"): summary of the characteristics of the dataset.

Index