Package 'recommenderlab'

August 27, 2019

Version 0.2-5

Date 2019-08-27

Title Lab for Developing and Testing Recommender Algorithms

Description Provides a research infrastructure to test and develop
recommender algorithms including UBCF, IBCF, FunkSVD and association
rule-based algorithms.

Classification/ACM G.4, H.2.8

Depends R (>= 2.10.0), Matrix, arules, proxy, registry

Imports methods, utils, stats, irlba, recosystem

Suggests ROCR, testthat

BugReports `https://github.com/mhahsler/recommenderlab/issues`

URL `https://github.com/mhahsler/recommenderlab`

License GPL-2

Copyright (C) Michael Hahsler

NeedsCompilation no

Author Michael Hahsler [aut, cre, cph],
Bregt Vereet [ctb, cph]

Maintainer Michael Hahsler <`mhahsler@lyle.smu.edu`>

Repository CRAN

Date/Publication 2019-08-27 17:50:08 UTC

R topics documented:

`binaryRatingMatrix` *Class "binaryRatingMatrix": A Binary Rating Matrix*

Description

A matrix to represent binary rating data. 1 codes for a positive rating and 0 codes for either no or a negative rating. This coding is common for market basked data where products are either bought or not.

Objects from the Class

Objects can be created by calls of the form `new("binaryRatingMatrix",data = im)`, where `im` is an `itemMatrix` as defined in package **arules**, by coercion from a matrix (all non-zero values will be a 1), or by using `binarize` for an object of class "realRatingMatrix".

Slots

`data`**:** Object of class `"itemMatrix"` (see package **arules**)

Extends

Class `"ratingMatrix"`, directly.

Methods

coerce `signature(from = "matrix", to = "binaryRatingMatrix")`: The matrix needs to be a logical matrix, or a 0-1 matrix (0 means FALSE and 1 means TRUE). NAs are interpreted as FALSE.

coerce `signature(from = "itemMatrix", to = "binaryRatingMatrix")`

coerce `signature(from = "data.frame", to = "binaryRatingMatrix")`

coerce `signature(from = "binaryRatingMatrix", to = "matrix")`

coerce `signature(from = "binaryRatingMatrix", to = "dgTMatrix")`

coerce `signature(from = "binaryRatingMatrix", to = "ngCMatrix")`

coerce `signature(from = "binaryRatingMatrix", to = "dgCMatrix")`

coerce `signature(from = "binaryRatingMatrix", to = "itemMatrix")`

coerce `signature(from = "binaryRatingMatrix", to = "list")`

See Also

`itemMatrix` in **arules**, `getList`.

Examples

```
## create a 0-1 matrix
m <- matrix(sample(c(0,1), 50, replace=TRUE), nrow=5, ncol=10,
    dimnames=list(users=paste("u", 1:5, sep=''),
    items=paste("i", 1:10, sep='')))
m

## coerce it into a binaryRatingMatrix
b <- as(m, "binaryRatingMatrix")
b

## coerce it back to see if it worked
as(b, "matrix")

## use some methods defined in ratingMatrix
dim(b)
dimnames(b)

## counts
rowCounts(b) ## number of ratings per user
colCounts(b) ## number of ratings per item

## plot
image(b)

## sample and subset
sample(b,2)
b[1:2,1:5]

## coercion
```

```
as(b, "list")
head(as(b, "data.frame"))
head(getData.frame(b, ratings=FALSE))

## creation from user/item tuples
df <- data.frame(user=c(1,1,2,2,2,3), items=c(1,4,1,2,3,5))
df
b2 <- as(df, "binaryRatingMatrix")
b2
as(b2, "matrix")
```

---

calcPredictionAccuracy — *Calculate the Prediction Error for a Recommendation*

Description

Calculate prediction accuracy. For predicted ratings MAE (mean average error), MSE (means squared error) and RMSE (root means squared error) are calculated. For topNLists various binary classification metrics are returned.

Usage

```
calcPredictionAccuracy(x, data, ...)
## S4 method for signature 'realRatingMatrix,realRatingMatrix'
calcPredictionAccuracy(x, data, byUser=FALSE,...)
## S4 method for signature 'topNList,realRatingMatrix'
calcPredictionAccuracy(x, data, byUser=FALSE, given=NULL, goodRating=NA,...)
## S4 method for signature 'topNList,binaryRatingMatrix'
calcPredictionAccuracy(x, data, byUser=FALSE, given=NULL,...)
```

Arguments

`x`	Predicted items in a "topNList" or predicted ratings as a "realRatingMatrix"
`data`	Actual ratings by the user as a "RatingMatrix"
`byUser`	logical; Should the errors be averaged by user or over all recommendations?
`given`	how many items were given to create the predictions.
`goodRating`	threshold for determining what rating is a good rating. Used only if x is a topN-List and data is a "realRatingMatrix".
`...`	further arguments.

Value

Returns a vector with the average measures for `byUser=TRUE`. Otherwise, a matrix with a row for each user is returned.

References

Asela Gunawardana and Guy Shani (2009). A Survey of Accuracy Evaluation Metrics of Recommendation Tasks, Journal of Machine Learning Research 10, 2935-2962.

See Also

`topNList`, `binaryRatingMatrix`, `realRatingMatrix`.

Examples

```
### real valued recommender
data(Jester5k)

## create 90/10 split (known/unknown) for the first 500 users in Jester5k
e <- evaluationScheme(Jester5k[1:500,], method="split", train=0.9,
    k=1, given=15)
e

## create a user-based CF recommender using training data
r <- Recommender(getData(e, "train"), "UBCF")

## create predictions for the test data using known ratings (see given above)
p <- predict(r, getData(e, "known"), type="ratings")
p

## compute error metrics averaged per user and then averaged over all
## recommendations
calcPredictionAccuracy(p, getData(e, "unknown"))
head(calcPredictionAccuracy(p, getData(e, "unknown"), byUser=TRUE))

## evaluate topNLists instead (you need to specify given and goodRating!)
p <- predict(r, getData(e, "known"), type="topNList")
p
calcPredictionAccuracy(p, getData(e, "unknown"), given=15, goodRating=5)

## evaluate a binary recommender
data(MSWeb)
MSWeb10 <- sample(MSWeb[rowCounts(MSWeb) >10,], 50)

e <- evaluationScheme(MSWeb10, method="split", train=0.9,
    k=1, given=3)
e

## create a user-based CF recommender using training data
r <- Recommender(getData(e, "train"), "UBCF")

## create predictions for the test data using known ratings (see given above)
p <- predict(r, getData(e, "known"), type="topNList", n=10)
p

calcPredictionAccuracy(p, getData(e, "unknown"), given=3)
```

dissimilarity *Dissimilarity and Similarity Calculation Between Rating Data*

Description

Calculate dissimilarities/similarities between ratings by users and for items.

Usage

```
## S4 method for signature 'binaryRatingMatrix'
dissimilarity(x, y = NULL, method = NULL, args = NULL, which="users")
## S4 method for signature 'realRatingMatrix'
dissimilarity(x, y = NULL, method = NULL, args = NULL, which="users")

similarity(x, y = NULL, method = NULL, args = NULL, ...)
## S4 method for signature 'ratingMatrix'
similarity(x, y = NULL, method = NULL, args = NULL, which="users")
```

Arguments

`x`	a ratingMatrix.
`y`	`NULL` or a second ratingMatrix to calculate cross-(dis)similarities.
`method`	(dis)similarity measure to use. Available measures are typically `"cosine"`, `"pearson"`, `"jaccard"`, etc. See `dissimilarity` for class `itemMatrix` in **arules** for details about measures for `binaryRatingMatrix` and `dist` in **proxy** for `realRatingMatrix`.
`args`	a list of additional arguments for the methods.
`which`	a character string indicating if the (dis)similarity should be calculated between `"users"` (rows) or `"items"` (columns).
`...`	further arguments.

Details

Similarities are computed from dissimilarities using $s = 1/(1 + d)$ or $s = 1 - d$ depending on the measure. For Pearson we use 1 - positive correlation.

Value

returns an object of class `dist`, `simil` or an appropriate object (e.g., a matrix) to represent a cross-(dis)similarity.

See Also

`ratingMatrix` and `dissimilarity` in **arules**.

Examples

```
data(MSWeb)

## between 5 users
dissimilarity(MSWeb[1:5,], method = "jaccard")
similarity(MSWeb[1:5,], method = "jaccard")

## between first 3 items
dissimilarity(MSWeb[,1:3], method = "jaccard", which = "items")
similarity(MSWeb[,1:3], method = "jaccard", which = "items")

## cross-similarity between first 2 users and users 10-20
similarity(MSWeb[1:2,], MSWeb[10:20,], method="jaccard")
```

`Error` *Error Calculation*

Description

Calculate the mean absolute error (MAE), mean square error (MSE), root mean square error (RMSE) and for matrices also the Frobenius norm (identical to RMSE).

Usage

```
MSE(true, predicted, na.rm = TRUE)
RMSE(true, predicted, na.rm = TRUE)
MAE(true, predicted, na.rm = TRUE)
frobenius(true, predicted, na.rm = TRUE)
```

Arguments

`true`	true values.
`predicted`	predicted values
`na.rm`	ignore missing values.

Details

Frobenius norm requires matrices.

Value

The error value.

Examples

```
true <- rnorm(10)
predicted <- rnorm(10)

MAE(true, predicted)
MSE(true, predicted)
RMSE(true, predicted)

true <- matrix(rnorm(9), nrow = 3)
predicted <- matrix(rnorm(9), nrow = 3)

frobenius(true, predicted)
```

`evaluate` *Evaluate a Recommender Models*

Description

Evaluates a single or a list of recommender model given an evaluation scheme.

Usage

```
evaluate(x, method, ...)

## S4 method for signature 'evaluationScheme,character'
evaluate(x, method, type="topNList",
  n=1:10, parameter=NULL, progress = TRUE, keepModel=FALSE)
## S4 method for signature 'evaluationScheme,list'
evaluate(x, method, type="topNList",
  n=1:10, parameter=NULL, progress = TRUE, keepModel=FALSE)
```

Arguments

`x`	an evaluation scheme (class `"evaluationScheme"`).
`method`	a character string or a list. If a single character string is given it defines the recommender method used for evaluation. If several recommender methods need to be compared, `method` contains a nested list. Each element describes a recommender method and consists of a list with two elements: a character string named `"name"` containing the method and a list named `"parameters"` containing the parameters used for this recommender method. See `Recommender` for available methods.
`type`	evaluate "topNList" or "ratings"?
`n`	N (number of recommendations) of the top-N lists generated (only if type="topNList").
`parameter`	a list with parameters for the recommender algorithm (only used when `method` is a single method).
`progress`	logical; report progress?
`keepModel`	logical; store used recommender models?
`...`	further arguments.

Value

Returns an object of class `"evaluationResults"` or if `method` is a list an object of class `"evaluationResultList"`.

See Also

`evaluationScheme`, `evaluationResults`. `evaluationResultList`.

Examples

```
### evaluate top-N list recommendations on a 0-1 data set
## Note: we sample only 100 users to make the example run faster
data("MSWeb")
MSWeb10 <- sample(MSWeb[rowCounts(MSWeb) >10,], 100)

## create an evaluation scheme (10-fold cross validation, given-3 scheme)
es <- evaluationScheme(MSWeb10, method="cross-validation",
        k=10, given=3)

## run evaluation
ev <- evaluate(es, "POPULAR", n=c(1,3,5,10))
ev

## look at the results (by the length of the topNList)
avg(ev)
plot(ev, annotate = TRUE)

## evaluate several algorithms with a list
algorithms <- list(
RANDOM = list(name = "RANDOM", param = NULL),
POPULAR = list(name = "POPULAR", param = NULL)
)

evlist <- evaluate(es, algorithms, n=c(1,3,5,10))
plot(evlist, legend="topright")

## select the first results
evlist[[1]]

### Evaluate using a data set with real-valued ratings
## Note: we sample only 100 users to make the example run faster
data("Jester5k")
es <- evaluationScheme(Jester5k[1:100], method="cross-validation",
  k=10, given=10, goodRating=5)
## Note: goodRating is used to determine positive ratings

## predict top-N recommendation lists
## (results in TPR/FPR and precision/recall)
ev <- evaluate(es, "RANDOM", type="topNList", n=10)
avg(ev)

## predict missing ratings
## (results in RMSE, MSE and MAE)
```

```
ev <- evaluate(es, "RANDOM", type="ratings")
avg(ev)
```

---

evaluationResultList-class

Class "evaluationResultList": Results of the Evaluation of a Multiple Recommender Methods

---

Description

Contains the evaluation results for several runs using multiple recommender methods in form of confusion matrices. For each run the used models might be avialable.

Objects from the Class

Objects are created by `evaluate`.

Slots

`.Data`**:** Object of class `"list"`: a list of `"evaluationResults"`.

Extends

Class `"list"`, from data part.

Methods

avg `signature(x = "evaluationResultList")`: returns a list of average confusion matrices.

[`signature(x = "evaluationResultList", i = "ANY", j = "missing", drop = "missing")`

coerce `signature(from = "list", to = "evaluationResultList")`

show `signature(object = "evaluationResultList")`

See Also

`evaluate`, `evaluationResults`.

`evaluationResults-class` *Class "evaluationResults": Results of the Evaluation of a Single Recommender Method*

Description

Contains the evaluation results for several runs using the same recommender method in form of confusion matrices. For each run the used model might be avialable.

Objects from the Class

Objects are created by `evaluate`.

Slots

`results`**:** Object of class `"list"`: contains objects of class `"ConfusionMatrix"`, one for each run specified in the used evaluation scheme.

Methods

avg `signature(x = "evaluationResults")`: returns an averaged confusion matrix.

getConfusionMatrix `signature(x = "evaluationResults")`: returns a list of confusion matrices.

getModel `signature(x = "evaluationResults")`: returns a list of used recommender models (if avilable).

getRuns `signature(x = "evaluationResults")`: returns the number of runs/number of confusion matrices.

show `signature(object = "evaluationResults")`

See Also

`evaluate`

`evaluationScheme` *Creator Function for evaluationScheme*

Description

Creates an evaluationScheme object from a data set. The scheme can be a simple split into training and test data, k-fold cross-evaluation or using k independent bootstrap samples.

Usage

```
evaluationScheme(data, ...)

## S4 method for signature 'ratingMatrix'
evaluationScheme(data, method="split",
    train=0.9, k=NULL, given, goodRating = NA)
```

Arguments

`data`	data set as a ratingMatrix.
`method`	a character string defining the evaluation method to use (see details).
`train`	fraction of the data set used for training.
`k`	number of folds/times to run the evaluation (defaults to 10 for cross-validation and bootstrap and 1 for split).
`given`	single number of items given for evaluation or a vector of length of data giving the number of items given for each observation. Negative values implement all-but schemes. For example, `given = -1` means all-but-1 evaluation.
`goodRating`	numeric; threshold at which ratings are considered good for evaluation. E.g., with `goodRating=3` all items with actual user rating of greater or equal 3 are considered positives in the evaluation process. Note that this argument is only used if the ratingMatrix is a of subclass realRatingMatrix!
`...`	further arguments.

Details

`evaluationScheme` creates an evaluation scheme (training and test data) with `k` runs and one of the following methods:

`"split"` randomly assigns the proportion of objects specified by `train` to the training set and the rest is used for the test set.

`"cross-validation"` creates a k-fold cross-validation scheme. The data is randomly split into k parts and in each run k-1 parts are used for training and the remaining part is used for testing. After all k runs each part was used as the test set exactly once.

`"bootstrap"` creates the training set by taking a bootstrap sample (sampling with replacement) of size `train` times number of users in the data set. All objects not in the training set are used for testing.

For evaluation, Breese et al. (1998) introduced the four experimental protocols called Given 2, Given 5, Given 10 and All-but-1. During testing, the Given x protocol presents the algorithm with only x randomly chosen items for the test user, and the algorithm is evaluated by how well it is able to predict the withheld items. For All-but-x, the algorithm sees all but x withheld ratings for the test user. `given` controls x in the evaluations scheme. Positive integers result in a Given x protocol, while negative values produce a All-but-x protocol.

Value

Returns an object of class `"evaluationScheme"`.

References

Kohavi, Ron (1995). "A study of cross-validation and bootstrap for accuracy estimation and model selection". Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 1137-1143.

Breese JS, Heckerman D, Kadie C (1998). "Empirical Analysis of Predictive Algorithms for Collaborative Filtering." In Uncertainty in Artificial Intelligence. Proceedings of the Fourteenth Conference, pp. 43-52.

See Also

`getData`, `evaluationScheme`, `ratingMatrix`.

Examples

```
data("MSWeb")

MSWeb10 <- sample(MSWeb[rowCounts(MSWeb) >10,], 50)
MSWeb10

## simple split with 3 items given
esSplit <- evaluationScheme(MSWeb10, method="split",
        train = 0.9, k=1, given=3)
esSplit

## 4-fold cross-validation with all-but-1 items for learning.
esCross <- evaluationScheme(MSWeb10, method="cross-validation",
        k=4, given=-1)
esCross
```

`evaluationScheme-class`

Class "evaluationScheme": Evaluation Scheme

Description

An evaluation scheme created from a data set. The scheme can be a simple split into training and test data, k-fold cross-evaluation or using k bootstrap samples.

Objects from the Class

Objects can be created by `evaluationScheme(data,method="split",train=0.9,k=NULL,given=3)`.

Slots

`data`: Object of class `"ratingMatrix"`; the data set.

`given`: Object of class `"integer"`; given ratings are randomly selected for each evaluation user and presented to the recommender algorithm to calculate recommend items/ratings. The recommended items are compared to the remaining items for the evaluation user.

`goodRating`**:** Object of class `"numeric"`; Rating at which an item is considered a positive for evaluation.

`k`**:** Object of class `"integer"`; number of runs for evaluation. Default is 1 for method "split" and 10 for "cross-validation" and "bootstrap".

`knownData`**:** Object of class `"ratingMatrix"`; data set with only known (given) items.

`method`**:** Object of class `"character"`; evaluation method. Available methods are: "split", "cross-validation" and "bootstrap".

`runsTrain`**:** Object of class `"list"`; internal repesentation for the split in training and test data for the evaluation runs.

`train`**:** Object of class `"numeric"`; portion of data used for training for "split" and "bootstrap".

`unknownData`**:** Object of class `"ratingMatrix"`; data set with only unknown items.

Methods

getData `signature(x = "evaluationScheme")`: access data. Parameters are `type` ("train", "known" or "unknown") and `run` (1...k). `"train"` returns the training data for the run, `"known"` returns the known ratings used for prediction for the test data, and `"unknown"` returns the ratings used for evaluation for the test data.

show `signature(object = "evaluationScheme")`

See Also

`ratingMatrix` and the creator function `evaluationScheme`.

`funkSVD` *Funk SVD for Matrices with Missing Data*

Description

Implements matrix decomposition by the stochastic gradient descent optimization popularized by Simon Funk to minimize the error on the known values.

Usage

```
funkSVD(x, k = 10, gamma = 0.015, lambda = 0.001,
  min_improvement = 1e-06, min_epochs = 50, max_epochs = 200,
  verbose = FALSE)
```

Arguments

`x`	a matrix, potentially containing NAs.
`k`	number of features (i.e, rank of the approximation).
`gamma`	regularization term.
`lambda`	learning rate.

`min_improvement`	required minimum improvement per iteration.
`min_epochs`	minimum number of iterations per feature.
`max_epochs`	maximum number of iterations per feature.
`verbose`	show progress.

Details

Funk SVD decomposes a matrix (with missing values) into two components U and V. The singular values are folded into these matrices. The approximation for the original matrix can be obtained by $R = UV'$.

This function `predict` in this implementation folds in new data rows by estimating the u vectors using gradient descend and then calculating the reconstructed complete matrix r for these users via $r = uV'$.

Value

An object of class `"funkSVD"` with components

`U`	the U matrix.
`V`	the V matrix.
`parameters`	a list with parameter values.

Note

The code is based on the implmentation in package **rrecsys** by Ludovik Coba and Markus Zanker.

References

Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems, IEEE Computer, pp. 42-49, August 2009.

Examples

```
### this takes a while to run
## Not run:
data("Jester5k")

train <- as(Jester5k[1:100], "matrix")
fsvd <- funkSVD(train, verbose = TRUE)

### reconstruct the rating matrix as R = UV'
### and calculate the root mean square error on the known ratings
r <- tcrossprod(fsvd$U, fsvd$V)
rmse(train, r)

### fold in new users for matrix completion
test <- as(Jester5k[101:105], "matrix")
p <- predict(fsvd, test, verbose = TRUE)
rmse(test, p)
```

```
## End(Not run)
```

getList — *List and Data.frame Representation for Recommender Matrix Objects*

Description

Create a list or data.frame representation for various objects used in **recommenderlab**. These functions are used in addition to available coercion to allow for parameters like decode.

Usage

```
getList(from, ...)
## S4 method for signature 'realRatingMatrix'
getList(from, decode = TRUE, ratings = TRUE, ...)
## S4 method for signature 'binaryRatingMatrix'
getList(from, decode = TRUE, ...)
## S4 method for signature 'topNList'
getList(from, decode = TRUE, ...)

getData.frame(from, ...)
## S4 method for signature 'ratingMatrix'
getData.frame(from, decode = TRUE, ratings = TRUE, ...)
```

Arguments

`from`	object to be represented as a list.
`decode`	use item names or item IDs (column numbers) for items?
`ratings`	include ratings in the list or data.frame?
`...`	further arguments (currently unused).

Details

Lists have one vector with items (and ratings) per user. The data.frame has one row per rating with the user in the first column, the item as the second and the rating as the third.

Value

Returns a list or a data.frame.

See Also

`binaryRatingMatrix`, `realRatingMatrix`, `topNList`.

Examples

```
data(Jester5k)

getList(Jester5k[1,])
getData.frame(Jester5k[1,])
```

HybridRecommender *Create a Hybrid Recommender*

Description

Creates and combines recommendations using several recommender algorithms.

Usage

```
HybridRecommender(..., weights = NULL)
```

Arguments

`...`	objects of class 'Recommender'.
`weights`	weights for the recommenders. The recommenders are equally weighted by default.

Details

The hybrid recommender is initialized with a set of Recommender objects trained on the same training set (at least the training sets need to have the same items in the same order).

For creating recommendations (`predict`), each recommender algorithm is used to create ratings. The individual ratings are combined using weighted sum. Weights can be specified in `weights`.

Value

An object of class 'Recommender'.

See Also

`Recommender`

Examples

```
data("MovieLense")
MovieLense100 <- MovieLense[rowCounts(MovieLense) >100,]
train <- MovieLense100[1:100]
test <- MovieLense100[101:103]

## mix popular movies with a random recommendations for diversity and
## rerecommend some movies the user liked.
recom <- HybridRecommender(
  Recommender(train, method = "POPULAR"),
  Recommender(train, method = "RANDOM"),
  Recommender(train, method = "RERECOMMEND"),
  weights = c(.6, .1, .3)
  )

recom

getModel(recom)

as(predict(recom, test), "list")
```

---

internalFunctions *Internal Utility Functions*

Description

Utility functions used internally by recommender algorithms. See files starting with RECOM in the package's R directory for examples of usage.

Usage

```
returnRatings(ratings, newdata,
  type = c("topNList", "ratings", "ratingMatrix"),
  n, randomize = NULL, minRating = NA)

getParameters(defaults, parameter)
```

Arguments

ratings	a realRatingMatrix.
newdata	a realRatingMatrix.
type	type of recommendation to return.
n	max. number of entries in the top-N list.
randomize	randomization factor for producing the top-N list.
minRating	do not include ratings less than this.
defaults	list with parameters and default values.
parameter	list with actual parameters.

Details

returnRatings is used in the predict function of recommender algorithms to return different types of recommendations.

getParameters is a helper function which checks parameters for consistency and provides default values. Used in the Recommender constructor.

`Jester5k` *Jester dataset (5k sample)*

Description

The data set contains a sample of 5000 users from the anonymous ratings data from the Jester Online Joke Recommender System collected between April 1999 and May 2003.

Usage

```
data(Jester5k)
```

Format

The format of `Jester5k` is: Formal class 'realRatingMatrix' [package "recommenderlab"]

The format of `JesterJokes` is: vector of character strings.

Details

`Jester5k` contains a 5000 x 100 rating matrix (5000 users and 100 jokes) with ratings between -10.00 and +10.00. All selected users have rated 36 or more jokes.

The data also contains the actual jokes in `JesterJokes`.

References

Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. "Eigentaste: A Constant Time Collaborative Filtering Algorithm." Information Retrieval, 4(2), 133-151. July 2001.

Examples

```
data(Jester5k)
Jester5k

## number of ratings
nratings(Jester5k)

## number of ratings per user
summary(rowCounts(Jester5k))

## rating distribution
hist(getRatings(Jester5k), main="Distribution of ratings")

## 'best' joke with highest average rating
best <- which.max(colMeans(Jester5k))
cat(JesterJokes[best])
```

`MovieLense` *MovieLense Dataset (100k)*

Description

The 100k MovieLense ratings data set. 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. The data set contains about 100,000 ratings (1-5) from 943 users on 1664 movies. Movie metadata is also provided in `MovieLenseMeta`.

Usage

```
data(MovieLense)
```

Format

The format of `MovieLense` is an object of class "`realRatingMatrix`"

The format of `MovieLenseMeta` is a data.frame with movie title, year, IMDb URL and indicator variables for 19 genres.

Source

GroupLens Research, https://grouplens.org/datasets/movielens/

References

Herlocker, J., Konstan, J., Borchers, A., Riedl, J.. An Algorithmic Framework for Performing Collaborative Filtering. Proceedings of the 1999 Conference on Research and Development in Information Retrieval. Aug. 1999.

Examples

```
data(MovieLense)
MovieLense

## look at the first few ratings of the first user
head(as(MovieLense[1,], "list")[[1]])


## visualize part of the matrix
image(MovieLense[1:100,1:100])

## number of ratings per user
hist(rowCounts(MovieLense))

## number of ratings per movie
hist(colCounts(MovieLense))

## mean rating (averaged over users)
```

```
mean(rowMeans(MovieLense))

## available movie meta information
head(MovieLenseMeta)
```

`MSWeb` *Anonymous web data from www.microsoft.com*

Description

Vroots visited by users in a one week timeframe.

Usage

```
data(MSWeb)
```

Format

The format is: Formal class `"binaryRatingMatrix"`.

Details

The data was created by sampling and processing the www.microsoft.com logs. The data records the use of www.microsoft.com by 38000 anonymous, randomly-selected users. For each user, the data lists all the areas of the web site (Vroots) that user visited in a one week timeframe in February 1998.

This dataset contains 32710 valid users and 285 Vroots.

Source

Asuncion, A., Newman, D.J. (2007). UCI Machine Learning Repository, Irvine, CA: University of California, School of Information and Computer Science. `http://www.ics.uci.edu/~mlearn/MLRepository.html`

References

J. Breese, D. Heckerman., C. Kadie (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering, Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI.

Examples

```
data(MSWeb)
MSWeb

nratings(MSWeb)

## look at first two users
as(MSWeb[1:2,], "list")
```

```
## items per user
hist(rowCounts(MSWeb), main="Distribution of Vroots visited per user")
```

normalize *Normalize the ratings*

Description

Provides the generic for normalize/denormalize and a method to normalize/denormalize the ratings in a realRatingMatrix.

Usage

```
normalize(x, ...)
## S4 method for signature 'realRatingMatrix'
normalize(x, method="center", row=TRUE)

denormalize(x, ...)
## S4 method for signature 'realRatingMatrix'
denormalize(x, method=NULL, row=NULL,
    factors=NULL)
```

Arguments

`x`	a realRatingMatrix.
`method`	normalization method. Currently "center" or "Z-score".
`row`	logical; normalize rows (or the columns)?
`factors`	a list with the factors to be used for denormalizing (elements are "mean" and "sds"). Usually these are not specified and the values stored in `x` are used.
`...`	further arguments (currently unused).

Details

Normalization tries to reduce the individual rating bias by row centering the data, i.e., by subtracting from each available rating the mean of the ratings of that user (row). Z-score in addition divides by the standard deviation of the row/column. Normalization can also be done on columns.

Denormalization reverses normalization. It uses the normalization information stored in x unless the user specifies method, row and factors.

Value

A normalized realRatingMatrix.

Examples

```
## create a matrix with ratings
m <- matrix(sample(c(NA,0:5),50, replace=TRUE, prob=c(.5,rep(.5/6,6))),
nrow=5, ncol=10, dimnames = list(users=paste('u', 1:5, sep=''),
items=paste('i', 1:10, sep='')))

## do normalization
r <- as(m, "realRatingMatrix")
r_n1 <- normalize(r)
r_n2 <- normalize(r, method="Z-score")

r
r_n1
r_n2

## show normalized data
image(r, main="Raw Data")
image(r_n1, main="Centered")
image(r_n2, main="Z-Score Normalization")
```

---

plot — *Plot Evaluation Results*

Description

Creates precision-recall or ROC plots for recommender evaluation results.

Usage

```
## S4 method for signature 'evaluationResults'
plot(x, y,
        avg = TRUE, add=FALSE, type= "b", annotate = FALSE, ...)
## S4 method for signature 'evaluationResultList'
plot(x, y,
        xlim=NULL, ylim=NULL, col = NULL, pch = NULL, lty = 1,
        avg = TRUE, type = "b", annotate= 0, legend="bottomright", ...)
```

Arguments

`x`	the object to be plotted.
`y`	a character string indicating the type of plot (e.g., "ROC" or "prec/rec").
`avg`	plot average of runs?
`add`	add to a plot?
`type`	line type (see `plot`).
`annotate`	annotate N (recommendation list size) to plot.
`xlim,ylim`	plot limits (see `plot`).

`col`	colors (see `plot`).
`pch`	point symbol to use (see `plot`).
`lty`	line type (see `plot`)
`legend`	where to place legend (see `legend`).
`...`	further arguments passed on to `plot`.

See Also

`evaluationResults`, `evaluationResultList`. See `evaluate` for examples.

`predict` *Predict Recommendations*

Description

Creates recommendations using a recommender model and data about new users.

Usage

```
## S4 method for signature 'Recommender'
predict(object, newdata, n = 10, data=NULL,
    type="topNList", ...)
```

Arguments

`object`	a recommender model (class `"Recommender"`).
`newdata`	data for active users (class `"ratingMatrix"`) or the index of users in the training data to create recommendations for. If an index is used then some recommender algorithms need to be passed the training data as argument `data`. Some algorithms may only support user indices.
`n`	number of recommendations in the top-N list.
`data`	training data needed by some recommender algorithms if `newdata` is a user index and not user data.
`type`	type of recommendation. The default type is `"topNList"` which creates a top-N recommendation list with recommendations. Some recommenders can also predict ratings with type `"ratings"` which returns only predicted ratings with known ratings represented by `NA`, or type `"ratingMatrix"` which returns a completed rating matrix (Note that the predicted ratings may differ from the known ratings).
`...`	further arguments.

Value

Returns an object of class `"topNList"` or of other appropriate classes.

See Also

`Recommender`, `ratingMatrix`.

Examples

```
data("MovieLense")
MovieLense100 <- MovieLense[rowCounts(MovieLense) >100,]
train <- MovieLense100[1:50]

rec <- Recommender(train, method = "POPULAR")
rec

## create top-N recommendations for new users
pre <- predict(rec, MovieLense100[101:102], n = 10)
pre
as(pre, "list")

## predict ratings for new users
pre <- predict(rec, MovieLense100[101:102], type="ratings")
pre
as(pre, "matrix")[,1:10]


## create recommendations using user ids with ids 1..10 in the
## training data
pre <- predict(rec, 1:10 , data = train, n = 10)
pre
as(pre, "list")
```

`ratingMatrix` *Class "ratingMatrix": Virtual Class for Rating Data*

Description

Defines a common class for rating data.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

[`signature(x = "ratingMatrix", i = "ANY", j = "ANY", drop = "ANY")`: subset the rating matrix (drop is ignorred).

coerce `signature(from = "ratingMatrix", to = "list")`

coerce `signature(from = "ratingMatrix", to = "data.frame")`: a data.frame with three columns. Col 1 contains user ids, col 2 contains item ids and col 3 contains ratings.

colCounts `signature(x = "ratingMatrix")`: number of ratings per column.

rowCounts `signature(x = "ratingMatrix")`: number of ratings per row.

colMeans `signature(x = "ratingMatrix")`: column-wise rating means.

rowMeans `signature(x = "ratingMatrix")`: row-wise rating means.

dim `signature(x = "ratingMatrix")`: dimensions of the rating matrix.

dimnames<- `signature(x = "ratingMatrix", value = "list")`: replace dimnames.

dimnames `signature(x = "ratingMatrix")`: retrieve dimnames.

getNormalize `signature(x = "ratingMatrix")`: returns a list with normalization information for the matrix (NULL if data is not normalized).

getRatings `signature(x = "ratingMatrix")`: returns all ratings in `x` as a numeric vector.

getRatingMatrix `signature(x = "ratingMatrix")`: returns the ratings as a sparse matrix. The format is different for binary and real rating matices.

image `signature(x = "ratingMatrix")`: plot the matrix.

nratings `signature(x = "ratingMatrix")`: number of ratings in the matrix.

sample `signature(x = "ratingMatrix")`: sample from users (rows).

show `signature(object = "ratingMatrix")`

See Also

See implementing classes `realRatingMatrix` and `binaryRatingMatrix`. See `getList`, `getData.frame`, `similarity`, `dissimilarity` and `dissimilarity`.

`realRatingMatrix` *Class "realRatingMatrix": Real-valued Rating Matrix*

Description

A matrix containing ratings (typically 1-5 stars, etc.).

Objects from the Class

Objects can be created by calls of the form `new("realRatingMatrix", data = m)`, where `m` is sparse matrix of class

`dgCMatrix` in package **Matrix** or by coercion from a regular matrix, a data.frame containing user/item/rating triplets as rows, or a sparse matrix in triplet form (`dgTMatrix` in package **Matrix**).

Slots

`data`**:** Object of class

`"dgCMatrix"`, a sparse matrix defined in package **Matrix**. Note that this matrix drops NAs instead of zeroes. Operations on `"dgCMatrix"` potentially will delete zeroes.

`normalize`**:** `NULL` or a list with normalizaton factors.

Extends

Class "ratingMatrix", directly.

Methods

coerce `signature(from = "matrix", to = "realRatingMatrix")`: Note that unknown ratings have to be encoded in the matrix as NA and not as 0 (which would mean an actual rating of 0).

coerce `signature(from = "realRatingMatrix", to = "matrix")`

coerce `signature(from = "data.frame", to = "realRatingMatrix")`: coercion from a data.frame with three columns. Col 1 contains user ids, col 2 contains item ids and col 3 contains ratings.

coerce `signature(from = "realRatingMatrix", to = "data.frame")`: produces user/item/rating triplets.

coerce `signature(from = "realRatingMatrix", to = "dgTMatrix")`

coerce `signature(from = "dgTMatrix", to = "realRatingMatrix")`

coerce `signature(from = "realRatingMatrix", to = "dgCMatrix")`

coerce `signature(from = "dgCMatrix", to = "realRatingMatrix")`

coerce `signature(from = "realRatingMatrix", to = "ngCMatrix")`

binarize `signature(x = "realRatingMatrix")`: create a "binaryRatingMatrix" by setting all ratings larger or equal to the argument `minRating` as 1 and all others to 0.

removeKnownRatings `signature(x = "realRatingMatrix")`: removes all ratings in `x` for which ratings are available in the realRatingMatrix (of same dimensions as `x`) passed as the argument `known`.

rowSds `signature(x = "realRatingMatrix")`: calculate the standard deviation of ratings for rows (users).

colSds `signature(x = "realRatingMatrix")`: calculate the standard deviation of ratings for columns (items).

See Also

See ratingMatrix inherited methods,

binaryRatingMatrix, topNList, getList and getData.frame. Also see dgCMatrix, dgTMatrix and ngCMatrix in **Matrix**.

Examples

```
## create a matrix with ratings
m <- matrix(sample(c(NA,0:5),100, replace=TRUE, prob=c(.7,rep(.3/6,6))),
nrow=10, ncol=10, dimnames = list(
    user=paste('u', 1:10, sep=''),
    item=paste('i', 1:10, sep='')
    ))
m

## coerce into a realRatingMAtrix
r <- as(m, "realRatingMatrix")
```

```
r

## get some information
dimnames(r)
rowCounts(r) ## number of ratings per user
colCounts(r) ## number of ratings per item
colMeans(r) ## average item rating


## histogram of ratings
hist(getRatings(r), breaks="FD")

## inspect a subset
image(r[1:5,1:5])

## coerce it back to see if it worked
as(r, "matrix")

## coerce to data.frame (user/item/rating triplets)
as(r, "data.frame")

## binarize into a binaryRatingMatrix with all 4+ rating a 1
b <- binarize(r, minRating=4)
b
as(b, "matrix")
```

`Recommender` *Create a Recommender Model*

Description

Learns a recommender model from given data.

Usage

```
Recommender(data, ...)
## S4 method for signature 'ratingMatrix'
Recommender(data, method, parameter=NULL)
```

Arguments

`data`	training data.
`method`	a character string defining the recommender method to use (see details).
`parameter`	parameters for the recommender algorithm.
`...`	further arguments.

Details

Recommender uses the registry mechanism from package **registry** to manage methods. This let's the user easily specify and add new methods. The registry is called `recommenderRegistry`. See examples section.

Value

An object of class 'Recommender'.

See Also

`Recommender`, `ratingMatrix`, `predict`.

Examples

```
data("MSWeb")
MSWeb10 <- sample(MSWeb[rowCounts(MSWeb) >10,], 100)

rec <- Recommender(MSWeb10, method = "POPULAR")
rec

getModel(rec)

## save and read a recommender model
saveRDS(rec, file = "rec.rds")
rec2 <- readRDS("rec.rds")
rec2
unlink("rec.rds")

## look at registry and a few methods
recommenderRegistry$get_entry_names()

recommenderRegistry$get_entry("POPULAR", dataType = "binaryRatingMatrix")

recommenderRegistry$get_entry("SVD", dataType = "realRatingMatrix")
```

`Recommender-class` *Class "Recommender": A Recommender Model*

Description

Represents a recommender model learned for a given data set (a rating matrix).

Objects from the Class

Objects are created by the creator function `Recommender(data,method,parameter = NULL)`

Slots

`method`**:** Object of class `"character"`; used recommendation method.

`dataType`**:** Object of class `"character"`; concrete class of the input data.

`ntrain`**:** Object of class `"integer"`; size of training set.

`model`**:** Object of class `"list"`; the model.

`predict`**:** Object of class `"function"`; code to compute a recommendation using the model.

Methods

getModel `signature(x = "Recommender")`: retrieve the model.

predict `signature(object = "Recommender")`: create recommendations for new data (argument `newdata`).

show `signature(object = "Recommender")`

See Also

See `Recommender` for the constructor function and a description of availble methods.

`sparseNAMatrix-class` *Sparse Matrix Representation With NAs Not Explicitly Stored*

Description

Coerce from and to a sparse matrix representation where `NA`s are not explicitly stored.

Usage

```
dropNA(x)
dropNA2matrix(x)
```

Arguments

`x`	a matrix (for `dropNA()`). A sparse matrix (for `dropNA2matrix()`)

Details

The representation is based on the sparse dgCMatrix in **Matrix** but instead of zeros, `NA`s are dropped.

Note: Be careful when working with matrix operations (multiplication, addition, etc.) since these will use the dgCMatrix superclass which assumes that all dropped values are zero and not NA! This means that the operations might remove zeros or add to NAs incorrectly.

Value

Returns a dgCMatrix or a matrix, respectively.

See Also

`dgCMatrix` in **Matrix**.

Examples

```
m <- matrix(sample(c(NA,0:5),50, replace=TRUE, prob=c(.5,rep(.5/6,6))),
    nrow=5, ncol=10, dimnames = list(users=paste('u', 1:5, sep=''),
    items=paste('i', 1:10, sep='')))
m

## drop all NAs in the representation
sparse <- dropNA(m)
sparse

## convert back to matrix
dropNA2matrix(sparse)
```

`topNList` *Class "topNList": Top-N List*

Description

Recommendations a Top-N list.

Objects from the Class

Objects can be created by `predict` with a recommender model and new data. Alternatively, objects can be created from a realRatingMatrix using `getTopNLists` (see below).

Slots

`ratings`**:** Object of class `"list"`. Each element in the list represents a top-N recommendation (an integer vector) with item IDs (column numbers in the rating matrix). The items are ordered in each vector.

`items`**:** Object of class `"list"` or `NULL`. If available, a list of the same structure as `items` with the ratings.

`itemLabels`**:** Object of class `"character"`

`n`**:** Object of class `"integer"` specifying the number of items in each recommendation. Note that the actual number on recommended items can be less depending on the data and the used algorithm.

Methods

coerce `signature(from = "topNList", to = "dgTMatrix")`

coerce `signature(from = "topNList", to = "dgCMatrix")`

coerce `signature(from = "topNList", to = "ngCMatrix")`

coerce `signature(from = "topNList", to = "matrix")`: returns a matrix with the ratings for the top-N items. All other items have a rating of NA.

coerce `signature(from = "topNList", to = "list")`: returns a list with the items in the topN-List.

bestN `signature(x = "topNList")`: returns only the best `n` recommendations (second argument is `n` which defaults to 10). The additional argument `minRating` can be used to remove all entries with a rating below this value.

getTopNLists `signature(x = "realRatingMatrix")`: create top-N lists from the ratings in `x`. Arguments are `n` (defaults to 10), `randomize` (default is `NULL`) and `minRating` (default is `NA`). Items with a rating below `minRating` will not be part of the top-N list. `randomize` can be used to get diversity in the predictions by randomly selecting items with a bias to higher rated items. The bias is introduced by choosing the items with a probability proportional to the rating $(r - min(r) + 1)^{randomize}$. The larger the value the more likely it is to get very highly rated items and a negative value for `randomize` will select low-rated items.

getRatings `signature(x = "topNList")`: get the ratings associated with the items recommended in the top-N list.

length `signature(x = "topNList")`: for how many users does this object contain a top-N list?

removeKnownItems `signature(x = "topNList")`: remove items from the top-N list which are known (have a rating) for the user given as a ratingMatrix passed on as argument `known`.

colCounts `signature(x = "topNList")`: in how many top-N does each item occur?

rowCounts `signature(x = "topNList")`: number of recommendations per user.

show `signature(object = "topNList")`

See Also

`evaluate`, `getList`, `realRatingMatrix`

Index