

Package ‘arules’

April 17, 2009

Version 1.0-0

Date 2009-03-23

Title Mining Association Rules and Frequent Itemsets

Author Michael Hahsler, Christian Buchta, Bettina Gruen and Kurt Hornik

Maintainer Michael Hahsler <michael@hahsler.net>

Description Provides the infrastructure for representing, manipulating and analyzing transaction data and patterns (frequent itemsets and association rules). Also provides interfaces to C implementations of the association mining algorithms Apriori and Eclat by C. Borgelt.

Classification/ACM G.4, H.2.8, I.5.1

URL <http://R-Forge.R-project.org/projects/arules/>

Depends R (>= 2.7.0), stats, methods, Matrix (>= 0.999375-9)

Imports stats, Matrix

Suggests pmml

License GPL-2

Copyright The code for apriori and eclat in src/rapriori.c was obtained from <http://fuzzy.cs.uni-magdeburg.de/~borgelt/> and is Copyright (C) 1996-2003 Christian Borgelt. All other code is Copyright (C) Michael Hahsler, Christian Buchta, Bettina Gruen and Kurt Hornik.

LazyLoad yes

Repository CRAN

Date/Publication 2009-03-24 19:34:30

R topics documented:

Adult	3
affinity	5
APpearance-class	6
apriori	7
AScontrol-classes	8
ASparameter-classes	10
associations-class	12
combine	13
coverage	14
crossTable	15
dissimilarity	15
duplicated	17
eclat	18
Epub	20
Groceries	20
image	21
Income	22
inspect	24
interestMeasure	24
is.closed	28
is.maximal	28
is.superset	29
itemCoding	30
itemFrequency	32
itemFrequencyPlot	33
itemMatrix-class	34
itemsets-class	37
length	38
LIST	39
match	40
predict	41
proximity-classes	42
random.transactions	43
read.transactions	45
ruleInduction	46
rules-class	48
sample	49
sets	50
size	51
sort	52
subset	53
support	54
tidLists-class	56
transactions-class	58
unique	60
WRITE	61

<i>Adult</i>	3
[-methods]	62
Index	64

Adult *Adult Data Set*

Description

The `AdultUCI` data set contains the questionnaire data of the “Adult” database (originally called the “Census Income” Database) formatted as a `data.frame`. The `Adult` data set contains the data already prepared and coerced to `transactions` for use with **arules**.

Usage

```
data("Adult")
data("AdultUCI")
```

Format

The `AdultUCI` data set contains a data frame with 48842 observations on the following 15 variables.

- age** a numeric vector.
- workclass** a factor with levels `Federal-gov`, `Local-gov`, `Never-worked`, `Private`, `Self-emp-inc`, `Self-emp-not-inc`, `State-gov`, and `Without-pay`.
- education** an ordered factor with levels `Preschool` < `1st-4th` < `5th-6th` < `7th-8th` < `9th` < `10th` < `11th` < `12th` < `HS-grad` < `Prof-school` < `Assoc-acdm` < `Assoc-voc` < `Some-college` < `Bachelors` < `Masters` < `Doctorate`.
- education-num** a numeric vector.
- marital-status** a factor with levels `Divorced`, `Married-AF-spouse`, `Married-civ-spouse`, `Married-spouse-absent`, `Never-married`, `Separated`, and `Widowed`.
- occupation** a factor with levels `Adm-clerical`, `Armed-Forces`, `Craft-repair`, `Exec-managerial`, `Farming-fishing`, `Handlers-cleaners`, `Machine-op-inspct`, `Other-service`, `Priv-house-serv`, `Prof-specialty`, `Protective-serv`, `Sales`, `Tech-support`, and `Transport-moving`.
- relationship** a factor with levels `Husband`, `Not-in-family`, `Other-relative`, `Own-child`, `Unmarried`, and `Wife`.
- race** a factor with levels `Amer-Indian-Eskimo`, `Asian-Pac-Islander`, `Black`, `Other`, and `White`.
- sex** a factor with levels `Female` and `Male`.
- capital-gain** a numeric vector.
- capital-loss** a numeric vector.
- fnlwgt** a numeric vector.
- hours-per-week** a numeric vector.

native-country a factor with levels Cambodia, Canada, China, Columbia, Cuba, Dominican-Republic, Ecuador, El-Salvador, England, France, Germany, Greece, Guatemala, Haiti, Holand-Netherlands, Honduras, Hong, Hungary, India, Iran, Ireland, Italy, Jamaica, Japan, Laos, Mexico, Nicaragua, Outlying-US (Guam-USVI-etc), Peru, Philippines, Poland, Portugal, Puerto-Rico, Scotland, South, Taiwan, Thailand, Trinidad&Tobago, United-States, Vietnam, and Yugoslavia.

income an ordered factor with levels small < large.

Details

The “Adult” database was extracted from the census bureau database found at <http://www.census.gov/ftp/pub/DES/www/welcome.html> in 1994 by Ronny Kohavi and Barry Becker, Data Mining and Visualization, Silicon Graphics. It was originally used to predict whether income exceeds USD 50K/yr based on census data. We added the attribute income with levels small and large (>50K).

We prepared the data set for association mining as shown in the section Examples. We removed the continuous attribute fnlwgt (final weight). We also eliminated education-num because it is just a numeric representation of the attribute education. The other 4 continuous attributes we mapped to ordinal attributes as follows:

age cut into levels Young (0-25), Middle-aged (26-45), Senior (46-65) and Old (66+).

hours-per-week cut into levels Part-time (0-25), Full-time (25-40), Over-time (40-60) and Too-much (60+).

capital-gain and capital-loss each cut into levels None (0), Low (0 < median of the values greater zero < max) and High (>=max).

Source

<http://www.ics.uci.edu/~mlearn/MLRepository.html>

References

A. Asuncion & D. J. Newman (2007): UCI Repository of Machine Learning Databases. Irvine, CA: University of California, Department of Information and Computer Science.

The data set was first cited in Kohavi, R. (1996): Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*.

Examples

```
data("AdultUCI")
dim(AdultUCI)
AdultUCI[1:2,]

## remove attributes
AdultUCI[["fnlwgt"]] <- NULL
AdultUCI[["education-num"]] <- NULL

## map metric attributes
```

```

AdultUCI[[ "age"]] <- ordered(cut(AdultUCI[[ "age"]], c(15,25,45,65,100)),
  labels = c("Young", "Middle-aged", "Senior", "Old"))

AdultUCI[[ "hours-per-week"]] <- ordered(cut(AdultUCI[[ "hours-per-week"]],
  c(0,25,40,60,168)),
  labels = c("Part-time", "Full-time", "Over-time", "Workaholic"))

AdultUCI[[ "capital-gain"]] <- ordered(cut(AdultUCI[[ "capital-gain"]],
  c(-Inf,0,median(AdultUCI[[ "capital-gain"]][AdultUCI[[ "capital-gain"]]>0]),
  Inf)), labels = c("None", "Low", "High"))

AdultUCI[[ "capital-loss"]] <- ordered(cut(AdultUCI[[ "capital-loss"]],
  c(-Inf,0,median(AdultUCI[[ "capital-loss"]][AdultUCI[[ "capital-loss"]]>0]),
  Inf)), labels = c("None", "Low", "High"))

## create transactions
Adult <- as(AdultUCI, "transactions")
Adult

```

affinity

Computing Affinity Between Items

Description

Provides the generic function `affinity` and the S4 methods to compute and return a similarity matrix with the affinities between items for a set of `transactions`.

Usage

```
affinity(x)
```

Arguments

`x` a matrix or an object of class `itemMatrix` or `transactions`.

Details

Affinity between the two items i and j is defined by Aggarwal et al. (2002) as

$$A(i, j) = \frac{\text{sup}(\{i, j\})}{\text{sup}(\{i\}) + \text{sup}(\{j\}) - \text{sup}(\{i, j\})},$$

where $\text{sup}(\cdot)$ is the support measure. This means that affinity is the *Jaccard similarity* between items.

Value

returns an object of class `ar_similarity` which represents the affinities between items in `x`.

References

Charu C. Aggarwal, Cecilia Procopiuc, and Philip S. Yu (2002) Finding localized associations in market basket data, *IEEE Trans. on Knowledge and Data Engineering*, 14(1):51–62.

See Also

[dissimilarity](#), [ar_similarity-class](#), [itemMatrix-class](#)

Examples

```
data("Adult")

## choose a sample, calculate affinities
s <- sample(Adult, 500)
s

a <- affinity(s)
summary(as.vector(a))
```

APappearance-class *Class “APappearance” — Specifying the ‘appearance’ Argument of apriori()*

Description

Specifies the restrictions on the associations mined by [apriori](#). Note that appearance is not supported by the implementation of [eclat](#).

Objects from the Class

If appearance restrictions are used, an appearance object will be created automatically within the [apriori](#) function using the information in the named list of the function’s `appearance` argument. In this case, the item labels used in the list will be automatically matched against the items in the used transaction database. The list can contain the following elements:

default: one of "both", "lhs", "rhs", "none" (the default is "both"). This element specified the default appearance for all items not explicitly mentioned in the other elements of the list.

lhs, rhs, both, none, items: character vectors giving the labels of the items which may only appear in the corresponding place of the rules/itemsets.

Objects can also be created by calls of the form `new("APappearance", ...)`. In this case, item IDs (column numbers of the transactions incidence matrix) have to be used instead of labels.

Slots

- set:** an integer scalar indicating how many items are specified for each of lhs, rhs, items, both and none
- items:** an integer vector of item IDs (column numbers)
- labels:** a character vector of item labels
- default:** a character scalar indicating the value for default appearance

References

Christian Borgelt (2004) *Apriori — Finding Association Rules/Hyperedges with the Apriori Algorithm*. <http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori.html>

See Also

[apriori](#)

Examples

```
data("Adult")
## Mine only rules with small or large income in the right-hand-side.
rules <- apriori(Adult, parameter = list(confidence = 0.5),
  appearance = list(rhs = c("income=small", "income=large"),
  default="lhs"))
```

apriori

Mining Associations with Apriori

Description

Mine frequent itemsets, association rules or association hyperedges using the Apriori algorithm. The Apriori algorithm employs level-wise search for frequent itemsets. The implementation of Apriori used includes some improvements (e.g., a prefix tree and item sorting).

Usage

```
apriori(data, parameter = NULL, appearance = NULL, control = NULL)
```

Arguments

- | | |
|------------|--|
| data | object of class transactions or any data structure which can be coerced into transactions (e.g., a binary matrix or data.frame). |
| parameter | object of class APparameter or named list. The default behavior is to mine rules with support 0.1, confidence 0.8, and maxlen 5. |
| appearance | object of class APappearance or named list. With this argument item appearance can be restricted. By default all items can appear unrestricted. |
| control | object of class APcontrol or named list. Controls the performance of the mining algorithm (item sorting, etc.) |

Details

Calls the C implementation of the Apriori algorithm by Christian Borgelt for mining frequent itemsets, rules or hyperedges.

Value

Returns an object of class `rules` or `itemsets`.

References

R. Agrawal, T. Imielinski, and A. Swami (1993) Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington D.C.

Christian Borgelt and Rudolf Kruse (2002) Induction of Association Rules: Apriori Implementation. *15th Conference on Computational Statistics (COMPSTAT 2002, Berlin, Germany)* Physica Verlag, Heidelberg, Germany.

Christian Borgelt (2003) Efficient Implementations of Apriori and Eclat. *Workshop of Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL, USA)*.

See Also

`APparameter-class`, `APcontrol-class`, `APappearance-class`, `transactions-class`, `itemsets-class`, `rules-class`

Examples

```
data("Adult")
## Mine association rules.
rules <- apriori(Adult,
                 parameter = list(supp = 0.5, conf = 0.9,
                                 target = "rules"))
summary(rules)
```

AScontrol-classes *Classes “AScontrol”, “APcontrol”, “ECcontrol” — Specifying the ‘control’ Argument of apriori() and eclat()*

Description

The `AScontrol` class holds the algorithmic parameters for the used mining algorithms. `APcontrol` and `ECcontrol` directly extend `AScontrol` with additional slots for parameters only suitable for the algorithms `Apriori` (`APcontrol`) and `Eclat` (`ECcontrol`).

Objects from the Class

A suitable default control object will be automatically created by the `apriori` or the `eclat` function. By specifying a named list (names equal to slots) as `control` argument for the `apriori` or the `eclat` function, default values can be replaced by the values in the list. Objects can also be created by calls of the form `new("APcontrol", ...)` or `new("ECcontrol", ...)`.

Slots

Common slots defined in `AScontrol`:

sort: an integer scalar indicating how to sort items with respect to their frequency: (default: 2)

- 1:** ascending
- 1:** descending
- 0:** do not sort
- 2:** ascending
- 2:** descending with respect to transaction size sum

verbose: a logical indicating whether to report progress

Additional slots for Apriori in `APcontrol`:

filter: a numeric scalar indicating how to filter unused items from transactions (default: 0.1)

- = 0: do not filter items with respect to. usage in sets
- < 0: fraction of removed items for filtering
- > 0: take execution times ratio into account

tree: a logical indicating whether to organize transactions as a prefix tree (default: TRUE)

heap: a logical indicating whether to use heapsort instead of quicksort to sort the transactions (default: TRUE)

memopt: a logical indicating whether to minimize memory usage instead of maximize speed (default: FALSE)

load: a logical indicating whether to load transactions into memory (default: TRUE)

Additional slots for Eclat in `ECcontrol`:

sparse: a numeric value for the threshold for sparse representation (default: 7)

Methods

coerce signature(from = "NULL", to = "APcontrol")

coerce signature(from = "list", to = "APcontrol")

coerce signature(from = "NULL", to = "ECcontrol")

coerce signature(from = "list", to = "ECcontrol")

References

Christian Borgelt (2004) *Apriori — Finding Association Rules/Hyperedges with the Apriori Algorithm*. <http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori.html>

See Also

[apriori](#), [eclat](#)

ASparameter-classes

Classes “ASparameter”, “APparameter”, “ECparameter” — Specifying the ‘parameter’ Argument of [apriori\(\)](#) and [eclat\(\)](#)

Description

The `ASparameter` class holds the mining parameters (e.g., minimum support) for the used mining algorithms. `APparameter` and `ECparameter` directly extend `ASparameter` with additional slots for parameters only suitable for the Apriori (`APparameter`) or the Eclat algorithms (`ECparameter`).

Objects from the Class

A suitable default parameter object will be automatically created by the [apriori](#) or the [eclat](#) function. By specifying a named list (names equal to slots) as `parameter` argument for the [apriori](#) or the [eclat](#) function, default values can be replaced by the values in the list. Objects can be created by calls of the form `new ("APparameter", ...)` or `new ("ECparameter", ...)`.

Slots

Common slots defined in `ASparameter`:

support: a numeric value for the minimal support of an item set (default: 0.1)

minlen: an integer value for the minimal number of items per item set (default: 1)

maxlen: an integer value for the maximal number of items per item set (default: 5)

target: a character string indicating the type of association mined. One of

- "frequent itemsets"
- "maximally frequent itemsets"
- "closed frequent itemsets"
- "rules" (only available for Apriori)
- "hyperedgesets" (only available for Apriori; see references for the definition of association hyperedgesets)

ext: a logical indicating whether to produce extended information on quality measures (e.g., `lhs.support`) (default: `FALSE`)

Additional slots for Apriori in `APparameter`:

confidence: a numeric value for the minimal confidence of rules/association hyperedges (default: 0.8)

smax: a numeric value for the maximal support of itemsets/rules/hyperedgesets (default: 1)

arem: a character string indicating the used additional rule evaluation measure (default: "none") given by one of

- "none": no additional evaluation measure
- "diff": absolute confidence difference
- "quot": difference of confidence quotient to 1
- "aimp": absolute difference of improvement to 1
- "info": information difference to prior
- "chi2": normalized χ^2 measure

aval: a logical indicating whether to return the additional rule evaluation measure selected with `arem`.

minval: a numeric value for the minimal value of additional evaluation measure selected with `arem` (default: 0.1)

originalSupport: a logical indicating whether to use for minimum support the original definition of the support of a rule (lhs and rhs) instead of lhs support. Make sure to use `ext = TRUE` if `originalSupport` is set to `FALSE` (default: `TRUE`)

Additional slots for Eclat in `ECparameter`:

tidLists: a logical indicating whether to return also a list of supporting transactions (transaction IDs) (default: `FALSE`)

Methods

coerce signature(from = "NULL", to = "APparameter")

coerce signature(from = "list", to = "APparameter")

coerce signature(from = "NULL", to = "ECparameter")

coerce signature(from = "list", to = "ECparameter")

show signature(object = "ASparameter")

References

Christian Borgelt (2004) *Apriori — Finding Association Rules/Hyperedges with the Apriori Algorithm*. <http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori.html>

See Also

[apriori](#), [eclat](#)

associations-class *Class "associations" - A Set of Associations*

Description

The `associations` class is a virtual class which is extended to represent mining result (e.g., sets of itemsets or rules). The class provides accessors for the quality slot and a method for sorting the associations.

Objects from the Class

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

Slots

quality: a data.frame for quality measures (e.g., interest measures as support or confidence). Each quality measure is a named vector with the same length as the number of elements in the set of associations and each vector element belongs to the association with the same index.

info: a list which is used to store algorithm specific mining information. Typically it contains at least the elements "data" (name of the transaction data set), "ntransactions" (length of the data set), "support" (the minimum support used for mining).

Methods

info<- signature(x = "associations"); replaces the info list.

info signature(x = "associations"); returns the info list.

items signature(x = "associations"); dummy method. This method has to be implemented by all subclasses of `associations` and return the items which make up each association as an object of class `itemMatrix`.

labels signature(object = "associations"); dummy method. This method has to be implemented by all subclasses of `associations` and return a vector of length(object) of labels for the elements in the association.

length signature(x = "associations"); dummy method. This method has to be implemented by all subclasses of `associations` and return the number of elements in the association.

quality<- signature(x = "associations"); replaces the quality data.frame. The lengths of the vectors in the data.frame have to equal the number of associations in the set.

quality signature(x = "associations"); returns the quality data.frame.

show signature(object = "associations")

Subclasses

`itemsets-class`, `rules-class`

See Also

`SORT`, `WRITE`, `length`, `is.subset`, `is.superset`, `sets`, `unique`, `itemMatrix-class`

`combine`*Combining Objects*

Description

Provides the S4 methods to combine several objects based on `itemMatrix` into a single object.

Note, use `union` rather than `c` to combine several mined `itemsets` (or `rules`) into a single set.

Usage

```
## S4 method for signature 'itemMatrix':  
c(x, ..., recursive = FALSE)  
  
## S4 method for signature 'transactions':  
c(x, ..., recursive = FALSE)  
  
## S4 method for signature 'rules':  
c(x, ..., recursive = FALSE)  
  
## S4 method for signature 'itemsets':  
c(x, ..., recursive = FALSE)
```

Arguments

<code>x</code>	first object.
<code>...</code>	further objects of the same class as <code>x</code> to be combined.
<code>recursive</code>	a logical. If <code>recursive=TRUE</code> , the function recursively descends through lists combining all their elements into a vector.

Value

An object of the same class as `x`.

See Also

[itemMatrix-class](#), [transactions-class](#), [rules-class](#), [itemsets-class](#)

Examples

```
data("Adult")  
  
## combine transactions  
a1 <- Adult[1:10]  
a2 <- Adult[101:110]  
  
aComb <- c(a1, a2)  
summary(aComb)
```

```
## combine rules
r1 <- apriori(Adult[1:1000])
r2 <- apriori(Adult[1001:2000])
rComb <- unique(c(r1, r2))
rComb
```

coverage

Calculate coverage for rules

Description

Provides the generic function and the needed S4 method to calculate the coverage (support of the left-hand-side) of rules.

Usage

```
coverage(x, transactions = NULL)
```

Arguments

`x` the set of rules.
`transactions` the data set used to generate 'x'. Only needed if the quality slot of 'x' does not contain support and confidence.

Details

Coverage is calculated from the rules quality measures (support and confidence) stored in the quality slot or, if these values are not present, as the support of the LHS.

Value

A numeric vector of the same length as `x` containing the coverage values for the sets in `x`.

See Also

[rules-class](#)

Examples

```
data("Income")

## find and some rules (we only use 5 rules here) and calculate coverage
rules <- apriori(Income)[1:5]
quality(rules) <- cbind(quality(rules), coverage = coverage(rules))

inspect(rules)
```

crossTable	<i>Cross-tabulate joint occurrences across pairs of items</i>
------------	---

Description

Provides the generic function `crossTable` and the S4 method to cross-tabulate joint occurrences across pairs of items.

Usage

```
crossTable(x, ...)
```

Arguments

`x` object to be cross-tabulated (`transactions` or `itemMatrix`).
`...` further arguments (currently unused).

Value

A symmetric matrix of n time n , where n is the number of items times in `x`. The matrix contains the co-occurrence counts between pairs of items.

See Also

[transactions-class](#), [itemMatrix-class](#).

Examples

```
data("Groceries")  
  
ct <- crossTable(Groceries)  
ct[1:5, 1:5]
```

dissimilarity	<i>Dissimilarity Computation</i>
---------------	----------------------------------

Description

Provides the generic function `dissimilarity` and the S4 methods to compute and returns distances for binary data in a matrix, [transactions](#) or [associations](#).

Usage

```
dissimilarity(x, y = NULL, method = NULL, args = NULL, ...)
## S4 method for signature 'itemMatrix':
dissimilarity(x, y = NULL, method = NULL, args = NULL,
             which = "transactions")
## S4 method for signature 'associations':
dissimilarity(x, y = NULL, method = NULL, args = NULL,
             which = "transactions")
## S4 method for signature 'matrix':
dissimilarity(x, y = NULL, method = NULL, args = NULL)
```

Arguments

x	the set of elements (e.g., matrix, itemMatrix, transactions, itemsets, rules).
y	NULL or a second set to calculate cross dissimilarities.
method	the distance measure to be used. Implemented measures are (defaults to "jaccard"): <ul style="list-style-type: none"> "affinity": measure based on the affinity, a similarity measure between items. It is defined as the average <i>affinity</i> between the items in two transactions (see Aggarwal et al. (2002)). "cosine": the <i>cosine</i> distance. "dice": the <i>Dice's coefficient</i> defined by Dice (1945). Similar to <i>Jaccard</i> but gives double the weight to agreeing items. "jaccard": the number of items which occur in both elements divided by the total number of items in the elements (Sneath, 1957). This measure is often also called: <i>binary</i>, <i>asymmetric binary</i>, etc. "matching": the <i>Matching coefficient</i> defined by Sokal and Michener (1958). This coefficient gives the same weight to presents and absence of items. "pearson": the 1 - <i>Pearson correlation coefficient</i> .
args	a list of additional arguments for the methods. For calculating "affinity" for associations, the affinities between the items in the transactions are needed and passed to the method as the first element in args.
which	a character string indicating if the dissimilarity should be calculated between transavtions (default) or items (use "items").
...	further arguments.

Value

returns an object of class `dist`.

References

Sneath, P. H. A. (1957) Some thoughts on bacterial classification. *Journal of General Microbiology* 17, pages 184–200.

Sokal, R. R. and Michener, C. D. (1958) A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin* 38, pages 1409–1438.

Dice, L. R. (1945) Measures of the amount of ecologic association between species. *Ecology* 26, pages 297–302.

Charu C. Aggarwal, Cecilia Procopiuc, and Philip S. Yu. (2002) Finding localized associations in market basket data. *IEEE Trans. on Knowledge and Data Engineering* 14(1):51–62.

See Also

[affinity](#), [dist-class](#), [itemMatrix-class](#), [associations-class](#).

Examples

```
## cluster items in Groceries with support > 5%
data("Groceries")

s <- Groceries[,itemFrequency(Groceries)>0.05]
d_jaccard <- dissimilarity(s, which = "items")
plot(hclust(d_jaccard, method = "ward"))

## cluster transactions for a sample of Adult
data("Adult")
s <- sample(Adult, 200)

## calculate Jaccard distances and do hclust
d_jaccard <- dissimilarity(s)
plot(hclust(d_jaccard))

## calculate affinity-based distances and do hclust
d_affinity <- dissimilarity(s, method = "affinity")
plot(hclust(d_affinity))

## cluster rules
rules <- apriori(Adult)
rules <- subset(rules, subset = lift > 2)

## we need to supply the item affinities from the dataset (sample)
d_affinity <- dissimilarity(rules, method = "affinity",
  args = list(affinity = affinity(s)))
plot(hclust(d_affinity))
```

Description

Provides the generic function `duplicated` and the S4 methods for `itemMatrix` and `associations`. `duplicated` finds duplicated elements in an `itemMatrix`. It returns a logical vector indicating which elements are duplicates.

Note that `duplicated` can also be used to find transactions with identical items and identical rules and itemsets stored in `rules` and `itemsets`.

Usage

```
duplicated(x, incomparables = FALSE, ...)
```

Arguments

`x` an object of class `itemMatrix` or `associations`.
`...` further arguments (currently unused).
`incomparables` argument currently unused.

Value

A logical vector indicating duplicated elements.

See Also

[unique](#), [rules-class](#), [itemsets-class](#), [itemMatrix-class](#)

Examples

```
data("Adult")

r1 <- apriori(Adult[1:1000], parameter = list(support = 0.5))
r2 <- apriori(Adult[1001:2000], parameter = list(support = 0.5))

## Note this creates a collection of rules from two sets of rules
r_comb <- c(r1, r2)
duplicated(r_comb)
```

Description

Mine frequent itemsets with the Eclat algorithm. This algorithm uses simple intersection operations for equivalence class clustering along with bottom-up lattice traversal.

Usage

```
eclat(data, parameter = NULL, control = NULL)
```

Arguments

data	object of class <code>transactions</code> or any data structure which can be coerced into <code>transactions</code> (e.g., <code>binary matrix</code> , <code>data.frame</code>).
parameter	object of class <code>ECparameter</code> or named list (default values are: support 0.1 and maxlen 5)
control	object of class <code>ECcontrol</code> or named list for algorithmic controls.

Details

Calls the C implementation of the Eclat algorithm by Christian Borgelt for mining frequent itemsets.

Value

Returns an object of class `itemsets`.

References

Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. (1997) *New algorithms for fast discovery of association rules*. Technical Report 651, Computer Science Department, University of Rochester, Rochester, NY 14627.

Christian Borgelt (2003) Efficient Implementations of Apriori and Eclat. *Workshop of Frequent Item Set Mining Implementations* (FIMI 2003, Melbourne, FL, USA).

See Also

[ECparameter-class](#), [ECcontrol-class](#), [transactions-class](#), [itemsets-class](#), [apriori](#)

Examples

```
data("Adult")
## Mine itemsets with minimum support of 0.1.
itemsets <- eclat(Adult,
                  parameter = list(supp = 0.1, maxlen = 15))
```

Epub

Epub Data Set

Description

The Epub data set contains the download history of documents from the electronic publication platform of the Vienna University of Economics and Business Administration. The data was recorded between Jan 2003 and Mar 2006.

Usage

`data(Epub)`

Format

Object of class `transactions` with 3975 transactions and 465 items. Item labels are document IDs of the form "doc_11d". Session IDs and time stamps for transactions are also provided.

Source

Provided by Michael Hahsler from ePub-WU at <http://epub.wu-wien.ac.at>.

Groceries

Groceries Data Set

Description

The Groceries data set contains 1 month (30 days) of real-world point-of-sale transaction data from a typical local grocery outlet. The data set contains 9835 transactions and the items are aggregated to 169 categories.

If you use this data set in your paper, please refer to the paper in the references section.

Usage

`data(Groceries)`

Format

Object of class `transactions`.

Source

The data set is provided for arules by Michael Hahsler, Kurt Hornik and Thomas Reutterer.

References

Michael Hahsler, Kurt Hornik, and Thomas Reutterer (2006) Implications of probabilistic data modeling for mining association rules. In M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nuernberger, and W. Gaul, editors, *From Data and Information Analysis to Knowledge Engineering, Studies in Classification, Data Analysis, and Knowledge Organization*, pages 598–605. Springer-Verlag.

image

Visual Inspection of Binary Incidence Matrices

Description

Provides the S4 methods `image` to generate level plots to visually inspect binary incidence matrices, i.e., objects based on `itemMatrix` (e.g., transactions, `tidLists`, items in itemsets or rhs/lhs in rules). These plots can be used to identify problems in a data set (e.g., recording problems with some transactions containing all items).

Usage

```
## S4 method for signature 'itemMatrix':
image(x,
      xlab = "Items (Columns)",
      ylab = "Elements (Rows)", ...)

## S4 method for signature 'transactions':
image(x, ...)

## S4 method for signature 'tidLists':
image(x,
      xlab="Transactions (Columns)",
      ylab="Items/itemsets (Rows)", ...)
```

Arguments

<code>x</code>	the object (<code>itemMatrix</code> , <code>transactions</code> or <code>tidLists</code>).
<code>xlab</code> , <code>ylab</code>	labels for the plot.
<code>...</code>	further arguments passed on to <code>image</code> in package Matrix which in turn are passed on to <code>levelplot</code> in lattice .

See Also

[image](#) (for `dgTMatrix` in **Matrix**), [levelplot](#) (in **lattice**), [itemMatrix-class](#), [transactions-class](#), [tidLists-class](#)

Examples

```
data("Epub")

## in this data set we can see that not all
## items were available from the beginning.
image(Epub[1:1000])
```

Income

*Income Data Set***Description**

The IncomeESL data set originates from an example in the book ‘The Elements of Statistical Learning’ (see Section source). The data set is an extract from this survey. It consists of 8993 instances (obtained from the original data set with 9409 instances, by removing those observations with the annual income missing) with 14 demographic attributes. The data set is a good mixture of categorical and continuous variables with a lot of missing data. This is characteristic of data mining applications. The `Income` data set contains the data already prepared and coerced to [transactions](#).

Usage

```
data("Income")
data("IncomeESL")
```

Format

IncomeESL is a data frame with 8993 observations on the following 14 variables.

income an ordered factor with levels `[0, 10) < [10, 15) < [15, 20) < [20, 25) < [25, 30) < [30, 40) < [40, 50) < [50, 75) < 75+`

sex a factor with levels `male female`

marital status a factor with levels `married cohabitation divorced widowed single`

age an ordered factor with levels `14-17 < 18-24 < 25-34 < 35-44 < 45-54 < 55-64 < 65+`

education an ordered factor with levels `grade <9 <grades 9-11 <high school graduate <college (1-3 years) <college graduate <graduate study`

occupation a factor with levels `professional/managerial sales laborer clerical/service homemaker student military retired unemployed`

years in bay area an ordered factor with levels `<1 < 1-3 < 4-6 < 7-10 < >10`

dual incomes a factor with levels `not married yes no`

number in household an ordered factor with levels `1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9+`

number of children an ordered factor with levels `0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9+`

householder status a factor with levels `own rent live with parents/family`

type of home a factor with levels house condominium apartment mobile Home other
ethnic classification a factor with levels american indian asian black east indian
 hispanic pacific islander white other
language in home a factor with levels english spanish other

Details

To create `Income` (the transactions object), the original data frame in `IncomeESL` is prepared in a similar way as described in ‘The Elements of Statistical Learning.’ We removed cases with missing values and cut each ordinal variable (age, education, income, years in bay area, number in household, and number of children) at its median into two values (see Section examples).

Source

Impact Resources, Inc., Columbus, OH (1987).

Obtained from the web site of the book: Hastie, T., Tibshirani, R. & Friedman, J. (2001) *The Elements of Statistical Learning*. Springer-Verlag. (<http://www-stat.stanford.edu/~tibs/ElemStatLearn/>; called ‘Marketing’)

Examples

```
data("IncomeESL")
IncomeESL[1:3, ]

## remove incomplete cases
IncomeESL <- IncomeESL[complete.cases(IncomeESL), ]

## preparing the data set
IncomeESL[["income"]] <- factor((as.numeric(IncomeESL[["income"]]) > 6) +1,
  levels = 1 : 2 , labels = c("$0-$40,000", "$40,000+"))

IncomeESL[["age"]] <- factor((as.numeric(IncomeESL[["age"]]) > 3) +1,
  levels = 1 : 2 , labels = c("14-34", "35+"))

IncomeESL[["education"]] <- factor((as.numeric(IncomeESL[["education"]]) > 4) +1,
  levels = 1 : 2 , labels = c("no college graduate", "college graduate"))

IncomeESL[["years in bay area"]] <- factor(
  (as.numeric(IncomeESL[["years in bay area"]]) > 4) +1,
  levels = 1 : 2 , labels = c("1-9", "10+"))

IncomeESL[["number in household"]] <- factor(
  (as.numeric(IncomeESL[["number in household"]]) > 3) +1,
  levels = 1 : 2 , labels = c("1", "2+"))

IncomeESL[["number of children"]] <- factor(
  (as.numeric(IncomeESL[["number of children"]]) > 1) +0,
  levels = 0 : 1 , labels = c("0", "1+"))

## creating transactions
Income <- as(IncomeESL, "transactions")
```

Income

`inspect` *Display Associations and Transactions in Readable Form*

Description

Provides the generic function `inspect` and S4 methods to display associations and transactions plus additional information formatted for online inspection.

Usage

```
inspect(x, ...)
```

Arguments

`x` a set of associations or transactions or an `itemMatrix`.
`...` additional arguments (currently unused)

See Also

[itemMatrix-class](#), [itemsets-class](#), [rules-class](#), [transactions-class](#)

Examples

```
data("Adult")
rules <- apriori(Adult)
inspect(rules[1000])
```

`interestMeasure` *Calculating various additional interest measures*

Description

Provides the generic function `interestMeasure` and the needed S4 method to calculate various additional interest measures for existing sets of itemsets or rules.

Usage

```
interestMeasure(x, method, transactions = NULL, reuse = TRUE, ...)
```

Arguments

<code>x</code>	a set of itemsets or rules.
<code>method</code>	name or vector of names of the desired interest measures (see details for available measures).
<code>transactions</code>	the transaction data set used to mine the associations.
<code>reuse</code>	logical indicating if information in quality slot should be reuse for calculating the measures. This speedes up the process significantly since only very little (or no) transaction counting is necessary if support, confidence and lift are already available. Use <code>reuse=FALSE</code> to force counting (might be very slow).
<code>...</code>	further arguments for the measure calculation.

Details

For itemsets the following measures are implemented:

"allConfidence" (see, Omiencinski, 2003) is defined on itemsets as the minimum confidence of all possible rule generated from the itemset.

"crossSupportRatio" (see, Xiong et al., 2003) is defined on itemsets as the ratio of the support of the least frequent item to the support of the most frequent item. Cross-support patterns have a ratio smaller than a set threshold. Normally many found patterns are cross-support patterns which contain frequent as well as rare items. Such patterns often tend to be spurious.

"support" calculate itemset support.

For rules the following measures are implemented:

"chiSquare" (see Liu et al. 1999). The chi-square statistic to test for independence between the lhs and rhs of the rule. The critical value of the chi-square distribution with 1 degree of freedom (2x2 contingency table) at $\alpha = 0.05$ is 3.84; higher chi-square values indicate that the lhs and the rhs are not independent.

"confidence" calculate rule confidence. Range 0 ... 1.

"conviction" (see Brin et al. 1997) defined as $P(X)P(\bar{Y})/P(X \wedge \bar{Y})$. Range: 0.5 ... 1 ... ∞ (1 indicates unrelated items).

"cosine" (see Tan et al. 2004) equivalent to the IS measure. Range: 0 ... 1.

"coverage" calculate rule coverage (support of LHS). Range: 0 ... 1.

"doc" calculate difference of confidence, which is defined by Hofmann and Wilhelm (2001) as $\text{conf}(X \Rightarrow Y) - \text{conf}(\bar{X} \Rightarrow Y)$. Range: -1 ... 1.

"gini" gini index (see Tan et al. 2004). Range: 0 ... 1.

"hyperLift" (see, Hahsler and Hornik, 2007) is an adaptation of the lift measure which is more robust for low counts. It is based on the idea that under independence the count c_{XY} of the transactions which contain all items in a rule $X \Rightarrow Y$ follows a hypergeometric distribution (represented by the random variable C_{XY}) with the parameters given by the counts c_X and c_Y .

Lift is defined for the rule $X \Rightarrow Y$ as:

$$\text{lift}(X \Rightarrow Y) = \frac{P(X \cup Y)}{P(X)P(Y)} = \frac{c_{XY}}{E[C_{XY}]},$$

where $E[C_{XY}] = c_X c_Y / m$ with m being the number of transactions in the database.

Hyper-lift is defined as:

$$\text{hyperlift}(X \Rightarrow Y) = \frac{c_{XY}}{Q_\delta[C_{XY}]},$$

where $Q_\delta[C_{XY}]$ is the quantile of the hypergeometric distribution given by δ . The quantile can be given as parameter `d` (default: `d=0.99`). Range: $0 \dots \infty$.

"hyperConfidence" (Hahsler and Hornik, 2007) calculates the confidence level that we observe too high/low counts for rules $X \Rightarrow Y$ using the hypergeometric model. Since the counts are drawn from a hypergeometric distribution (represented by the random variable C_{XY}) with known parameters given by the counts c_X and c_Y , we can calculate a confidence interval for the observed counts c_{XY} stemming from the distribution. Hyperconfidence reports the confidence level (significance level if `significance=TRUE` is used) for

complements - $1 - P[C_{XY} \geq c_{XY} | c_X, c_Y]$

substitutes - $1 - P[C_{XY} < c_{XY} | c_X, c_Y]$.

A confidence level of, e.g., > 0.95 indicates that there is only a 5% chance that the count for the rule was generated randomly.

Per default complementary effects are mined, substitutes can be found by using the parameter `complements = FALSE`. Range: $0 \dots 1$.

"improvement" (see Bayardo et al. 2000) the improvement of a rule is the minimum difference between its confidence and the confidence of any proper sub-rule with the same consequent. Range: $0 \dots 1$.

"leverage" (see Piatetsky-Shapiro 1991) defined as $P(X \Rightarrow Y) - (P(X)P(Y))$. It measures the difference of X and Y appearing together in the data set and what would be expected if X and Y were statistically dependent. Range: $-1 \dots 1$.

"lift" calculate rule lift. Range: $0 \dots \infty$.

"oddsRatio" (see Tan et al. 2004). The odds of finding X in transactions which contain Y divided by the odds of finding X in transactions which do not contain Y. Range: $0 \dots 1 \dots \infty$ (1 indicates that Y is not associated to X).

"phi" the correlation coefficient ϕ (see Tan et al. 2004) Range: -1 (perfect neg. correlation) to $+1$ (perfect pos. correlation).

"RLD" (Relative Linkage Disequilibrium; see Kenett and Salini 2008). RLD evaluates the deviation of the support of the whole rule from the support expected under independence given the supports of the LHS and the RHS. The code was contributed by Silvia Salini. Range: $0 \dots 1$.

"support" calculate rule support. Range: $0 \dots 1$.

Value

If only one method is used, the function returns a numeric vector containing the values of the interest measure for each association in the set of associations `x`.

If more than one methods are specified, the result is a data.frame containing the different measures for each association.

References

- R. Bayardo, R. Agrawal, and D. Gunopulos (2000). Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 4(2/3):217–240, 2000.
- Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur (1997). Dynamic itemset counting and implication rules for market basket data. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 255–264, Tucson, Arizona, USA.
- Michael Hahsler and Kurt Hornik. New probabilistic interest measures for association rules. *Intelligent Data Analysis*, 11(5):437–455, 2007
- Heike Hofmann and Adalbert Wilhelm. Visual comparison of association rules. *Computational Statistics*, 16(3):399–415, 2001.
- Ron Kenett and Silvia Salini. Relative Linkage Disequilibrium: A New measure for association rules. In *8th Industrial Conference on Data Mining ICDM 2008 July 16–18, 2008, Leipzig/Germany*, to appear, 2008.
- Bing Liu, Wynne Hsu, and Yiming Ma (1999). Pruning and summarizing the discovered associations. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 125–134. ACM Press, 1999.
- Edward R. Omiecinski (2003). Alternative interest measures for mining associations in databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):57–69, Jan/Feb 2003.
- Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava (2004). Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313.
- Piatetsky-Shapiro, G. (1991). Discovery, analysis, and presentation of strong rules. In: *Knowledge Discovery in Databases*, pages 229–248.
- Hui Xiong, Pang-Ning Tan, and Vipin Kumar (2003). Mining strong affinity association patterns in data sets with skewed support distribution. In Bart Goethals and Mohammed J. Zaki, editors, *Proceedings of the IEEE International Conference on Data Mining*, November 19–22, 2003, Melbourne, Florida, pages 387–394.

See Also

[itemsets-class](#), [rules-class](#)

Examples

```
data("Income")
rules <- apriori(Income)

## calculate a single measure and add it to the quality slot
quality(rules) <- cbind(quality(rules),
  hyperConfidence = interestMeasure(rules, method = "hyperConfidence",
  Income))

inspect(head(SORT(rules, by = "hyperConfidence"))

## calculate several measures
m <- interestMeasure(rules, c("confidence", "oddsRatio", "leverage"), Income)
inspect(head(rules))
head(m)
```

is.closed	<i>Find Closed Itemsets</i>
-----------	-----------------------------

Description

Provides the generic function and the S4 method `is.closed` for finding closed itemsets. The closure of an itemset is its largest proper superset which has the same support (is contained in exactly the same transactions). An itemset is closed, if it is its own closure (Pasquier et al. 1999).

Usage

```
is.closed(x)
```

Arguments

`x` a set of itemsets.

Value

a logical vector with the same length as `x` indicating for each element in `x` if it is a closed itemset.

References

Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal (1999). Discovering frequent closed itemsets for association rules. In *Proceeding of the 7th International Conference on Database Theory*, Lecture Notes In Computer Science (LNCS 1540), pages 398–416. Springer, 1999.

See Also

[itemsets-class](#)

is.maximal	<i>Find Maximal Itemsets</i>
------------	------------------------------

Description

Provides the generic function and the S4 method `is.maximal` for finding maximal itemsets. An itemset is maximal in a set if no proper superset of the itemset is contained in the set (Zaki et al., 1997).

Usage

```
is.maximal(x, ...)  
## S4 method for signature 'itemMatrix':  
is.maximal(x)
```

Arguments

`x` the set of itemsets or an itemMatrix object.
`...` further arguments.

Value

a logical vector with the same length as `x` indicating for each element in `x` if it is a maximal itemset.

References

Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li (1997). *New algorithms for fast discovery of association rules*. Technical Report 651, Computer Science Department, University of Rochester, Rochester, NY 14627.

See Also

[is.superset](#), [itemMatrix-class](#), [itemsets-class](#)

<code>is.superset</code>	<i>Find Super and Subsets</i>
--------------------------	-------------------------------

Description

Provides the generic functions and the S4 methods `is.subset` and `is.superset` for finding super or subsets in associations and itemMatrix objects.

Usage

```
is.subset(x, y = NULL, proper = FALSE)
is.superset(x, y = NULL, proper = FALSE)
```

Arguments

`x`, `y` associations or itemMatrix objects. If `y = NULL`, the super or subset structure within set `x` is calculated.
`proper` a logical indicating if all or just proper super or subsets.

Details

looks for each element in `x` which elements in `y` are supersets or subsets. Note that the method can be very slow and memory intensive if `x` and/or `y` contain many elements.

Value

returns a logical matrix with `length(x)` rows and `length(y)` columns. Each logical row vector represents which elements in `y` are supersets (subsets) of the corresponding element in `x`. If either `x` or `y` have length zero, `NULL` is returned instead of a matrix.

See Also

[associations-class](#), [itemMatrix-class](#)

Examples

```
data("Adult")
set <- eclat(Adult, parameter = list(supp = 0.8))

### find the supersets of each itemset in set
is.superset(set, set)
```

itemCoding

Item Coding – Handling Item Labels and Column IDs Conversions

Description

Provides the generic functions and the S4 methods for converting item labels into column IDs used in the binary matrix representation and vice versa.

`decode` converts from the numeric (column IDs) representation to readable item labels. `decode` is used by `LIST`.

`encode` converts from readable item labels to an `itemMatrix` using a given coding. With this method it is possible to create several compatible `itemMatrix` objects (i.e., use the same binary representation for items) from data.

`recode` recodes an `itemMatrix` object so its coding is compatible with another object.

Usage

```
decode(x, ...)
## S4 method for signature 'list':
decode(x, itemLabels)
## S4 method for signature 'numeric':
decode(x, itemLabels)

encode(x, ...)
## S4 method for signature 'list':
encode(x, itemLabels, itemMatrix = TRUE)
## S4 method for signature 'character':
encode(x, itemLabels, itemMatrix = TRUE)
## S4 method for signature 'numeric':
encode(x, itemLabels, itemMatrix = TRUE)

recode(x, ...)
## S4 method for signature 'itemMatrix':
recode(x, itemLabels = NULL, match = NULL)
```

Arguments

<code>x</code>	a vector or a list of vectors of character strings (for <code>encode</code>) or of numeric (for <code>decode</code>), or an object of class <code>itemMatrix</code> (for <code>recode</code>).
<code>itemLabels</code>	a vector of character strings used for coding where the position of an item label in the vector gives the item's column ID. The used <code>itemLabels</code> vector can be obtained from <code>itemMatrix</code> , <code>transactions</code> and <code>associations</code> by the method <code>itemLabels</code> .
<code>itemMatrix</code>	return an object of class <code>itemMatrix</code> otherwise an object of the same class as <code>x</code> is returned.
<code>match</code>	an <code>itemMatrix</code> object whose item coding <code>x</code> should match.
<code>...</code>	further arguments.

Value

`recode` always returns an object of class `itemMatrix`.

For `encode` with `itemMatrix = TRUE` an object of class `itemMatrix` is returned. Otherwise the result is of the same type as `x`, e.g., a list or a vector.

See Also

[LIST](#), [associations-class](#), [itemMatrix-class](#)

Examples

```
data("Adult")

## Example 1: Manual decoding
## get code
iLabels <- itemLabels(Adult)
head(iLabels)

## get undecoded list and decode in a second step
list <- LIST(Adult[1:5], decode = FALSE)
list

decode(list, itemLabels = iLabels)

## Example 2: Manually create an itemMatrix
data <- list(
  c("income=small", "age=Young"),
  c("income=large", "age=Middle-aged")
)

iM <- encode(data, iLabels)
iM

inspect(iM)

## use the itemMatrix to create transactions
```

```

as(iM, "transactions")

## Example 3: use recode
## select first 100 transactions and all education-related items
sub <- Adult[1:100, itemInfo(Adult)$variables == "education"]
itemLabels(sub)
image(sub)

## recode to match Adult again
sub.recoded <- recode(sub, match = Adult)
image(sub.recoded)

```

itemFrequency

Getting Frequency/Support for Single Items

Description

Provides the generic function `itemFrequency` and S4 methods to get the frequency/support for all single items in an objects based on `itemMatrix`. For example, it is used to get the single item support from an object of class `transactions` without mining.

Usage

```

itemFrequency(x, ...)

## S4 method for signature 'itemMatrix':
itemFrequency(x, type)

```

Arguments

<code>x</code>	an object.
<code>...</code>	further arguments are passed on.
<code>type</code>	a character string specifying if "relative" frequency/support or "absolute" frequency/support (item counts) is returned. (default: "relative").

Value

`itemFrequency` returns a named numeric vector. Each element is the frequency/support of the corresponding item in object `x`. The items appear in the vector in the same order as in the binary matrix in `x`.

See Also

[itemFrequencyPlot](#), [itemMatrix-class](#), [transactions-class](#)

Examples

```

data("Adult")
itemFrequency(Adult, type = "relative")

```

 itemFrequencyPlot *Creating a Item Frequencies/Support Bar Plot*

Description

Provides the generic function `itemFrequencyPlot` and the S4 method to create an item frequency bar plot for inspecting the item frequency distribution for objects based on `itemMatrix` (e.g., `transactions`, or items in `itemsets` and `rules`).

Usage

```
itemFrequencyPlot(x, ...)
## S4 method for signature 'itemMatrix':
itemFrequencyPlot(x, type = c("relative", "absolute"),
  support = NULL, topN = NULL,
  population = NULL, popCol = "black", popLwd = 1,
  lift = FALSE, horiz = FALSE,
  names = TRUE, cex.names = par("cex.axis"),
  xlab = NULL, ylab = NULL, mai = NULL, ...)
```

Arguments

<code>x</code>	the object to be plotted.
<code>...</code>	further arguments are passed on (see <code>barplot</code> from possible arguments).
<code>type</code>	a character string indicating whether item frequencies should be displayed relative of absolute.
<code>support</code>	a numeric value. Only display items which have a support of at least <code>support</code> . If no population is given, <code>support</code> is calculated from <code>x</code> otherwise from the population. Support is interpreted relative or absolute according to the setting of <code>type</code> .
<code>topN</code>	a integer value. Only plot the <code>topN</code> items with the highest item frequency or lift (if <code>lift = TRUE</code>). The items are plotted ordered by descending support.
<code>population</code>	object of same class as <code>x</code> ; if <code>x</code> is a segment of a population, the population mean frequency for each item can be shown as a line in the plot.
<code>popCol</code>	plotting color for population.
<code>popLwd</code>	line width for population.
<code>lift</code>	a logical indicating whether to plot the lift ratio between instead of frequencies. The lift ratio is gives how many times an item is more frequent in <code>x</code> than in population.
<code>horiz</code>	a logical. If <code>horiz = FALSE</code> (default), the bars are drawn vertically. If <code>TRUE</code> , the bars are drawn horizontally.
<code>names</code>	a logical indicating if the names (bar labels) should be displayed?
<code>cex.names</code>	a numeric value for the expansion factor for axis names (bar labels).

xlab	a character string with the label for the x axis (use an empty string to force no label).
ylab	a character string with the label for the y axis (see xlab).
mai	a numerical vector giving the plots margin sizes in inches (see ‘? par’).

Value

A numeric vector with the midpoints of the drawn bars; useful for adding to the graph.

See Also

[itemFrequency](#), [itemMatrix-class](#)

Examples

```
data(Adult)

## the following example compares the item frequencies
## of people with a large income (boxes) with the average in the data set
Adult.largeIncome <- Adult[Adult %in%
  "income=large"]

## simple plot
itemFrequencyPlot(Adult.largeIncome)

## plot with the averages of the population plotted as a line
## (for first 72 variables/items)
itemFrequencyPlot(Adult.largeIncome[, 1:72],
  population = Adult[, 1:72])

## plot lift ratio (frequency in x / frequency in population)
## for items with a support of 20% in the population
itemFrequencyPlot(Adult.largeIncome,
  population = Adult, support = 0.2,
  lift = TRUE, horiz = TRUE)
```

itemMatrix-class *Class “itemMatrix” — Sparse Binary Incidence Matrix to Represent Sets of Items*

Description

The `itemMatrix` class is the basic building block for transactions, itemsets and rules in package **arules**. The class contains a sparse Matrix representation of items (a set of itemsets or transactions) and the corresponding item labels.

Objects from the Class

Objects can be created by calls of the form `new("itemMatrix", ...)`. However, most of the time objects will be created by coercion from a matrix, list or data.frame.

Slots

data: Object of class `ngCMatrix` (from package **Matrix**) which stores item occurrences in sparse representation. Note that the `ngCMatrix` is column-oriented and `itemMatrix` is row-oriented with each row representing an element (an itemset, a transaction, etc.). As a result, the `ngCMatrix` in this slot is always a transposed version of the binary incidence matrix in `itemMatrix`.

itemInfo: a `data.frame` which contains named vectors of the length equal to the number of elements in the set. If the slot is not empty (contains no item labels), the first element in the `data.frame` must have the name "labels" and contain a character vector with the item labels used for representing an item. In addition to the item labels, the `data.frame` can contain arbitrary named vectors (of the same length) to represent, e.g., variable names and values which were used to create the binary items or hierarchical category information associated with each item label.

itemsetInfo: a `data.frame` which may contain additional information for the rows (mostly representing itemsets) in the matrix.

Methods

coerce signature(`from` = "matrix", `to` = "itemMatrix"); expects `from` to be a binary matrix only containing 0s and 1s.

coerce signature(`from` = "list", `to` = "itemMatrix"); `from` is a list of vectors. Each vector contains one set/transaction/...

coerce signature(`from` = "itemMatrix", `to` = "ngCMatrix"); access the sparse matrix representation. Note, the `ngCMatrix` contains a transposed form of the `itemMatrix`.

coerce signature(`from` = "itemMatrix", `to` = "dgCMatrix"); access the sparse matrix representation. Note, the `dgCMatrix` contains a transposed form of the `itemMatrix`.

coerce signature(`from` = "itemMatrix", `to` = "matrix"); coerces to a dense 0-1 matrix of storage.mode "integer" instead of "double" to save memory.

coerce signature(`from` = "itemMatrix", `to` = "list"); see also the methods for `LIST`.

dim signature(`x` = "itemMatrix"); returns the dimensions of the `itemMatrix`.

%in% signature(`x` = "itemMatrix", `table` = "character"); matches the strings in `table` against the item labels in `x` and returns a logical vector indicating if a row (itemset) in `x` contains *any* of the items specified in `table`. Note that there is a `%in%` method with signature(`x` = "itemMatrix", `table` = "character"). This method is described in together with `match`.

%ain% signature(`x` = "itemMatrix", `table` = "character"); matches the strings in `table` against the item labels in `x` and returns a logical vector indicating if a row (itemset) in `x` contains *all* of the items specified in `table`.

%pin% signature(`x` = "itemMatrix", `table` = "character"); matches the strings in `table` against the item labels in `x` (using *partial* matching) and returns a logical vector indicating if a row (itemset) in `x` contains *any* of the items specified in `table`.

itemLabels signature(`object` = "itemMatrix"); returns the item labels used for encoding as a character vector.

itemLabels<- signature(object = "itemMatrix"); replaces the item labels used for encoding.

itemInfo signature(object = "itemMatrix"); returns the whole item/column information data.frame including labels.

itemInfo<- signature(object = "itemMatrix"); replaces the item/column info by a data.frame.

itemsetInfo signature(object = "itemMatrix"); returns the item set/row information data.frame.

itemsetInfo<- signature(object = "itemMatrix"); replaces the item set/row info by a data.frame.

labels signature(x = "transactions"); returns the labels (item labels and element names) for the matrix as a list of two vectors named `items` and `elements`. The following arguments can be used to customize the representation of the elements: `itemSep`, `setStart` and `setEnd`.

nitems signature(x = "itemMatrix"); returns the number of items (number in columns) in the itemMatrix.

show signature(object = "itemMatrix")

summary signature(object = "itemMatrix")

See Also

[LIST](#), [c](#), [duplicated](#), [inspect](#), [is.subset](#), [is.superset](#), [itemFrequency](#), [itemFrequencyPlot](#), [match](#), [length](#), [sets](#), [subset](#), [unique](#), [\[-methods](#), [image](#), [ngCMatrix-class](#) (from [Matrix](#)), [transactions-class](#), [itemsets-class](#), [rules-class](#)

Examples

```
## Generate random data and coerce data to itemMatrix.
m <- matrix(as.integer(runif(100000)>0.8), ncol=20)
dimnames(m) <- list(NULL, paste("item", c(1:20), sep=""))
i <- as(m, "itemMatrix")

## Get the number of elements (rows) in the itemMatrix.
length(i)

## Get first 5 elements (rows) of the itemMatrix as list.
as(i[1:5], "list")

## Get first 5 elements (rows) of the itemMatrix as matrix.
as(i[1:5], "matrix")

## Get first 5 elements (rows) of the itemMatrix as sparse ngCMatrix.
## Warning: for efficiency reasons, the ngCMatrix you get is transposed!
as(i[1:5], "ngCMatrix")
```

itemsets-class Class “itemsets” — A Set of Itemsets

Description

The `itemsets` class represents a set of itemsets and the associated quality measures.

Note that the class can also represent a multiset of itemsets with duplicated elements. Duplicated elements can be removed with `unique`.

Objects from the Class

Objects are the result of calling the functions `apriori` (e.g., with `target="frequent itemsets"` in the parameter list) or `eclat`. Objects can also be created by calls of the form `new("itemsets", ...)`.

Slots

items: object of class `itemMatrix` containing the items in the set of itemsets

quality: a `data.frame` containing the quality measures for the itemsets

tidLists: object of class `tidLists` containing the IDs of the transactions which support each itemset. The slot contains `NULL` if no transactions ID list is available (transactions ID lists are only available for `eclat`).

Extends

Class `associations`, directly.

Methods

coerce signature(`from` = "itemsets", `to` = "data.frame"); represent the itemsets in readable form

items signature(`x` = "itemsets"); returns the `itemMatrix` representing the set of itemsets

items<- signature(`x` = "itemsets"); replaces the `itemMatrix` representing the set of itemsets

itemInfo signature(`object` = "itemsets"); returns the whole item information data frame including item labels

labels signature(`object` = "itemsets"); returns labels for the itemsets as a character vector. The labels have the following format: "item1, item2,..., itemn"

itemLabels signature(`object` = "itemsets"); returns the item labels used to encode the itemsets as a character vector. The index for each label is the column index of the item in the binary matrix.

summary signature(`object` = "itemsets")

tidLists signature(`object` = "itemsets"); returns the transaction ID list

See Also

[\[-methods\]](#), [apriori](#), [c](#), [duplicated](#), [eclat](#), [inspect](#), [is.maximal](#), [length](#), [match](#), [sets](#), [size](#), [subset](#), [associations-class](#), [tidLists-class](#)

Examples

```
data("Adult")

## Mine frequent itemsets with Eclat.
fsets <- eclat(Adult, parameter = list(supp = 0.5))

## Display the 5 itemsets with the highest support.
fsets.top5 <- SORT(fsets)[1:5]
inspect(fsets.top5)

## Get the itemsets as a list
as(items(fsets.top5), "list")

## Get the itemsets as a binary matrix
as(items(fsets.top5), "matrix")

## Get the itemsets as a sparse matrix, a ngCMatrix from package Matrix.
## Warning: for efficiency reasons, the ngCMatrix you get is transposed
as(items(fsets.top5), "ngCMatrix")
```

length

Getting the Number of Elements

Description

S4 methods for `length` which return the number of elements of objects defined in the package **arules**.

Usage

```
## S4 method for signature 'rules':
length(x)

## S4 method for signature 'itemsets':
length(x)

## S4 method for signature 'tidLists':
length(x)

## S4 method for signature 'itemMatrix':
length(x)
```

Arguments

`x` an object of class `transactions`, `rules`, `itemsets`, `tidLists`, or `itemMatrix`.

Details

For `itemMatrix` and `transactions` the length is defined as the number of rows (transactions) in the binary incidence matrix.

For sets of associations (`rules`, `itemsets` and `associations` in general) the length is defined as the number of elements in the set (i.e., the number of rules or itemsets).

For `tidLists` the length is the number of lists (one per item or itemset) in the object.

Value

An integer scalar giving the “length” of `x`.

 LIST

List Representation for Objects Based on “itemMatrix”

Description

Provides the generic function `LIST` and the S4 methods to create a list representation from objects based on `itemMatrix` (e.g., `transactions`, `tidLists`, or `itemsets`). These methods can be used for the coercion to a list.

Usage

```
LIST(from, ...)
```

```
## S4 method for signature 'itemMatrix':
```

```
LIST(from, decode = TRUE)
```

```
## S4 method for signature 'transactions':
```

```
LIST(from, decode = TRUE)
```

```
## S4 method for signature 'tidLists':
```

```
LIST(from, decode = TRUE)
```

Arguments

`from` the object to be converted into a list.

`...` further arguments.

`decode` a logical controlling whether the items/transactions are decoded from the column numbers internally used by `itemMatrix` to the names stored in the object `from`. The default behavior is to decode.

Details

Using `LIST` with `decode = TRUE` is equivalent to the standard coercion `as(x, "list")`. `LIST` returns the object `from` as a list of vectors. Each vector represents one row of the `itemMatrix` (e.g., items in a transaction or itemset).

Value

a list primitive.

See Also

`decode`, `coerce`, `itemMatrix`, `list-method`, `itemMatrix-class`

Examples

```
data(Adult)

LIST(Adult[1:5])
LIST(Adult[1:5], decode = FALSE)
```

match

Value Matching

Description

Provides the generic function `match` and the S4 methods for associations and transactions. `match` returns a vector of the positions of (first) matches of its first argument in its second.

`%in%` is a more intuitive interface as a binary operator, which returns a logical vector indicating if there is a match or not for its left operand.

Usage

```
match(x, table, nomatch = NA_integer_, incomparables = NULL)

x %in% table
```

Arguments

<code>x</code>	an object of class <code>itemMatrix</code> , <code>transactions</code> or <code>associations</code> .
<code>table</code>	a set of associations or transactions to be matched against.
<code>nomatch</code>	the value to be returned in the case when no match is found.
<code>incomparables</code>	not implemented.

Value

`match`: An integer vector of the same length as `x` giving the position in `table` of the first match if there is a match, otherwise `nomatch`.

`%in%`: A logical vector, indicating if a match was located for each element of `x`.

See Also

[rules-class](#), [itemsets-class](#), [itemMatrix-class](#)

Examples

```
data("Adult")

## get unique transactions, count frequency of unique transactions
## and plot frequency of unique transactions
vals <- unique(Adult)
cnts <- tabulate(match(Adult, vals))
plot(sort(cnts, decreasing=TRUE))

## find all transactions which are equal to transaction 10 in Adult
which(Adult %in% Adult[10])
```

predict

Model Predictions

Description

Provides the S4 method `predict` for `itemMatrix` (e.g., transactions). Predicts the membership (nearest neighbor) of new data to clusters represented by medoids or labeled examples.

Usage

```
## S4 method for signature 'itemMatrix':
predict(object, newdata, labels = NULL, blocksize = 200, ...)
```

Arguments

<code>object</code>	medoids (no labels needed) or examples (labels needed).
<code>newdata</code>	objects to predict labels for.
<code>labels</code>	an integer vector containing the labels for the examples in <code>object</code> .
<code>blocksize</code>	a numeric scalar indicating how much memory <code>predict</code> can use for big <code>x</code> and/or <code>y</code> (approx. in MB). This is only a crude approximation for 32-bit machines (64-bit architectures need double the <code>blocksize</code> in memory) and using the default Jaccard method for dissimilarity calculation. In general, reducing <code>blocksize</code> will decrease the memory usage but will increase the run-time.
<code>...</code>	further arguments passed on to <code>dissimilarity</code> . E.g., <code>method</code> .

Value

An integer vector of the same length as `newdata` containing the predicted labels for each element.

See Also

`dissimilarity`, `itemMatrix-class`

Examples

```
data("Adult")

## sample
small <- sample(Adult, 500)
large <- sample(Adult, 5000)

## cluster a small sample
d_jaccard <- dissimilarity(small)
hc <- hclust(d_jaccard)
l <- cutree(hc, k=4)

## predict labels for a larger sample
labels <- predict(small, large, l)

## plot the profile of the 1. cluster
itemFrequencyPlot(large[labels==1, itemFrequency(large) > 0.1])
```

proximity-classes *Classes “dist”, “ar_cross_dissimilarity” and “ar_similarity” — Proximity Matrices*

Description

Simple classes to represent proximity matrices. For compatibility with clustering functions in R, we represent dissimilarities as the S3 class `dist`. For cross-dissimilarities and similarities, we provide the S4 classes `ar_cross_dissimilarities` and `ar_similarities`.

Objects from the Class

`dist` objects are the result of calling the method `dissimilarity` with one argument or any R function returning a S3 `dist` object.

`ar_cross_dissimilarity` objects are the result of calling the method `dissimilarity` with two arguments, by calls of the form `new("similarity", ...)`, or by coercion from matrix.

`ar_similarity` objects are the result of calling the method `affinity`, by calls of the form `new("similarity", ...)`, or by coercion from matrix.

Slots

The S4 classes have a method slot which contains the type of measure used for calculation.

See Also

`dist` (in package `stats`), `dissimilarity`, `affinity`.

```
random.transactions
```

Simulate a Random Transaction Data Set

Description

Simulates a random `transactions` object using different methods.

Usage

```
random.transactions(nItems, nTrans, method = "independent", ...,
                    verbose = FALSE)
```

Arguments

<code>nItems</code>	an integer. Number of items.
<code>nTrans</code>	an integer. Number of transactions.
<code>method</code>	name of the simulation method used (default: all items occur independently).
<code>...</code>	further arguments used for the specific simulation method (see details).
<code>verbose</code>	report progress.

Details

The function generates a `nitems` times `ntrans` transaction database.

Currently two simulation methods are implemented:

method "independent" (see **Hahsler et al., 2005**) All items are treated as independent. Each transaction is the result of `nItems` independent Bernoulli trials, one for each item with success probabilities given by the numeric vector `iProb` of length `nItems` (default: 0.01 for each item).

method "agrawal" (see **Agrawal and Srikant, 1994**) This method creates transactions with correlated items uses the following additional parameters:

lTrans average length of transactions.

nPats number of patterns (potential maximal frequent itemsets) used.

lPats average length of patterns.

corr correlation between consecutive patterns.

cmean mean of the corruption level (normal distr.).

cvar variance of the corruption level.

The simulation is a two-stage process. First, a set of `nPats` patterns (potential maximal frequent itemsets) is generated. The length of the patterns is Poisson distributed with mean `lPats` and consecutive patterns share some items controlled by the correlation parameter `corr`. For later use, for each pattern a pattern weight is generated by drawing from an exponential distribution with a mean of 1 and a corruption level is chosen from a normal distribution with mean `cmean` and variance `cvar`.

The patterns are created using the following function:

```
random.patterns(nItems, nPats = 2000, method = "agrawal", lPats
= 4, corr = 0.5, cmean = 0.5, cvar = 0.1, iWeight = NULL, verbose
= FALSE)
```

The function returns the patterns as an `itemsets` objects which can be supplied to `random.transactions` as the argument `patterns`. If no argument `patterns` is supplied, the default values given above are used.

In the second step, the transactions are generated using the patterns. The length the transactions follows a Poisson distribution with mean `lPats`. For each transaction, patterns are randomly chosen using the pattern weights till the transaction length is reached. For each chosen pattern, the associated corruption level is used to drop some items before adding the pattern to the transaction.

Value

Returns an object of class `transactions`.

References

Michael Hahsler, Kurt Hornik, and Thomas Reutterer (2006). Implications of probabilistic data modeling for mining association rules. In M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nuernberger, and W. Gaul, editors, *From Data and Information Analysis to Knowledge Engineering, Studies in Classification, Data Analysis, and Knowledge Organization*, pages 598–605. Springer-Verlag.

Rakesh Agrawal and Ramakrishnan Srikant (1994). Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, Santiago, Chile.

See Also

`transactions-class`.

Examples

```
## generate random 1000 transactions for 200 items with
## a success probability decreasing from 0.2 to 0.0001
## using the method described in Hahsler et al. (2006).
trans <- random.transactions(nItems = 200, nTrans = 1000,
  iProb = seq(0.2, 0.0001, length=200))

## display random data set
image(trans)

## use the method by Agrawal and Srikant (1994) to simulate transactions
```

```
## which contains correlated items. This should create data similar to
## T10I4D100K (just only 1000 transactions)
patterns <- random.patterns(nItems = 1000)
summary(patterns)

trans2 <- random.transactions(nItems = 1000, nTrans = 1000,
  method = "agrawal", patterns = patterns)
image(trans2)

## plot data with items ordered by item frequency
image(trans2[,order(itemFrequency(trans2), decreasing=TRUE)])
```

read.transactions *Read Transaction Data*

Description

Reads a transaction data file from disk and creates a `transactions` object.

Usage

```
read.transactions(file, format = c("basket", "single"), sep = NULL,
  cols = NULL, rm.duplicates = FALSE)
```

Arguments

<code>file</code>	the file name.
<code>format</code>	a character string indicating the format of the data set. One of "basket" or "single", can be abbreviated.
<code>sep</code>	a character string specifying how fields are separated in the data file, or NULL (default). For basket format, this can be a regular expression; otherwise, a single character must be given. The default corresponds to white space separators.
<code>cols</code>	For the 'single' format, <code>cols</code> is a numeric vector of length two giving the numbers of the columns (fields) with the transaction and item ids, respectively. For the 'basket' format, <code>cols</code> can be a numeric scalar giving the number of the column (field) with the transaction ids. If <code>cols = NULL</code>
<code>rm.duplicates</code>	a logical value specifying if duplicate items should be removed from the transactions.

Details

For 'basket' format, each line in the transaction data file represents a transaction where the items (item labels) are separated by the characters specified by `sep`. For 'single' format, each line corresponds to a single item, containing at least ids for the transaction and the item.

Value

Returns an object of class `transactions`.

See Also

`transactions-class`

Examples

```
## create a demo file using basket format for the example
data <- paste("item1,item2","item1","item2,item3", sep="\n")
cat(data)
write(data, file = "demo_basket")

## read demo data
tr <- read.transactions("demo_basket", format = "basket", sep=",")
inspect(tr)

## create a demo file using single format for the example
## column 1 contains the transaction ID and column 2 contains one item
data <- paste("trans1 item1", "trans2 item1","trans2 item2", sep = "\n")
cat(data)
write(data, file = "demo_single")

## read demo data
tr <- read.transactions("demo_single", format = "single", cols = c(1,2))
inspect(tr)

## tidy up
unlink("demo_basket")
unlink("demo_single")
```

ruleInduction

Rule Induction for a Set of Itemsets

Description

Provides the generic function and the needed S4 method to induce all rules which can be generated by the given itemsets from a transactions data set.

Usage

```
ruleInduction(x, ...)
## S4 method for signature 'itemsets':
ruleInduction(x, transactions, confidence = 0.8,
              control = NULL)
```

Arguments

<code>x</code>	the set of itemsets from which rules will be induced.
<code>...</code>	further arguments.
<code>transactions</code>	the transaction data set used to mine the itemsets.
<code>confidence</code>	a numeric value giving the minimum confidence for the rules.
<code>control</code>	a named list with elements <code>method</code> indicating the method ("apriori" or "ptree"), and the logical arguments <code>reduce</code> and <code>verbose</code> to indicate if unused items are removed and if the output should be verbose. Currently, "ptree" is the default method.

Details

If in control `method = "apriori"` is used, a very simple rule induction method is used. All rules are mined from the transactions data set using Apriori with the minimal support found in itemsets. And in a second step all rules which do not stem from one of the itemsets are removed. This procedure will be in many cases very slow (e.g., for itemsets with many elements or very low support).

If in control `method = "ptree"` is used, the transactions are counted into a prefix tree and then the rules are selectively generated using the counts in the tree. This is usually faster than the above approach.

If in control `reduce = TRUE` is used, unused items are removed from the data before creating rules. This might be slower for large transaction data sets. However, if `method = "ptree"` this is highly recommended as the items are further reordered to reduce the counting time.

If argument `transactions` is missing it is assumed that `x` contains a lattice (complete set) of frequent itemsets together with their support counts. Then rules can be induced directly without support counting. This approach is very fast.

Value

An object of class `rules`.

References

Michael Hahsler, Christian Buchta, and Kurt Hornik. Selective association rule generation. *Computational Statistics*, 23(2):303-315, April 2008

See Also

[itemsets-class](#), [rules-class](#) [transactions-class](#)

Examples

```
data("Adult")

## find all closed frequent itemsets
closed <- apriori(Adult,
  parameter = list(target = "closed", support = 0.4))
```

```
## rule induction
rules <- ruleInduction(closed, Adult, control = list(verbose = TRUE))
summary(rules)

## inspect the resulting rules
inspect(SORT(rules, by = "lift")[1:5])

## use lattice of frequent itemsets
ec <- eclat(Adult, parameter = list(support = 0.4))
rec <- ruleInduction(ec)
inspect(rec[1:5])
```

rules-class

Class "rules" — A Set of Rules

Description

The rules class represents a set of rules.

Note that the class can also represent a multiset of rules with duplicated elements. Duplicated elements can be removed with [unique](#).

Objects from the Class

Objects are the result of calling the function [apriori](#). Objects can also be created by calls of the form `new("rules", ...)`.

Slots

lhs: Object of class [itemMatrix](#); the left-hand-sides of the rules (antecedents)

rhs: Object of class [itemMatrix](#); the right-hand-sides of the rules (consequents)

quality: a data.frame

Extends

Class [associations](#), directly.

Methods

coerce signature(from = "rules", to = "data.frame"); represents the set of rules as a data.frame

itemInfo signature(object = "rules"); returns the whole item information data frame including item labels

itemLabels signature(object = "rules"); returns the item labels used to encode the rules

items signature(x = "rules"); returns for each rule the union of the items in the lhs and rhs (i.e., the itemsets which generated the rule) as an `itemMatrix`

generatingItemsets signature(x = "rules"); returns a collection of the itemsets which generated the rules (one itemset for each rule). Note that the collection can be a multiset and contain duplicated elements. Use `unique` to remove duplicates and obtain a proper set.

labels signature(object = "rules"); returns labels for the rules ("lhs => rhs") as a character vector. The representation can be customized using the additional parameter `ruleSep` and parameters for label defined in `itemMatrix`

itemLabels signature(object = "rules"); returns the item labels as a character vector. The index for each label is the column index of the item in the binary matrix.

lhs signature(x = "rules"); returns the `itemMatrix` representing the left-hand-side of the rules (antecedents)

lhs<- signature(x = "rules"); replaces the `itemMatrix` representing the left-hand-side of the rules (antecedents)

rhs signature(x = "rules"); returns the `itemMatrix` representing the right-hand-side of the rules (consequents)

rhs<- signature(x = "rules"); replaces the `itemMatrix` representing the right-hand-side of the rules (consequents)

summary signature(object = "rules")

See Also

[\[-methods\]](#), [apriori](#), [c](#), [duplicated](#), [inspect](#), [length](#), [match](#), [sets](#), [size](#), [subset](#), [associations-class](#), [itemMatrix-class](#),

Examples

```
data("Adult")

## Mine rules.
rules <- apriori(Adult, parameter = list(support = 0.4))

## Select a subset of rules using partial matching on the items
## in the right-hand-side and a quality measure
rules.sub <- subset(rules, subset = rhs %pin% "sex" & lift > 1.3)

## Display rules.
inspect(SORT(rules.sub)[1:3])
```

Description

Provides the generic function `sample` and the S4 method to take a sample of the specified size from the elements of `x` using either with or without replacement. `sample` can be used to sample from a set of transactions or associations.

Usage

```
sample(x, size, replace = FALSE, prob = NULL)
```

Arguments

`x` object to be sampled from (a set of associations or transactions).
`size` sample size.
`replace` a logical. Sample with replacement?
`prob` a numeric vector of probability weights.

Value

An object of the same class as `x`.

See Also

[associations-class](#), [transactions-class](#), [itemMatrix-class](#).

Examples

```
data("Adult")

## sample with replacement
s <- sample(Adult, 500, replace = TRUE)
s
```

sets

Set Operations

Description

Provides the generic functions and the S4 methods for the set operations `union`, `intersect`, `setequal`, `setdiff` and `is.element` on sets of associations (e.g., rules, itemsets) and itemMatrix.

Usage

```
union(x, y)
intersect(x, y)
setequal(x, y)
setdiff(x, y)
is.element(el, set)
```

Arguments

`x`, `y`, `el`, `set` sets of associations or itemMatrix objects.

Details

All S4 methods for set operations are defined for the class name "ANY" in the signature, so they should work for all S4 classes for which the following methods are available: `match`, `length` and `unique`.

Value

`union`, `intersect`, `setequal` and `setdiff` return an object of the same class as `x` and `y`.

`is.element` returns a logic vector of length `el` indicating for each element if it is included in `set`.

See Also

[associations-class](#), [itemMatrix-class](#)

Examples

```
data("Adult")

## mine some rules
r <- apriori(Adult)

## take 2 subsets
r1 <- r[1:10]
r2 <- r[6:15]

union(r1, r2)
intersect(r1, r2)
setequal(r1, r2)
```

size

Getting the Size of Each Element

Description

Provides the generic function `size` and S4 methods to get the size of each element from objects based on `itemMatrix`. For example, it is used to get a vector of transaction sizes (i.e., the number of present items (ones) per element (row) of the binary incidence matrix) from an object of class `transactions`.

Usage

```
size(x, ...)
```

Arguments

`x` an object.
`...` further (unused) arguments.

Value

`size` returns a numeric vector of length `length(x)`. Each element is the size of the corresponding element (row in the matrix) in object `x`. For rules, `size` returns the sum of the number of elements in the LHS and the RHS.

See Also

[itemMatrix-class](#), [transactions-class](#)

Examples

```
data("Adult")
summary(size(Adult))
```

sort

Sorting Associations

Description

Provides the generic function `SORT` and the S4 method to sort elements in class `associations` (e.g., itemsets or rules) according to the value of measures stored in the association's slot `quality` (e.g., support).

Usage

```
## S4 method for signature 'associations':
sort(x, decreasing = TRUE, na.last = NA,
     by = "support")

## deprecated since arules version 0.5-1
## S4 method for signature 'associations':
SORT(x, by = "support", na.last = NA,
     decreasing = TRUE)
```

Arguments

<code>x</code>	an object to be sorted.
<code>decreasing</code>	a logical. Should the sort be increasing or decreasing? (default is decreasing)
<code>na.last</code>	for controlling the treatment of NAs. If <code>TRUE</code> , missing values in the data are put last; if <code>FALSE</code> , they are put first; if <code>NA</code> , they are removed.
<code>by</code>	a character string specifying the quality measure stored in <code>x</code> to be used to sort <code>x</code> .

Value

An object of the same class as `x`.

See Also

[associations-class](#)

Examples

```
data("Adult")

## Mine frequent itemsets with Eclat.
fsets <- eclat(Adult, parameter = list(supp = 0.5))

## Print the 5 itemsets with the highest support as a data.frame.
as(sort(fsets)[1:5], "data.frame")
```

subset

Subsetting Itemsets, Rules and Transactions

Description

Provides the generic function `subset` and S4 methods to subset associations or transactions (item-Matrix) which meet certain conditions (e.g., contains certain items or satisfies a minimum lift).

Usage

```
subset(x, ...)
```

```
## S4 method for signature 'itemMatrix':
subset(x, subset, ...)
```

```
## S4 method for signature 'itemsets':
subset(x, subset, ...)
```

```
## S4 method for signature 'rules':
subset(x, subset, ...)
```

```
## S4 method for signature 'itemMatrix':
subset(x, subset, ...)
```

Arguments

<code>x</code>	object to be subsetted.
<code>subset</code>	logical expression indicating elements to keep.
<code>...</code>	further arguments to be passed to or from other methods.

Details

`subset` works on the rows/itemsets/rules of `x`. The expression given in `subset` will be evaluated using `x`, so the items (`lhs/rhs/items`) and the columns in the quality data.frame can be directly referred to by their names.

Important operators to select itemsets containing items specified by their labels are `%in%` (select itemsets matching *any* given item), `%ain%` (select only itemsets matching *all* given item) and `%pin%` (`%in%` with partial matching).

Value

An object of the same class as `x` containing only the elements which satisfy the conditions.

See Also

[itemMatrix-class](#), [itemsets-class](#), [rules-class](#), [transactions-class](#)

Examples

```
data("Adult")
rules <- apriori(Adult)

## select all rules with item "marital-status=Never-married" in
## the right-hand-side and lift > 2
rules.sub <- subset(rules, subset = rhs %in% "marital-status=Never-married"
  & lift > 2)

## use partial matching for all items corresponding to the variable
## "marital-status"
rules.sub <- subset(rules, subset = rhs %pin% "marital-status=")

## select only rules with items "age=Young" and "workclass=Private" in
## the left-hand-side
rules.sub <- subset(rules, subset = lhs %ain%
  c("age=Young", "workclass=Private"))
```

Description

Provides the generic function and the needed S4 method to count support for given itemsets (and other types of associations) in a given transaction database.

Usage

```

support(x, transactions, ...)
## S4 method for signature 'itemMatrix':
support(x, transactions,
        type= c("relative", "absolute"), control = NULL)
## S4 method for signature 'associations':
support(x, transactions,
        type= c("relative", "absolute"), control = NULL)

```

Arguments

<code>x</code>	the set of itemsets for which support should be counted.
<code>...</code>	further arguments are passed on.
<code>transactions</code>	the transaction data set used for mining.
<code>type</code>	a character string specifying if "relative" support or "absolute" support (counts) are returned for the itemsets in <code>x</code> . (default: "relative")
<code>control</code>	a named list with elements <code>method</code> indicating the method ("tidlists" or "ptree"), and the logical arguments <code>reduce</code> and <code>verbose</code> to indicate if unused items are removed and if the output should be verbose.

Details

Normally, itemset support is counted during mining the database with a set minimum support. However, if only the support information for a single or a few itemsets is needed, one might not want to mine the database for all frequent itemsets.

If in control `method = "ptree"` is used, the counters for the itemsets are organized in a prefix tree. The transactions are sequentially processed and the corresponding counters in the prefix tree are incremented (see Hahsler et al, 2008). This method is used by default since it is typically significantly faster than tid list intersection.

If in control `method = "tidlists"` is used, support is counted using transaction ID list intersection which is used by several fast mining algorithms (e.g., by Eclat). However, Support is determined for each itemset individually which is slow for a large number of long itemsets in dense data.

If in control `reduce = TRUE` is used, unused items are removed from the data before creating rules. This might be slower for large transaction data sets.

Value

A numeric vector of the same length as `x` containing the support values for the sets in `x`.

References

Michael Hahsler, Christian Buchta, and Kurt Hornik. Selective association rule generation. *Computational Statistics*, 23(2):303-315, April 2008.

See Also

[itemMatrix-class](#), [associations-class](#), [transactions-class](#)

Examples

```
data("Income")

## find and some frequent itemsets
itemsets <- eclat(Income)[1:5]

## inspect the support returned by eclat
inspect(itemsets)

## count support in the database
support(items(itemsets), Income)
```

tidLists-class *Class “tidLists” — Transaction ID Lists for Items/Itemsets*

Description

Transaction ID lists contains a set of lists. Each list is associated with an item/itemset and stores the IDs of the transactions which support the item/itemset. `tidLists` uses the class `ngCMatrix` to efficiently store the transaction ID lists as a sparse matrix. Each column in the matrix represents one transaction ID list.

`tidLists` can be used for different purposes. For some operations (e.g., support counting) it is efficient to coerce a `transactions` database into `tidLists` where each list contains the transaction IDs for an item (and the support is given by the length of the list).

The implementation of the Eclat mining algorithm (which uses transaction ID list intersection) can also produce transaction ID lists for the found itemsets as part of the returned `itemsets` object. These lists can then be used for further computation.

Objects from the Class

Objects are created by Eclat if the `eclat` function is called with `tidLists = TRUE` in the `ECparameter` object, and returned as part of the mined `itemsets`. Objects can also be created by coercion from an object of class `transactions` or by calls of the form `new("tidLists", ...)`.

Slots

data: object of class `ngCMatrix`.

itemInfo: a data.frame to store item/itemset labels (see `itemMatrix` class).

transactionInfo: a data.frame with vectors of the same length as the number of transactions. Each vector can hold additional information e.g., store transaction IDs or user IDs for each transaction.

Methods

coerce signature(from = "tidLists", to = "ngCMatrix"); access the sparse matrix representation. In the ngCMatrix each column represents the transaction IDs for one item/itemset.

coerce signature(from = "tidLists", to = "dgCMatrix")

coerce signature(from = "tidLists", to = "list")

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

coerce signature(from = "tidLists", to = "itemMatrix")

coerce signature(from = "tidLists", to = "transactions")

coerce signature(from = "itemMatrix", to = "tidLists")

coerce signature(from = "transactions", to = "tidLists")

dim signature(x = "tidLists"); returns the dimensions of the sparse Matrix representing the tidLists.

itemInfo returns the slot itemInfo.

itemLabels signature(object = "tidLists"); returns the item labels as a character vector.

labels signature(x = "transactions"); returns the labels (item labels and transaction IDs) for the incidence matrix as a list of two vectors named items and transactionID.

show signature(object = "tidLists")

summary signature(object = "tidLists")

transactionInfo signature(x = "transactions"): returns the slot transactionInfo.

See Also

[\[-methods, LIST, eclat, image, length, size, ngCMatrix \(in Matrix\), itemMatrix-class, itemsets-class, transactions-class\]](#)

Examples

```
## Create transaction data set.
data <- list(
  c("a", "b", "c"),
  c("a", "b"),
  c("a", "b", "d"),
  c("b", "e"),
  c("b", "c", "e"),
  c("a", "d", "e"),
  c("a", "c"),
  c("a", "b", "d"),
  c("c", "e"),
  c("a", "b", "d", "e")
)
t <- as(data, "transactions")

## Mine itemsets with tidLists.
```

```
f <- eclat(data, parameter = list(support = 0, tidLists = TRUE))

## Get dimensions of the tidLists.
dim(tidLists(f))

## Coerce tidLists to list.
as(tidLists(f), "list")

## Inspect visually.
image(tidLists(f))
```

transactions-class *Class “transactions” — Binary Incidence Matrix for Transactions*

Description

The `transactions` class represents transaction data used for mining itemsets or rules. It is a direct extension of class `itemMatrix` to store a binary incidence matrix, item labels, and optionally transaction IDs and user IDs.

Objects from the Class

Objects are created by coercion from objects of other classes or by calls of the form `new("transactions", ...)`.

Slots

transactionInfo: a data.frame with vectors of the same length as the number of transactions. Each vector can hold additional information, e.g., store transaction IDs or user IDs for each transaction.

data: object of class `ngCMatrix` to store the binary incidence matrix (see `itemMatrix` class)

itemInfo: a data.frame to store item labels (see `itemMatrix` class)

Extends

Class `itemMatrix`, directly.

Methods

coerce signature(`from` = "matrix", `to` = "transactions"); produces a transactions data set from a binary incidence matrix. The row names are used as item labels and the column names are stores as transaction IDs.

coerce signature(`from` = "list", `to` = "transactions"); produces a transactions data set from a list. The names of the items in the list are used as item labels and the item IDs and the incidence matrix is produced automatically.

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

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

coerce signature(from = "data.frame", to = "transactions"); recodes the data frame containing only categorical variables (all have to be factors) into a binary transaction data set. The needed number of dummy items are automatically generated. The item labels are generated by concatenating variable names and levels with a '='. The variable names and levels are stored in the labels data frame as the components `variables` and `levels`. Note that NA in one of the levels is replaced by the string "NA", i.e. the special meaning is lost.

coerce signature(from = "transactions", to = "data.frame"); represents the set of transactions in a printable form as a data.frame. Note that this does not reverse coercion from data.frame to transactions.

labels signature(x = "transactions"); returns the labels (item labels and transaction IDs) for the incidence matrix as a list of two vectors named `items` and `transactionID`.

transactionInfo<- signature(x = "transactions"); replaces the transactionInfo data frame

transactionInfo signature(x = "transactions"); returns transactionInfo

show signature(object = "transactions")

summary signature(object = "transactions")

See Also

[\[-methods, LIST, WRITE, c, image, inspect, read.transactions, random.transactions, sets, itemMatrix-class\]](#)

Examples

```
## 1. example: creating transactions form a list
a_list <- list(
  c("a", "b", "c"),
  c("a", "b"),
  c("a", "b", "d"),
  c("c", "e"),
  c("a", "b", "d", "e")
)

## set transaction names
names(a_list) <- paste("Tr", c(1:5), sep = "")
a_list

## coerce into transactions
trans <- as(a_list, "transactions")

## analyze transactions
summary(trans)
image(trans)

## 2. example: creating transactions from a matrix
a_matrix <- matrix(
  c(1, 1, 1, 0, 0,
    1, 1, 0, 0, 0,
```

```

      1,1,0,1,0,
      0,0,1,0,1,
      1,1,0,1,1), ncol = 5)

## set dim names
dimnames(a_matrix) <- list(
  c("a", "b", "c", "d", "e"),
  paste("Tr", c(1:5), sep = ""))

a_matrix

## coerce
trans2 <- as(a_matrix, "transactions")
trans2

## example 3: creating transactions from data.frame
a_data.frame <- data.frame(
  age = as.factor(c(6,8,7,6,9,5)),
  grade = as.factor(c(1,3,1,1,4,1)))
## note: all attributes have to be factors
a_data.frame

## coerce
trans3 <- as(a_data.frame, "transactions")
image(trans3)

## 3. example creating from data.frame with NA
a_df <- sample(c(LETTERS[1:5], NA), 10, TRUE)
a_df <- data.frame(X = a_df, Y = sample(a_df))

a_df

trans3 <- as(a_df, "transactions")
trans3
as(trans3, "data.frame")

```

unique

Remove Duplicated Elements from a Collection

Description

Provides the generic function `unique` and the S4 methods for `itemMatrix`. `unique` uses [duplicated](#) to return an `itemMatrix` with the duplicate elements removed.

Note that `unique` can also be used on collections of associations.

Usage

```
unique(x, incomparables = FALSE, ...)
```

Arguments

`x` an object of class `itemMatrix` or `associations`.
`...` further arguments (currently unused).
`incomparables` currently unused.

Value

An object of the same class as `x` with duplicated elements removed.

See Also

[duplicated](#), [associations-class](#), [itemMatrix-class](#)

Examples

```
data("Adult")

r1 <- apriori(Adult[1:1000], parameter = list(support = 0.5))
r2 <- apriori(Adult[1001:2000], parameter = list(support = 0.5))

## Note that this produces a collection of rules from two sets
r_comb <- c(r1, r2)
r_comb <- unique(r_comb)
r_comb
```

WRITE

Writes transactions or associations to disk

Description

Provides the generic function `WRITE` and the S4 methods to write transactions or associations (itemsets, rules) to disk.

Usage

```
WRITE(x, file = "", ...)
```

Arguments

`x` the transactions or associations (rules, itemsets, etc.) object.
`file` either a character string naming a file or a connection open for writing. `""` indicates output to the console.
`...` further arguments passed on to [write.table](#).

Details

WRITE first uses coercion to data.frame to obtain a printable form of `x` and then uses `write.table` to write the data to disk.

Note: To save and load associations in compact form, use `save` and `load` from the **base** package. Alternatively, association can be written to disk in PMML (Predictive Model Markup Language). This requires the packages **pmml** and **XML**. See Examples section for usage.

See Also

`write.table` (in **base**), `transactions-class`, `associations-class`

Examples

```
data("Epub")

## write the formatted result to screen
WRITE(head(Epub))

## write the formatted result to file in CSV format
WRITE(Epub, file = "data.csv", sep = ",", col.names = NA)

## write rules in CSV format
rules <- apriori(Epub, parameter=list(support=0.002, conf=0.8))
WRITE(rules, file = "data.csv", sep = ",", col.names = NA)

unlink("data.csv") # tidy up

## write rules as PMML
library(pmml)
rules_pmml <- pmml(rules)
saveXML(rules_pmml, "data.xml")

unlink("data.xml") # tidy up
```

[-methods

Methods for "[": Extraction or Subsetting in Package 'arules'

Description

Methods for "[", i.e., extraction or subsetting in package **arules**. Subsetting can be done by integers containing column/row numbers, vectors of logicals or strings containing parts of item labels.

Methods

[signature(`x = "itemMatrix"`, `i = "ANY"`, `j = "ANY"`, `drop = "ANY"`); extracts parts of an `itemMatrix`. The first argument selects rows (e.g., transactions or rules) and the second argument selects columns (items). Either argument can be omitted to select all rows or columns.

- [signature(x = "itemsets", i = "ANY", j = "ANY", drop= "ANY"); extracts a subset of itemsets and the associated quality measures. j has to be missing.
- [signature(x = "rules", i = "ANY", j = "ANY", drop= "ANY"); extracts a subset of rules and the associated quality measures. j has to be missing.
- [signature(x = "transactions", i = "ANY", j = "ANY", drop= "ANY"); extracts a subset of transactions/items from a transactions object (a binary incidence matrix). i and j can be numeric where i selects transactions and j selects items.
- [signature(x = "tidLists", i = "ANY", j = "ANY", drop= "ANY"); extracts parts (transaction ID vectors) from tidLists. i selects the items or itemsets and j selects transactions in the lists.

See Also

[itemMatrix-class](#), [itemsets-class](#), [rules-class](#), [transactions-class](#), [tidLists-class](#)

Examples

```
data(Adult)
Adult

## select first 10 transactions
Adult[1:10]

## select first 10 items for first 100 transactions
Adult[1:100, 1:10]

## select the first 100 transactions for the items containing
## "income" or "age=Young" in their labels
Adult[1:100, c("income=small", "income=large" ,"age=Young")]
```

Index

- *Topic **arith**
 - sort, 51
- *Topic **array**
 - [*-methods*], 61
- *Topic **attribute**
 - length, 37
 - size, 50
- *Topic **classes**
 - APappearance-class, 5
 - AScontrol-classes, 7
 - ASparameter-classes, 9
 - associations-class, 11
 - itemMatrix-class, 33
 - itemsets-class, 36
 - proximity-classes, 41
 - rules-class, 47
 - tidLists-class, 55
 - transactions-class, 57
- *Topic **cluster**
 - affinity, 4
 - dissimilarity, 14
 - predict, 40
- *Topic **datagen**
 - random.transactions, 42
- *Topic **datasets**
 - Adult, 2
 - Epub, 19
 - Groceries, 19
 - Income, 21
- *Topic **file**
 - read.transactions, 44
 - WRITE, 60
- *Topic **hplot**
 - image, 20
 - itemFrequencyPlot, 32
- *Topic **manip**
 - combine, 12
 - duplicated, 16
 - is.superset, 28
 - itemCoding, 29
 - LIST, 38
 - match, 39
 - sample, 48
 - sets, 49
 - sort, 51
 - subset, 52
 - unique, 59
- *Topic **models**
 - affinity, 4
 - apriori, 6
 - coverage, 13
 - crossTable, 14
 - dissimilarity, 14
 - eclat, 17
 - interestMeasure, 23
 - is.closed, 27
 - is.maximal, 27
 - itemFrequency, 31
 - predict, 40
 - ruleInduction, 45
 - support, 53
- *Topic **print**
 - inspect, 23
 - [, Matrix, ANY, ANY, ANY-method
(*-methods*), 61
 - [, Matrix, lMatrix, missing, ANY-method
(*-methods*), 61
 - [, Matrix, logical, missing, ANY-method
(*-methods*), 61
 - [, itemMatrix, ANY, ANY, ANY-method
(*-methods*), 61
 - [, itemMatrix-method (*-methods*),
61
 - [, itemsets, ANY, ANY, ANY-method
(*-methods*), 61
 - [, itemsets-method (*-methods*), 61
 - [, rules, ANY, ANY, ANY-method
(*-methods*), 61

- [, rules-method (*[-methods]*), 61
- [, tidLists, ANY, ANY, ANY-method (*[-methods]*), 61
- [, tidLists-method (*[-methods]*), 61
- [, transactions, ANY, ANY, ANY-method (*[-methods]*), 61
- [, transactions-method (*[-methods]*), 61
- [-methods, 35, 37, 48, 56, 58
- [-methods, 61
- [<-, Matrix, ANY, ANY, ANY-method (*[-methods]*), 61
- [<-, Matrix, missing, missing, ANY-method (*[-methods]*), 61
- %ain% (*itemMatrix-class*), 33
- %ain%, *itemMatrix*, character-method (*itemMatrix-class*), 33
- %in% (*match*), 39
- %in%, *itemMatrix*, character-method (*itemMatrix-class*), 33
- %pin% (*itemMatrix-class*), 33
- %pin%, *itemMatrix*, character-method (*itemMatrix-class*), 33

- Adult, 2
- AdultUCI (*Adult*), 2
- affinity, 4, 15, 16, 41, 42
- affinity, *itemMatrix*-method (*affinity*), 4
- affinity, matrix-method (*affinity*), 4
- APappearance, 6
- APappearance-class, 7
- APappearance-class, 5
- APcontrol, 6
- APcontrol-class, 7
- APcontrol-class (*AScontrol-classes*), 7
- APparameter, 6
- APparameter-class, 7
- APparameter-class (*ASparameter-classes*), 9
- apriori, 5, 6, 6, 8–10, 18, 36, 37, 47, 48
- ar_cross_dissimilarity-class (*proximity-classes*), 41
- ar_similarity-class, 5
- ar_similarity-class (*proximity-classes*), 41

- AScontrol-class (*AScontrol-classes*), 7
- AScontrol-classes, 7
- ASparameter-class (*ASparameter-classes*), 9
- ASparameter-classes, 9
- associations, 14, 36, 38, 47, 51
- associations-class, 16, 29, 30, 37, 48–50, 52, 55, 60, 61
- associations-class, 11

- barplot, 32
- c, 35, 37, 48, 58
- c (*combine*), 12
- c, *itemMatrix*-method (*combine*), 12
- c, *itemsets*-method (*combine*), 12
- c, rules-method (*combine*), 12
- c, transactions-method (*combine*), 12
- coerce, data.frame, transactions-method (*transactions-class*), 57
- coerce, *itemMatrix*, dgCMatrix-method (*itemMatrix-class*), 33
- coerce, *itemMatrix*, list-method, 39
- coerce, *itemMatrix*, list-method (*itemMatrix-class*), 33
- coerce, *itemMatrix*, matrix-method (*itemMatrix-class*), 33
- coerce, *itemMatrix*, ngCMatrix-method (*itemMatrix-class*), 33
- coerce, *itemMatrix*, tidLists-method (*tidLists-class*), 55
- coerce, *itemsets*, data.frame-method (*itemsets-class*), 36
- coerce, list, APappearance-method (*APappearance-class*), 5
- coerce, list, APcontrol-method (*AScontrol-classes*), 7
- coerce, list, APparameter-method (*ASparameter-classes*), 9
- coerce, list, ECcontrol-method (*AScontrol-classes*), 7
- coerce, list, ECparameter-method (*ASparameter-classes*), 9
- coerce, list, *itemMatrix*-method (*itemMatrix-class*), 33
- coerce, list, transactions-method (*transactions-class*), 57

- coerce, matrix, itemMatrix-method
(*itemMatrix-class*), 33
- coerce, matrix, transactions-method
(*transactions-class*), 57
- coerce, ngCMatrix, list-method
(*LIST*), 38
- coerce, NULL, APappearance-method
(*APappearance-class*), 5
- coerce, NULL, APcontrol-method
(*AScontrol-classes*), 7
- coerce, NULL, APparameter-method
(*ASparameter-classes*), 9
- coerce, NULL, ECcontrol-method
(*AScontrol-classes*), 7
- coerce, NULL, ECparameter-method
(*ASparameter-classes*), 9
- coerce, rules, data.frame-method
(*rules-class*), 47
- coerce, tidLists, dgCMatrix-method
(*tidLists-class*), 55
- coerce, tidLists, itemMatrix-method
(*tidLists-class*), 55
- coerce, tidLists, list-method
(*tidLists-class*), 55
- coerce, tidLists, matrix-method
(*tidLists-class*), 55
- coerce, tidLists, ngCMatrix-method
(*tidLists-class*), 55
- coerce, tidLists, transactions-method
(*tidLists-class*), 55
- coerce, transactions, data.frame-method
(*transactions-class*), 57
- coerce, transactions, list-method
(*transactions-class*), 57
- coerce, transactions, matrix-method
(*transactions-class*), 57
- coerce, transactions, tidLists-method
(*tidLists-class*), 55
- combine, 12
- coverage, 13
- coverage, rules-method (*coverage*),
13
- crossTable, 14
- crossTable, itemMatrix-method
(*crossTable*), 14
- decode, 39
- decode (*itemCoding*), 29
- decode, list-method (*itemCoding*),
29
- decode, numeric-method
(*itemCoding*), 29
- dim, itemMatrix-method
(*itemMatrix-class*), 33
- dim, tidLists-method
(*tidLists-class*), 55
- dissimilarity, 5, 14, 41, 42
- dissimilarity, associations-method
(*dissimilarity*), 14
- dissimilarity, itemMatrix-method
(*dissimilarity*), 14
- dissimilarity, matrix-method
(*dissimilarity*), 14
- dist, 42
- dist-class, 16
- dist-class (*proximity-classes*), 41
- duplicated, 16, 35, 37, 48, 59, 60
- duplicated, itemMatrix-method
(*duplicated*), 16
- duplicated, itemsets-method
(*duplicated*), 16
- duplicated, rules-method
(*duplicated*), 16
- ECcontrol, 18
- ECcontrol-class, 18
- ECcontrol-class
(*AScontrol-classes*), 7
- eclat, 5, 8–10, 17, 36, 37, 55, 56
- ECparameter, 18, 55
- ECparameter-class, 18
- ECparameter-class
(*ASparameter-classes*), 9
- encode (*itemCoding*), 29
- encode, character-method
(*itemCoding*), 29
- encode, list-method (*itemCoding*),
29
- encode, numeric-method
(*itemCoding*), 29
- Epub, 19
- generatingItemsets (*rules-class*),
47
- generatingItemsets, rules-method
(*rules-class*), 47
- Groceries, 19

- image, 20, 20, 35, 56, 58
- image, itemMatrix-method (*image*), 20
- image, tidLists-method (*image*), 20
- image, transactions-method (*image*), 20
- Income, 21
- IncomeESL (*Income*), 21
- info (*associations-class*), 11
- info, associations-method (*associations-class*), 11
- info<- (*associations-class*), 11
- info<-, associations-method (*associations-class*), 11
- initialize, AParameter-method (*AParameter-classes*), 9
- initialize, AScontrol-method (*AScontrol-classes*), 7
- initialize, ASparameter-method (*ASparameter-classes*), 9
- inspect, 23, 35, 37, 48, 58
- inspect, itemMatrix-method (*inspect*), 23
- inspect, itemsets-method (*inspect*), 23
- inspect, rules-method (*inspect*), 23
- inspect, transactions-method (*inspect*), 23
- interestMeasure, 23
- interestMeasure, itemsets-method (*interestMeasure*), 23
- interestMeasure, rules-method (*interestMeasure*), 23
- intersect (*sets*), 49
- intersect, ANY-method (*sets*), 49
- intersect-methods (*sets*), 49
- is.closed, 27
- is.closed, itemsets-method (*is.closed*), 27
- is.element (*sets*), 49
- is.element, ANY-method (*sets*), 49
- is.element-methods (*sets*), 49
- is.maximal, 27, 37
- is.maximal, itemMatrix-method (*is.maximal*), 27
- is.maximal, itemsets-method (*is.maximal*), 27
- is.subset, 11, 35
- is.subset (*is.superset*), 28
- is.subset, associations-method (*is.superset*), 28
- is.subset, itemMatrix-method (*is.superset*), 28
- is.superset, 11, 28, 28, 35
- is.superset, associations-method (*is.superset*), 28
- is.superset, itemMatrix-method (*is.superset*), 28
- itemCoding, 29
- itemFrequency, 31, 33, 35
- itemFrequency, itemMatrix-method (*itemFrequency*), 31
- itemFrequency, tidLists-method (*itemFrequency*), 31
- itemFrequencyPlot, 31, 32, 35
- itemFrequencyPlot, itemMatrix-method (*itemFrequencyPlot*), 32
- itemInfo (*itemMatrix-class*), 33
- itemInfo, itemMatrix-method (*itemMatrix-class*), 33
- itemInfo, itemsets-method (*itemsets-class*), 36
- itemInfo, rules-method (*rules-class*), 47
- itemInfo, tidLists-method (*tidLists-class*), 55
- itemInfo<- (*itemMatrix-class*), 33
- itemInfo<-, itemMatrix-method (*itemMatrix-class*), 33
- itemLabels (*itemMatrix-class*), 33
- itemLabels, itemMatrix-method (*itemMatrix-class*), 33
- itemLabels, itemsets-method (*itemsets-class*), 36
- itemLabels, rules-method (*rules-class*), 47
- itemLabels, tidLists-method (*tidLists-class*), 55
- itemLabels<- (*itemMatrix-class*), 33
- itemLabels<-, itemMatrix-method (*itemMatrix-class*), 33
- itemMatrix, 12, 20, 31, 32, 36, 38, 39, 47, 48, 50, 55, 57
- itemMatrix-class, 5, 11, 12, 14, 16, 17, 20, 23, 28–31, 33, 39–41, 48–51, 53,

- 55, 56, 58, 60, 62
- itemMatrix-class, 33
- items (*itemsets-class*), 36
- items, associations-method (*associations-class*), 11
- items, itemsets-method (*itemsets-class*), 36
- items, rules-method (*rules-class*), 47
- items<- (*itemsets-class*), 36
- items<-, itemsets-method (*itemsets-class*), 36
- itemsetInfo (*itemMatrix-class*), 33
- itemsetInfo, itemMatrix-method (*itemMatrix-class*), 33
- itemsetInfo<- (*itemMatrix-class*), 33
- itemsetInfo<-, itemMatrix-method (*itemMatrix-class*), 33
- itemsets, 7, 12, 18, 32, 38, 55
- itemsets-class, 7, 11, 12, 17, 18, 23, 26–28, 35, 40, 46, 53, 56, 62
- itemsets-class, 36
- labels (*itemMatrix-class*), 33
- labels, associations-method (*associations-class*), 11
- labels, itemMatrix-method (*itemMatrix-class*), 33
- labels, itemsets-method (*itemsets-class*), 36
- labels, rules-method (*rules-class*), 47
- labels, tidLists-method (*tidLists-class*), 55
- labels, transactions-method (*transactions-class*), 57
- length, 11, 35, 37, 37, 48, 56
- length, associations-method (*associations-class*), 11
- length, itemMatrix-method (*length*), 37
- length, itemsets-method (*length*), 37
- length, rules-method (*length*), 37
- length, tidLists-method (*length*), 37
- levelplot, 20
- lhs (*rules-class*), 47
- lhs, rules-method (*rules-class*), 47
- lhs<- (*rules-class*), 47
- lhs<-, rules-method (*rules-class*), 47
- LIST, 29, 30, 35, 38, 56, 58
- LIST, itemMatrix-method (*LIST*), 38
- LIST, tidLists-method (*LIST*), 38
- LIST, transactions-method (*LIST*), 38
- match, 34, 35, 37, 39, 48
- match, itemMatrix, itemMatrix-method (*match*), 39
- match, itemsets, itemsets-method (*match*), 39
- match, rules, rules-method (*match*), 39
- ngCMatrix, 34, 55–57
- ngCMatrix-class, 35
- nitems (*itemMatrix-class*), 33
- nitems, itemMatrix-method (*itemMatrix-class*), 33
- predict, 40
- predict, itemMatrix-method (*predict*), 40
- print, summary.itemMatrix-method (*itemMatrix-class*), 33
- proximity-classes, 41
- quality (*associations-class*), 11
- quality, associations-method (*associations-class*), 11
- quality<- (*associations-class*), 11
- quality<-, associations-method (*associations-class*), 11
- random.patterns (*random.transactions*), 42
- random.transactions, 42, 58
- read.transactions, 44, 58
- recode (*itemCoding*), 29
- recode, itemMatrix-method (*itemCoding*), 29
- rhs (*rules-class*), 47
- rhs, rules-method (*rules-class*), 47
- rhs<- (*rules-class*), 47
- rhs<-, rules-method (*rules-class*), 47

- ruleInduction, 45
- ruleInduction, itemsets-method
(*ruleInduction*), 45
- rules, 7, 12, 32, 38
- rules-class, 7, 11–13, 17, 23, 26, 35, 40,
46, 53, 62
- rules-class, 47
- sample, 48
- sample, associations-method
(*sample*), 48
- sample, itemMatrix-method
(*sample*), 48
- setdiff(*sets*), 49
- setdiff, ANY-method(*sets*), 49
- setdiff-methods(*sets*), 49
- setequal(*sets*), 49
- setequal, ANY-method(*sets*), 49
- setequal-methods(*sets*), 49
- sets, 11, 35, 37, 48, 49, 58
- show, AParameter-method
(*AParameter-classes*), 9
- show, ASControl-method
(*ASControl-classes*), 7
- show, ASparameter-method
(*ASparameter-classes*), 9
- show, associations-method
(*associations-class*), 11
- show, itemMatrix-method
(*itemMatrix-class*), 33
- show, summary.itemMatrix-method
(*itemMatrix-class*), 33
- show, summary.itemsets-method
(*itemsets-class*), 36
- show, summary.rules-method
(*rules-class*), 47
- show, summary.tidLists-method
(*tidLists-class*), 55
- show, summary.transactions-method
(*transactions-class*), 57
- show, tidLists-method
(*tidLists-class*), 55
- show, transactions-method
(*transactions-class*), 57
- size, 37, 48, 50, 56
- size, itemMatrix-method(*size*), 50
- size, itemsets-method(*size*), 50
- size, rules-method(*size*), 50
- size, tidLists-method(*size*), 50
- SORT, 11
- SORT(*sort*), 51
- sort, 51
- SORT, associations-method(*sort*),
51
- sort, associations-method(*sort*),
51
- subset, 35, 37, 48, 52
- subset, itemMatrix-method
(*subset*), 52
- subset, itemsets-method(*subset*),
52
- subset, rules-method(*subset*), 52
- summary, itemMatrix-method
(*itemMatrix-class*), 33
- summary, itemsets-method
(*itemsets-class*), 36
- summary, rules-method
(*rules-class*), 47
- summary, tidLists-method
(*tidLists-class*), 55
- summary, transactions-method
(*transactions-class*), 57
- summary.associations-class
(*associations-class*), 11
- summary.itemMatrix-class
(*itemMatrix-class*), 33
- summary.itemsets-class
(*itemsets-class*), 36
- summary.rules-class
(*rules-class*), 47
- summary.tidLists-class
(*tidLists-class*), 55
- summary.transactions-class
(*transactions-class*), 57
- support, 53
- support, associations-method
(*support*), 53
- support, itemMatrix-method
(*support*), 53
- t, associations-method
(*associations-class*), 11
- t, ngCMatrix-method
(*itemMatrix-class*), 33
- t, tidLists-method
(*tidLists-class*), 55
- t, transactions-method
(*transactions-class*), 57

tidLists, 36, 38
tidLists (*itemsets-class*), 36
tidLists, *itemsets-method*
 (*itemsets-class*), 36
tidLists-class, 20, 37, 62
tidLists-class, 55
tidLists_or_NULL-class
 (*tidLists-class*), 55
transactionInfo
 (*transactions-class*), 57
transactionInfo, *tidLists-method*
 (*tidLists-class*), 55
transactionInfo, *transactions-method*
 (*transactions-class*), 57
transactionInfo<-
 (*transactions-class*), 57
transactionInfo<- , *transactions-method*
 (*transactions-class*), 57
transactions, 2, 4, 6, 14, 18, 19, 21, 31,
 32, 38, 42–45, 50, 55
transactions-class, 7, 12, 14, 18, 20,
 23, 31, 35, 43, 45, 46, 49, 51, 53, 55,
 56, 61, 62
transactions-class, 57

union, 12
union (*sets*), 49
union, *ANY-method (sets)*, 49
union-methods (*sets*), 49
unique, 11, 17, 35, 36, 47, 59
unique, *associations-method*
 (*unique*), 59
unique, *itemMatrix-method*
 (*unique*), 59

WRITE, 11, 58, 60
WRITE, *associations-method*
 (*WRITE*), 60
WRITE, *transactions-method*
 (*WRITE*), 60
write.table, 60, 61