Package 'arules'

November 19, 2021

Version 1.7-1

Date 2021-11-18

Title Mining Association Rules and Frequent Itemsets

Description Provides the infrastructure for representing,
manipulating and analyzing transaction data and patterns (frequent
itemsets and association rules). Also provides
C implementations of the association mining algorithms Apriori and Eclat.
Hahsler, Gruen and Hornik (2005) <doi:10.18637/jss.v014.i15>.

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

URL https://github.com/mhahsler/arules

BugReports https://github.com/mhahsler/arules/issues

Depends R (>= 3.4.0), Matrix (>= 1.2-0)

Imports stats, methods, generics, graphics, utils

Suggests pmml, XML, arulesViz, arulesCBA, testthat

License GPL-3

Copyright The source code for Apriori and Eclat was obtained from
http://www.borgelt.net/ and is Copyright (C) 1996-2003
Christian Borgelt. All other code is Copyright (C) Michael
Hahsler, Christian Buchta, Bettina Gruen and Kurt Hornik.

NeedsCompilation yes

Author Michael Hahsler [aut, cre, cph],
Christian Buchta [aut, cph],
Bettina Gruen [aut, cph],
Kurt Hornik [aut, cph],
Ian Johnson [ctb, cph],
Christian Borgelt [ctb, cph]

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

Repository CRAN

Date/Publication 2021-11-19 07:40:01 UTC

R topics documented:

`abbreviate` *Abbreviate function for item labels in transactions, itemMatrix and associations*

Description

Provides the generic function and the methods to abbreviate long item labels in transactions, associations (rules and itemsets) and transaction ID lists. Note that `abbreviate` is not a generic and this **arules** defines a generic with the R's abbreviate as the default.

Usage

```
abbreviate(names.arg, ...)
## S4 method for signature 'itemMatrix'
abbreviate(names.arg, minlength = 4, ..., method = "both.sides")
## S4 method for signature 'rules'
abbreviate(names.arg, minlength = 4, ..., method = "both.sides")
## S4 method for signature 'itemsets'
abbreviate(names.arg, minlength = 4, ..., method = "both.sides")
## S4 method for signature 'transactions'
abbreviate(names.arg, minlength = 4, ..., method = "both.sides")
## S4 method for signature 'tidLists'
abbreviate(names.arg, minlength = 4, ..., method = "both.sides")
```

Arguments

names.arg	an object of class "transactions", "itemMatrix", "itemsets", "rules" or "tidLists".
minlength	number of characters allowed in abbreviation
method	apply to level and value (both.sides)
...	further arguments passed on to the default abbreviation function.

Author(s)

Sudheer Chelluboina and Michael Hahsler based on code by Martin Vodenicharov.

See Also

abbreviate in base.

Examples

```
data(Adult)
inspect(head(Adult, 1))

Adult_abbr <- abbreviate(Adult, 15)
inspect(head(Adult_abbr, 1))
```

---

addComplement *Add Complement-items to Transactions*

Description

Provides the generic function addComplement and the S4 methods for transactions. This function adds for given items complement items. That is it adds an artificial item to each transactions which does not contain the original item. Such items are also called negative items (Antonie et al, 2014).

Usage

```
addComplement(x, labels, complementLabels=NULL)
```

Arguments

x	an object of class transactions.
labels	character strings; item labels for which complements should be created.
complementLabels	character strings; labels for the artificial complement-items. If omitted then the original label is prepended by "!" to form the complement-item label.

Value

Returns an object of class transactions with complement-items added.

Author(s)

Michael Hahsler

References

Antonie L., Li J., Zaiane O. (2014) Negative Association Rules. In: Aggarwal C., Han J. (eds) *Frequent Pattern Mining,* Springer International Publishing, pp. 135-145. doi: 10.1007/9783319-078212_6

See Also

transactions-class, merge

Examples

```
data("Groceries")

## add a complement-items for "whole milk" and "other vegetables"
g2 <- addComplement(Groceries, c("whole milk", "other vegetables"))
g2
tail(itemInfo(g2))
inspect(head(g2, 3))

## use a custom label for the complement-item
g3 <- addComplement(g2, "coffee", complementLabels = "NO coffee")
inspect(head(g2, 3))

## add complements for all items (this is excessive for this dataset)
g4 <- addComplement(Groceries, itemLabels(Groceries))
g4

## add complements for all items with a minimum support of 0.1
g5 <- addComplement(Groceries, names(which(itemFrequency(Groceries) >= 0.1)))
g5
```

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, Trinadad&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 https://www.census.gov/ 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).

Author(s)

Michael Hahsler

Source

https://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 <- transactions(AdultUCI)
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

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

where $sup(.)$ 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`.

Author(s)

Michael Hahsler

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)
image(a)
```

`APappearance-class` *Class APappearance — Specifying the appearance Argument of Apriori to Implement Rule Templates*

Description

Specifies the restrictions on the associations mined by `apriori`. These restrictions can implement certain aspects of rule templates described by Klemettinen (1994).

Note that appearance is only supported by the implementation of `apriori`.

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:

`lhs`, `rhs`, `both`, `items`, `none`: character vectors giving the labels of the items which can appear in the specified place (rhs, lhs or both for rules and items for itemsets). none specifies, that the items mentioned there cannot appear anywhere in the rule/itemset. Note that items cannot be specified in more than one place (i.e., you cannot specify an item in lhs and rhs, but have to specify it as both).

`default`: one of `"both"`, `"lhs"`, `"rhs"`, `"none"`. Specified the default appearance for all items not explicitly mentioned in the other elements of the list. Leave unspecified and the code will guess the correct setting.

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

Author(s)

Michael Hahsler and Bettina Gruen

References

Christian Borgelt (2004) *Apriori — Finding Association Rules/Hyperedges with the Apriori Algorithm*. https://borgelt.net/apriori.html

M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen and A. I. Verkamo (1994). Finding Interesting Rules from Large Sets of Discovered Association Rules. In *Proceedings of the Third International Conference on Information and Knowledge Management,* 401–407.

See Also

apriori

Examples

```
data("Adult")

## find only frequent itemsets which do not contain small or large income
is <- apriori(Adult, parameter = list(support= 0.1, target="frequent"),
  appearance = list(none = c("income=small", "income=large")))
itemFrequency(items(is))["income=small"]
itemFrequency(items(is))["income=large"]

## find itemsets that only contain small or large income, or young age
is <- apriori(Adult, parameter = list(support= 0.1, target="frequent"),
  appearance = list(items = c("income=small", "income=large", "age=Young")))
inspect(head(is))

## find only rules with income-related variables in the right-hand-side.
incomeItems <- grep("^income=", itemLabels(Adult), value = TRUE)
incomeItems
rules <- apriori(Adult, parameter = list(support=0.2, confidence = 0.5),
  appearance = list(rhs = incomeItems))
inspect(head(rules))

## Note: For more complicated restrictions you have to mine all rules/itemsets and
## then filter the results afterwards.
```

---

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 used C implementation of Apriori by Christian Borgelt 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 a data.frame).
parameter	object of class APparameter or named list. The default behavior is to mine rules with minimum support of 0.1, minimum confidence of 0.8, maximum of 10 items (maxlen), and a maximal time for subset checking of 5 seconds (maxtime).
appearance	object of class APappearance or named list. With this argument item appearance can be restricted (implements rule templates). By default all items can appear unrestricted.
control	object of class APcontrol or named list. Controls the algorithmic performance of the mining algorithm (item sorting, report progress (verbose), etc.)
...	Additional arguments are added to the parameter list.

Details

Warning about automatic conversion of matrices or data.frames to transactions. It is preferred to create transactions manually before calling apriori to have control over item coding. This is especially important when you are working with multiple datasets or several subsets of the same dataset. To read about item coding, see itemCoding.

If a data.frame is specified as x, then the data is automatically converted into transactions by discretizing numeric data using discretizeDF and then coercion to transactions. The discretization may fail if the data is not well behaved. Consult the manual page for discretizeDF for details.

Apriori only creates rules with one item in the RHS (Consequent)! The default value in APparameter for minlen is 1. This means that rules with only one item (i.e., an empty antecedent/LHS) like

$$\{\} => \{beer\}$$

will be created. These rules mean that no matter what other items are involved, the item in the RHS will appear with the probability given by the rule's confidence (which equals the support). If you want to avoid these rules then use the argument parameter=list(minlen=2).

Notes on run time and memory usage: If the minimum support is chosen too low for the dataset, then the algorithm will try to create an extremely large set of itemsets/rules. This will result in very long run time and eventually the process will run out of memory. To prevent this, the default maximal length of itemsets/rules is restricted to 10 items (via the parameter element maxlen=10) and the time for checking subsets is limited to 5 seconds (via maxtime=5). The output will show if you hit these limits in the "checking subsets" line of the output. The time limit is only checked when the subset size increases, so it may run significantly longer than what you specify in maxtime. Setting maxtime=0 disables the time limit.

Interrupting execution with Control-C/Esc is not recommended. Memory cleanup will be prevented resulting in a memory leak. Also, interrupts are only checked when the subset size increases, so it may take some time till the execution actually stops.

Value

Returns an object of class rules or itemsets.

Author(s)

Michael Hahsler and Bettina Gruen

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. doi: 10.1145/170035.170072

Christian Borgelt (2012) Frequent Item Set Mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2(6):437-456. J. Wiley & Sons, Chichester, United Kingdom 2012. doi: 10.1002/widm.1074

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).

APRIORI Implementation: https://borgelt.net/apriori.html

See Also

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

Examples

```
## Example 1: Create transaction data and mine association rules
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

## Use the constructor to create transactions
trans1 <- transactions(a_list)
trans1

rules <- apriori(trans1)
inspect(rules)
```

```
## Example 2: Mine association rules from an existing transactions dataset
##   using different minimum support and minimum confidence thresholds
data("Adult")

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")`

Author(s)

Michael Hahsler and Bettina Gruen

References

Christian Borgelt (2004) *Apriori — Finding Association Rules/Hyperedges with the Apriori Algorithm*. https://borgelt.net/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 item)

`maxlen`: an integer value for the maximal number of items per item set (default: 10 items)

`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; use `ruleInduction` for `eclat`.)
- `"hyperedgesets"` (only available for Apriori; see references for the definition of association hyperedgesets)

`ext`: a logical indicating whether to report coverage (i.e., LHS-support) as an extended quality measure (default: `TRUE`)

Additional slots for Apriori in `APparameter`:

`confidence`: a numeric value for the minimal confidence of rules/association hyperedges (default: 0.8). For frequent itemsets it is set to `NA`.

`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

Note: The measure is only reported if `aval` is set to `TRUE`. Use `minval` to set minimum thresholds on the measures.

`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 the original definition of minimum support (support of the LHS and RHS of the rule). If set to `FALSE` then a minimum threshold on coverage (i.e., the support of the LHS) is used instead. (default: `TRUE`)

`maxtime`: Time limit in seconds for checking subsets. `maxtime = 0` disables the time limit. (default: 5 seconds)

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")`

Author(s)

Michael Hahsler and Bettina Gruen

References

Christian Borgelt (2004) *Apriori — Finding Association Rules/Hyperedges with the Apriori Algorithm*. https://borgelt.net/apriori.html

See Also

`apriori`, `eclat`, `weclat` (for weighted rule mining), `ruleInduction`

---

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.

Details

The implementations of `associations` store itemsets (e.g., the LHS and RHS of a rule) as objects of class `itemMatrix` (i.e., sparse binary matrices). Quality measures (e.g., support) are stored in a data.frame accessible via method `quality`.

Associations can store multisets with duplicated elements. Duplicated elements can result from combining several sets of associations. Use `unique` to remove duplicate 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 a 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`

Author(s)

Michael Hahsler

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

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

Value

An object of the same class as `x`.

Author(s)

Michael Hahsler

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 (can contain the same rule multiple times)
r1 <- apriori(Adult[1:1000])
r2 <- apriori(Adult[1001:2000])
rComb <- c(r1, r2)
rComb

## union of rules (a set with only unique rules: same as unique(rComb))
rUnion <- union(r1,r2)
rUnion
```

---

confint *Confidence Intervals for Association Interest Measures*

Description

Computes confidence intervals for interest Measures used in association rule mining.

Usage

```
## S3 method for class 'rules'
confint(object, parm = "oddsRatio", level = 0.95,
  measure = NULL, side = c("two.sided", "lower", "upper"), method = NULL,
  replications = 1000, smoothCounts = 0, transactions = NULL, ...)
```

Arguments

`object`	an object of class rules.
`parm`, `measure`	name of the interest measures (i.e., parameter). `measure` can be used instead of `parm`.
`level`	the confidence level required.
`side`	Should a two-sided confidence interval or a one-sided limit be returned? Lower returns an interval with only a lower limit and upper returns an interval with only an upper limit.
`method`	method to construct the confidence interval. The available methods depends on the measure and the most common method is used by default.
`smoothCounts`	pseudo count for addaptive smoothing (Laplace smoothing). Often a pseudo counts of .5 is used for smoothing (see Detail Section).

replications	number of replications for method "simulation". Ignored for other methods.
transactions	if the rules object does not contain sufficient quality information, then a set of transactions to calculate the confidence interval for can be specified.
...	Additional parameters are ignored with a warning.

Details

This method creates a contingency table for each rule and then constructs a confidence interval for the specified measures.

Fast confidence interval approximations are currently available for the measures "support", "count", "confidence", "lift", "oddsRatio", and "phi". For all other measures, bootstrap sampling from a multinomial distribution is used.

Haldan-Anscombe correction (Haldan, 1940; Anscombe, 1956) to avoids issues with zero counts can be specified by `smoothCounts = 0.5`. Here .5 is added to each count in the contingency table.

Value

Returns a matrix with with one row for each rule and the two columns "LL" and "UL" with the interval. The matrix has the additional attributes:

measure	the interest measure.
level	the confidence level
side	the confidence level
smoothCounts	used count smoothing.
method	name of the method to create the interval
desc	desciption of the used method to calculate the confidence interval. The mentioned references can be found below.

Author(s)

Michael Hahsler

References

Wilson, E. B. (1927). "Probable inference, the law of succession, and statistical inference". Journal of the American Statistical Association. 22 (158): 209-212. doi: 10.1080/01621459.1927.10502953

Clopper, C.; Pearson, E. S. (1934). "The use of confidence or fiducial limits illustrated in the case of the binomial". Biometrika. 26 (4): 404-413. doi: 10.1093/biomet/26.4.404

Doob, J. L. (1935). "The Limiting Distributions of Certain Statistics". Annals of Mathematical Statistics. 6: 160-169. doi: 10.1214/aoms/1177732594

Fisher, R.A. (1962). "Confidence limits for a cross-product ratio". Australian Journal of Statistics, 4, 41.

Woolf, B. (1955). "On estimating the relation between blood group and diseases". Annals of Human Genetics, 19, 251-253.

Haldane, J.B.S. (1940). "The mean and variance of the moments of chi-squared when used as a test of homogeneity, when expectations are small". Biometrika, 29, 133-134.

Anscombe, F.J. (1956). "On estimating binomial response relations". Biometrika, 43, 461-464.

See Also

interestMeasure, is.redundant

Examples

```
data("Income")

# mine some rules with the consequent "language in home=english"
rules <- apriori(Income, parameter = list(support = 0.5),
  appearance = list(rhs = "language in home=english"))

# calculate the confidence interval for the rules' odds ratios.
# note that we use Haldane-Anscombe correction (with smoothCounts = .5)
# to avoid issues with 0 counts in the contingency table.
ci <- confint(rules, "oddsRatio",  smoothCounts = .5)
ci

# We add the odds ratio (with Haldane-Anscombe correction)
# and the confidence intervals to the quality slot of the rules.
quality(rules) <- cbind(
  quality(rules),
  oddsRatio = interestMeasure(rules, "oddsRatio", smoothCounts = .5),
  oddsRatio = ci)

rules <- sort(rules, by = "oddsRatio")
inspect(rules)

# use confidence intervals for lift to find rules with a lift significantly larger then 1.
# We set the confidence level to 95%, create a one-sided interval and check
# if the interval does not cover 1 (i.e., the lower limit is larger than 1).
ci <- confint(rules, "lift", level = 0.95, side = "lower")
ci

inspect(rules[ci[, "LL"] > 1])
```

---

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, reuse = TRUE)
```

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.
reuse	reuse support and confidence stored in 'x' or recompute from transactions?

Details

Coverage (also called cover or LHS-support) is the support of the left-hand-side of the rule, i.e., $supp(X)$. It represents a measure of to how often the rule can be applied.

Coverage is quickly calculated from the rules quality measures (support and confidence) stored in the quality slot. If these values are not present, then the support of the LHS is counted using the data supplied in transactions.

Coverage is also one of the measures available via the function interestMeasures.

Value

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

Author(s)

Michael Hahsler

See Also

interestMeasure, 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 *Cross-tabulate joint occurrences across pairs of items*

Description

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

Usage

```
crossTable(x, ...)

## S4 method for signature 'itemMatrix'
crossTable(x, measure = c("count", "support", "probability", "lift"),
  sort = FALSE)
```

Arguments

x	object to be cross-tabulated (transactions or itemMatrix).
measure	measure to return. Default is co-occurrence counts.
sort	sort the items by support.
...	additional arguments.

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.

Author(s)

Michael Hahsler

See Also

transactions-class, itemMatrix-class.

Examples

```
data("Groceries")

ct <- crossTable(Groceries, sort=TRUE)
ct[1:5, 1:5]

sp <- crossTable(Groceries, measure="support", sort=TRUE)
sp[1:5,1:5]

lift <- crossTable(Groceries, measure="lift", sort=TRUE)
lift[1:5,1:5]
```

DATAFRAME *Data.frame Representation for arules Objects*

Description

Provides the generic function DATAFRAME and the S4 methods to create a data.frame representation from some arules objects. based on rules. These methods are used for the coercion to a data.frame, but offers more control over the coercion process (item separators, etc.).

Usage

```
DATAFRAME(from, ...)
```

Arguments

from	the object to be converted into a data.frame (rules, itemsets, transactions).
...	further arguments.

Details

Using DATAFRAME is equivalent to the standard coercion as(x,"data.frame"). However, for rules, the argument separate = TRUE will produce separate columns for the LHS and the RHS of the rule. Furthermore, the arguments itemSep, setStart, setEnd (and ruleSep for separate = FALSE) will be passed on to the label method.

Value

a data.frame.

Author(s)

Michael Hahsler

See Also

coerce,rules,data.frame-method, coerce,itemsets,data.frame-method, coerce,transactions,data.frame-meth
labels,itemMatrix-method, LIST

Examples

```
data(Adult)

DATAFRAME(head(Adult))
DATAFRAME(head(Adult), setStart = '', itemSep = ' + ', setEnd = '')

rules <- apriori(Adult,
  parameter = list(supp = 0.5, conf = 0.9, target = "rules"))
rules <- head(rules, by = "conf")
```

```
### default coercions (same as as(rules, "data.frame"))
DATAFRAME(rules)

DATAFRAME(rules, separate = TRUE)
DATAFRAME(rules, separate = TRUE, setStart = '', itemSep = ' + ', setEnd = '')
```

`discretize` *Convert a Continuous Variable into a Categorical Variable*

Description

This function implements several basic unsupervised methods to convert a continuous variable into a categorical variable (factor) using different binning strategies. For convenience, a whole data.frame can be discretized (i.e., all numeric columns are discretized).

Usage

```
discretize(x, method = "frequency", breaks = 3,
  labels = NULL, include.lowest = TRUE, right = FALSE, dig.lab = 3,
  ordered_result = FALSE, infinity = FALSE, onlycuts = FALSE,
  categories, ...)

discretizeDF(df, methods = NULL, default = NULL)
```

Arguments

`x`	a numeric vector (continuous variable).
`method`	discretization method. Available are: `"interval"` (equal interval width), `"frequency"` (equal frequency), `"cluster"` (k-means clustering) and `"fixed"` (categories specifies interval boundaries). Note that equal frequency does not achieve perfect equally sized groups if the data contains duplicated values.
`breaks, categories`	`categories` **is deprecated, use** `breaks`**.** either number of categories or a vector with boundaries for discretization (all values outside the boundaries will be set to NA).
`labels`	character vector; labels for the levels of the resulting category. By default, labels are constructed using "(a,b]" interval notation. If `labels = FALSE`, simple integer codes are returned instead of a factor..
`include.lowest`	logical; should the first interval be closed to the left?
`right`	logical; should the intervals be closed on the right (and open on the left) or vice versa?
`dig.lab`	integer; number of digits used to create labels.
`ordered_result`	logical; return a ordered factor?

infinity	logical; should the first/last break boundary changed to +/-Inf?
onlycuts	logical; return only computed interval boundaries?
...	for method "cluster" further arguments are passed on to kmeans. .
df	data.frame; each numeric column in the data.frame is discretized.
methods	named list of lists or a data.frame; the named list contains list of discretization parameters (see parameters of discretize) for each numeric column (see details). If no specific discretization is specified for a column, then the default settings for discretize are used. Note: the names have to match exactly. If a data.frame is specified, then the discretization breaks in this data.frame are applied to df.
default	named list; parameters for discretize used for all columns not specified in methods.

Details

Discretize calculates breaks between intervals using various methods and then uses cut to convert the numeric values into intervals represented as a factor.

Discretization may fail for several reasons. Some reasons are

- A variable contains only a single value. In this case, the variable should be dropped or directly converted into a factor with a single level (see factor).
- Some calculated breaks are not unique. This can happen for method frequency with very skewed data (e.g., a large portion of the values is 0). In this case, non-unique breaks are dropped with a warning. It would be probably better to look at the histogram of the data and decide on breaks for the method fixed.

discretize only implements unsupervised discretization. See discretizeDF.supervised in package **arulesCBA** for supervised discretization.

discretizeDF applies discretization to each numeric column. Individual discretization parameters can be specified in the form: methods = list(column_name1 = list(method = ,...),column_name2 = list(...)). If no discretization method is specified for a column, then the discretization in default is applied (NULL invokes the default method in discretize()). The special method "none" can be specified to suppress discretization for a column.

Value

A factor representing the categorized continuous variable with attribute "discretized:breaks" indicating the used breaks or and "discretized:method" giving the used method. If onlycuts = TRUE is used, a vector with the calculated interval boundaries is returned.

discretizeDF returns a discretized data.frame.

Author(s)

Michael Hahsler

See Also

cut, discretizeDF.supervised.

Examples

```
data(iris)
x <- iris[,1]

### look at the distribution before discretizing
hist(x, breaks = 20, main = "Data")

def.par <- par(no.readonly = TRUE) # save default
layout(mat = rbind(1:2,3:4))

### convert continuous variables into categories (there are 3 types of flowers)
### the default method is equal frequency
table(discretize(x, breaks = 3))
hist(x, breaks = 20, main = "Equal Frequency")
abline(v = discretize(x, breaks = 3,
  onlycuts = TRUE), col = "red")
# Note: the frequencies are not exactly equal because of ties in the data

### equal interval width
table(discretize(x, method = "interval", breaks = 3))
hist(x, breaks = 20, main = "Equal Interval length")
abline(v = discretize(x, method = "interval", breaks = 3,
  onlycuts = TRUE), col = "red")

### k-means clustering
table(discretize(x, method = "cluster", breaks = 3))
hist(x, breaks = 20, main = "K-Means")
abline(v = discretize(x, method = "cluster", breaks = 3,
  onlycuts = TRUE), col = "red")

### user-specified (with labels)
table(discretize(x, method = "fixed", breaks = c(-Inf, 6, Inf),
    labels = c("small", "large")))
hist(x, breaks = 20, main = "Fixed")
abline(v = discretize(x, method = "fixed", breaks = c(-Inf, 6, Inf),
    onlycuts = TRUE), col = "red")

par(def.par)  # reset to default

### prepare the iris data set for association rule mining
### use default discretization
irisDisc <- discretizeDF(iris)
head(irisDisc)

### discretize all numeric columns differently
irisDisc <- discretizeDF(iris, default = list(method = "interval", breaks = 2,
  labels = c("small", "large")))
head(irisDisc)

### specify discretization for the petal columns and don't discretize the others
irisDisc <- discretizeDF(iris, methods = list(
  Petal.Length = list(method = "frequency", breaks = 3,
```

```
    labels = c("short", "medium", "long")),
  Petal.Width = list(method = "frequency", breaks = 2,
    labels = c("narrow", "wide"))
  ),
  default = list(method = "none")
  )
head(irisDisc)

### discretize new data using the same discretization scheme as the
###   data.frame supplied in methods. Note: NAs may occure if a new
###   value falls outside the range of values observed in the
###   originally discretized table (use argument infinity = TRUE in
###   discretize to prevent this case.)
discretizeDF(iris[sample(1:nrow(iris), 5),], methods = irisDisc)
```

`dissimilarity` *Dissimilarity Matrix Computation for Associations and Transactions*

Description

Provides the generic function `dissimilarity` and the S4 methods to compute and returns distances for binary data in a `matrix`, `transactions` or `associations` which can be used for grouping and clustering. See Hahsler (2016) for an introduction to distance-based clustering of association rules.

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 = "associations")

## 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"`):
	`"affinity"`: measure based on the `affinity`, a similarity measure between items. It is defined as the average *affinity* between the items in two transactions (see Aggarwal et al. (2002)). If `x` is not the full transaction set `args` needs to contain either precalculated affinities as element `"affinities"` or the transaction set as `"transactions"`.

"cosine": the *cosine* distance.

"dice": the *Dice's coefficient* defined by Dice (1945). Similar to *Jaccard* but gives double the weight to agreeing items.

"euclidean": the *euclidean* distance.

"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: *binary, asymmetric binary,* etc.

"matching": the *Matching coefficient* defined by Sokal and Michener (1958). This coefficient gives the same weight to presents and absence of items.

"pearson": $1 - r$ if $r > 1$ and 1 otherwise. r is *Pearson's correlation coefficient*.

"phi": same as pearson. Pearson's correlation coefficient reduces to the phi coefficient for the 2x2 contingency tables used here.

For associations the following additional measures are available:

"toivonen": Method described in Toivonen et al. (1995). For rules this measure is only defined between rules with the same consequent. The distance between two rules is defined as the number of transactions which is covered by only one of the two rules. The transactions used to mine the associations has to be passed on via args as element "transactions".

"gupta": Method described in Gupta et al. (1999). The distance between two rules is defined as 1 minus the proportion of transactions which are covered by both rules in the transactions covered by each rule individually. The transactions used to mine the associations has to be passed on via args as element "transactions".

args — a list of additional arguments for the methods.

which — a character string indicating if the dissimilarity should be calculated between transactions/associations (default) or items (use "items").

... — further arguments.

Value

returns an object of class dist.

Author(s)

Michael Hahsler

References

Aggarwal, C.C., 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.

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

Gupta, G., Strehl, A., and Ghosh, J. (1999) Distance based clustering of association rules. *In Intelligent Engineering Systems Through Artificial Neural Networks (Proceedings of ANNIE 1999)*, pages 759-764. ASME Press.

Hahsler, M. (2016) Grouping association rules using lift. In C. Iyigun, R. Moghaddess, and A. Oztekin, editors, 11th INFORMS Workshop on Data Mining and Decision Analytics (DM-DA 2016).

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.

Toivonen, H., Klemettinen, M., Ronkainen, P., Hatonen, K. and Mannila H. (1995) Pruning and grouping discovered association rules. *In Proceedings of KDD'95*.

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.D2"), main = "Dendrogram for items")

## cluster transactions for a sample of Adult
data("Adult")
s <- sample(Adult, 500)

##  calculate Jaccard distances and do hclust
d_jaccard <- dissimilarity(s)
hc <- hclust(d_jaccard, method = "ward.D2")
plot(hc, labels = FALSE, main = "Dendrogram for Transactions (Jaccard)")

## get 20 clusters and look at the difference of the item frequencies (bars)
## for the top 20 items) in cluster 1 compared to the data (line)
assign <- cutree(hc, 20)
itemFrequencyPlot(s[assign==1], population=s, topN=20)

## calculate affinity-based distances between transactions and do hclust
d_affinity <- dissimilarity(s, method = "affinity")
hc <- hclust(d_affinity, method = "ward.D2")
plot(hc, labels = FALSE, main = "Dendrogram for Transactions (Affinity)")

## cluster association rules
rules <- apriori(Adult, parameter=list(support=0.3))
rules <- subset(rules, subset = lift > 2)

## use affinity to cluster rules
## Note: we need to supply the transactions (or affinities) from the
## dataset (sample).
d_affinity <- dissimilarity(rules, method = "affinity",
  args = list(transactions = s))
```

```
hc <- hclust(d_affinity, method = "ward.D2")
plot(hc, main = "Dendrogram for Rules (Affinity)")

## create 4 groups and inspect the rules in the first group.
assign <- cutree(hc, k = 3)
inspect(rules[assign == 1])
```

`duplicated` *Find Duplicated Elements*

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.

Author(s)

Michael Hahsler

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)
```

---

eclat *Mining Associations with Eclat*

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

Details

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

Eclat can also return the transaction IDs for each found itemset using `tidLists=TRUE` as a parameter and the result can be retrieved as a `tidLists` object with method `tidLists()` for class `itemsets`. Note that storing transaction ID lists is very memory intensive, creating transaction ID lists only works for minimum support values which create a relatively small number of itemsets. See also `supportingTransactions`.

`ruleInduction` can be used to generate rules from the found itemsets.

A weighted version of ECLAT is available as function `weclat`. This version can be used to perform weighted association rule mining (WARM).

Value

Returns an object of class `itemsets`.

Author(s)

Michael Hahsler and Bettina Gruen

References

Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. (1997) *New algorithms for fast discovery of association rules*. KDD'97: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, August 1997, Pages 283-286.

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

ECLAT Implementation: https://borgelt.net/eclat.html

See Also

ECparameter-class, ECcontrol-class, transactions-class, itemsets-class, weclat, apriori, ruleInduction, supportingTransactions

Examples

```
data("Adult")
## Mine itemsets with minimum support of 0.1 and 5 or less items
itemsets <- eclat(Adult,
parameter = list(supp = 0.1, maxlen = 5))
itemsets

## Create rules from the itemsets
rules <- ruleInduction(itemsets, Adult, confidence = .9)
rules
```

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 Dec 2008.

Usage

```
data(Epub)
```

Format

Object of class transactions with 15729 transactions and 936 items. Item labels are document IDs of the from "doc_11d". Session IDs and time stamps for transactions are also provided.

Author(s)

Michael Hahsler

Source

Provided by Michael Hahsler from ePub-WU at https://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.

Author(s)

Michael Hahsler

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.

`hierarchy` *Support for Item Hierarchies*

Description

Often an item hierarchy is available for datasets used for association rule mining. For example in a supermarket dataset items like "bread" and "beagle" might belong to the item group (category) "baked goods."

We provide support to use the item hierarchy to aggregate items to different group levels, to produce multi-level transactions and to filter spurious associations mined from multi-level transactions.

Usage

```
## S4 method for signature 'itemMatrix'
aggregate(x, by)
## S4 method for signature 'itemsets'
aggregate(x, by)
## S4 method for signature 'rules'
aggregate(x, by)

addAggregate(x, by, postfix = "*")

filterAggregate(x)
```

Arguments

`x`	an transactions, itemsets or rules object.
`by`	name of a field (hierarchy level) available in `itemInfo` of `x` or a grouping vector of the same length as items in `x` by which should be aggregated. Items with the same group label in `by` will be aggregated into a single with that name. Note that the grouping vector will be coerced to factor before use.
`postfix`	characters added to mark group-level items.

Details

Transactions can store item hierarchies as additional columns in the itemInfo data.frame (`"labels"` is reserved for the item labels).

Aggregation: To perform analysis at a group level of the item hierarchy, `aggregate()` produces a new object with items aggregated to a given group level. A group-level item is present if one or more of the items in the group are present in the original object. If rules are aggregated, and the aggregation would lead to the same aggregated group item in the lhs and in the rhs, then that group item is removed from the lhs. Rules or itemsets, which are not unique after the aggregation, are also removed. Note also that the quality measures are not applicable to the new rules and thus are removed. If these measures are required, then aggregate the transactions before mining rules.

Multi-level analysis: To analyze relationships between individual items and item groups at the same time, `addAggregate()` can be used to create a new transactions object which contains both,

the original items and group-level items (marked with a given postfix). In association rule mining, all items are handled the same, which means that we will produce a large number of rules of the type

$$itemA => groupofitemA$$

with a confidence of 1. This will also happen if you mine itemsets. `filterAggregate()` can be used to filter these spurious rules or itemsets.

Value

`aggregate()` returns an object of the same class as `x` encoded with a number of items equal to the number of unique values in `by`. Note that for associations (itemsets and rules) the number of associations in the returned set will most likely be reduced since several associations might map to the same aggregated association and aggregate returns a unique set. If several associations map to a single aggregated association then the quality measures of one of the original associations is randomly chosen.

`addAggregate()` returns a new transactions object with the original items and the group-items added. `filterAggregateRules()` returns a new rules object with the spurious rules remove.

Author(s)

Michael Hahsler

See Also

`transactions`, `itemInfo`

Examples

```
data("Groceries")
Groceries

## Groceries contains a hierarchy stored in itemInfo
head(itemInfo(Groceries))

## Example 1: Aggregate items using an existing hierarchy stored in itemInfo.
## We aggregate to level2 stored in Groceries. All items with the same level2 label
## will become a single item with that name.
## Note that the number of items is therefore reduced to 55
Groceries_level2 <- aggregate(Groceries, by = "level2")
Groceries_level2
head(itemInfo(Groceries_level2)) ## labels are alphabetically sorted!


## compare original and aggregated transactions
inspect(head(Groceries, 2))
inspect(head(Groceries_level2, 2))

## Example 2: Aggregate using a character vector.
```

```
## We create here labels manually to organize items by their first letter.
mylevels <- toupper(substr(itemLabels(Groceries), 1, 1))
head(mylevels)

Groceries_alpha <- aggregate(Groceries, by = mylevels)
Groceries_alpha
inspect(head(Groceries_alpha, 2))

## Example 3: Aggregate rules
## Note: You could also directly mine rules from aggregated transactions to
## get support, lift and support
rules <- apriori(Groceries, parameter = list(supp = 0.005, conf = 0.5))
rules
inspect(rules[1])

rules_level2 <- aggregate(rules, by = "level2")
inspect(rules_level2[1])

## Example 4: Mine multi-level rules.
## (1) Add aggregate items. These items will have labels ending with a *
Groceries_multilevel <- addAggregate(Groceries, "level2")
summary(Groceries_multilevel)
inspect(head(Groceries_multilevel))

rules <- apriori(Groceries_multilevel,
  parameter = list(support = 0.01, conf = .9))
inspect(head(rules, by = "lift"))
## Note that this contains many spurious rules of type 'item X => aggregate of item X'
## with a confidence of 1 and high lift. We can filter spurious rules resulting from
## the aggregation
rules <- filterAggregate(rules)
inspect(head(rules, by = "lift"))
```

`hits` *Computing Transaction Weights With HITS*

Description

Compute the hub weights of a collection of transactions using the HITS (hubs and authorities) algorithm.

Usage

```
hits(data, iter = 16L, tol = NULL,
     type = c("normed", "relative", "absolute"), verbose = FALSE)
```

Arguments

`data`	an object of or coercible to class `transactions`.
`iter`	an integer value specifying the maximum number of iterations to use.

tol	convergence tolerance (default FLT_EPSILON).
type	a string value specifying the norming of the hub weights. For "normed" scale the weights to unit length (L2 norm), and for "relative" to unit sum.
verbose	a logical specifying if progress and runtime information should be displayed.

Details

Model a collection of transactions as a bipartite graph of hubs (transactions) and authorities (items) with unit arcs and free node weights. That is, a transaction weight is the sum of the (normalized) weights of the items and vice versa. The weights are estimated by iterating the model to a steady-state using a builtin convergence tolerance of FLT_EPSILON for (the change in) the norm of the vector of authorities.

Value

A numeric vector with transaction weights for data.

Author(s)

Christian Buchta

References

K. Sun and F. Bai (2008). Mining Weighted Association Rules without Preassigned Weights. *IEEE Transactions on Knowledge and Data Engineering*, 4 (30), 489–495.

See Also

Class transactions, function weclat

Examples

```
data(SunBai)

## calculate transaction weigths
w <- hits(SunBai)
w

## add transaction weight to the dataset
transactionInfo(SunBai)[["weight"]] <- w
transactionInfo(SunBai)

## calulate regular item frequencies
itemFrequency(SunBai, weighted = FALSE)

## calulate weighted item frequencies
itemFrequency(SunBai, weighted = TRUE)
```

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,
    xlab = "Items (Columns)",
    ylab = "Transactions (Rows)", ...)

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

Arguments

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

Author(s)

Michael Hahsler

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).

Author(s)

Michael Hahsler

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.

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 <- transactions(IncomeESL)
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 can be used to customize the output: `setStart`, `setEnd`, `itemSep` and `ruleSep`. Items are printed only one per line in case the output lines get very long. This can also be directly controlled using `linebreak`.

Value

Nothing is returned. This function is purely used for displaying object details. Use coercion with `as` to a list or a data.frame or the accessor functions provided for the object (see See Also section).

Author(s)

Michael Hahsler and Kurt Hornik

See Also

`itemMatrix-class`, `itemsets-class`, `rules-class`, `transactions-class`, `DATAFRAME`

Examples

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

## display some rules
inspect(rules[1000:1001])
inspect(rules[1000:1001], ruleSep = "~~>", itemSep = " + ", setStart = "", setEnd = "",
  linebreak = FALSE)

## to get rules in readable format, use coercion or DATAFRAME with additional parameters.
as(rules[1000:1001], "data.frame")
DATAFRAME(rules[1000:1001])
DATAFRAME(rules[1000:1001], separate = TRUE, setStart = "", setEnd = "")
```

`interestMeasure` *Calculate 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. A searchable list of definitions, equations and references for all available interest measures can be found here: https://mhahsler.github.io/arules/docs/measures

Usage

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

Arguments

`x`	a set of itemsets or rules.
`measure`	name or vector of names of the desired interest measures (see details for available measures). If measure is missing then all available measures are calculated.
`transactions`	the transaction data set used to mine the associations or a set of different transactions to calculate interest measures from (Note: you need to set `reuse=FALSE` in the later case).
`reuse`	logical indicating if information in quality slot should be reuse for calculating the measures. This speeds up the process significantly since only very little (or no) transaction counting is necessary if support, confidence and lift are already available. Use `reuse=FALSE` to force counting (might be very slow but is necessary if you use a different set of transactions than was used for mining).
`...`	further arguments for the measure calculation. Many measures are based on contingency table counts and zero counts can produce NaN values (division by zero). This issue can be resolved by using the additional parameter `smoothCounts` which performs additive smoothing by adds a "pseudo count" of `smoothCounts` to each count in the contingency table. Use `smoothCounts = 1` or larger values for Laplace smoothing. Use `smoothCounts = .5` for Haldane-Anscombe correction often used for chi-squared, phi correlation and related measures.

Details

The following measures are implemented for itemsets X:

"allConfidence" Is defined on itemsets as the minimum confidence of all possible rule generated from the itemset.

See details: https://mhahsler.github.io/arules/docs/measures#all-confidence

Range: $[0, 1]$

"crossSupportRatio", cross-support ratio 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.

See details: https://mhahsler.github.io/arules/docs/measures#cross-support-ratio

Range: $[0, 1]$

"lift" Lift is typically only defined for rules. In a similar way, we use the probability (support) of the itemset over the product of the probabilities of all items in the itemset, i.e., $\frac{supp(X)}{\prod_{x \in X} supp(X)}$.

Range: $[0, \infty]$ (1 indicated independence)

"support", supp Support is an estimate of $P(X)$, a measure of generality of the itemset. It is estimated by the number of transactions that contain the itemset over the total number of transactions in the data set.

See details: https://mhahsler.github.io/arules/docs/measures#support

Range: $[0, 1]$

"count" Absolute support count of the itemset, i.e., the number of transactions that contain the itemset.

See details: https://mhahsler.github.io/arules/docs/measures#support

Range: $[0, \infty]$

The following measures are implemented for rules of the form $X \Rightarrow Y$:

"addedValue", added Value, AV, Pavillon index, centered confidence Defined as the rule confidence minus the rule's support.

See details: https://mhahsler.github.io/arules/docs/measures#added-value

Range: $[-.5, 1]$

"boost", confidence boost Confidence boost is the ratio of the confidence of a rule to the confidence of any more general rule (i.e., a rule with the same consequent but one or more items removed in the LHS). Values larger than 1 mean the new rule boosts the confidence compared to the best, more general rule. The measure is related to the improvement measure.

See details: https://mhahsler.github.io/arules/docs/measures#confidence-boost

Range: $[0, \infty]$

"chiSquared", χ^2 statistic The chi-squared statistic to test for independence between the lhs and rhs of the rule. The critical value of the chi-squared distribution with 1 degree of freedom (2x2 contingency table) at $\alpha = 0.05$ is 3.84; higher chi-squared values indicate that the lhs and the rhs are not independent.

See details: https://mhahsler.github.io/arules/docs/measures#chi-squared

Note that the contingency table is likely to have cells with low expected values and that thus Fisher's Exact Test might be more appropriate (see below).

Called with `significance = TRUE`, the p-value of the test for independence is returned instead of the chi-squared statistic. For p-values, substitution effects (the ocurrence of one item makes the ocurrance of another item less likely) can be tested using the parameter `complements = FALSE`. Correction for multiple comparisons can be done using `p.adjust`.

Range: $[0, \infty]$ or p-value scale

"certainty", certainty factor, CF, Loevinger The certainty factor is a measure of variation of the probability that Y is in a transaction when only considering transactions with X. An increasing CF means a decrease of the probability that Y is not in a transaction that X is in. Negative CFs have a similar interpretation.

See details: https://mhahsler.github.io/arules/docs/measures#certainty-factor

Range: $[-1, 1]$ (0 indicates independence)

"collectiveStrength", Collective strength, S Collective strength gives 0 for perfectly negative correlated items, infinity for perfectly positive correlated items, and 1 if the items co-occur as expected under independence.

See details: https://mhahsler.github.io/arules/docs/measures#collective-strength

Range: $[0, \infty]$

"confidence", Strength, conf Confidence is a measure of rule validity. Rule confidence is an estimate of $P(Y|X)$.

See details: https://mhahsler.github.io/arules/docs/measures#confidence

Range: $[0, 1]$

"conviction" Conviction was developed as an alternative to lift that also incorporates the direction of the rule.

See details: https://mhahsler.github.io/arules/docs/measures#conviction

Range: $[0, \infty]$ (1 indicates unrelated items)

"cosine" A measure if correlation between the items in X and Y.

See details: https://mhahsler.github.io/arules/docs/measures#cosine

Range: $[0, 1]$(.5 indicates no correlation)

"count" Absolute support count of the rule, i.e., the number of transactions that contain all items in the rule.

See details: https://mhahsler.github.io/arules/docs/measures#support

Range: $[0, \infty]$

"coverage", cover, LHS-support It measures the probability that a rule applies to a randomly selected transaction. It is estimated by the proportion of transactions that contain the antecedent (LHS) of the rule. Therefore, coverage is sometimes called antecedent support or LHS support.

See details: https://mhahsler.github.io/arules/docs/measures#coverage

Range: $[0, 1]$

"confirmedConfidence", descriptive confirmed confidence How much higher is the confidence of a rule compared to the confidence of the rule $X \Rightarrow \overline{Y}$.

See details: https://mhahsler.github.io/arules/docs/measures#descriptive-confirmed-confidence

Range: $[-1, 1]$

"casualConfidence", casual confidence Confidence reinforced by the confidence of the rule $\overline{X} \Rightarrow \overline{Y}$.

See details: https://mhahsler.github.io/arules/docs/measures#casual-confidence

Range: $[0, 1]$

"casualSupport", casual support Support reinforced by the support of the rule $\overline{X} \Rightarrow \overline{Y}$.

See details: https://mhahsler.github.io/arules/docs/measures#casual-support Range: $[-1, 1]$

"counterexample", example and counter-example rate Rate of the examples minus the rate of counter examples (i.e., $X \Rightarrow \overline{Y}$).

See details: https://mhahsler.github.io/arules/docs/measures#example-and-counter-example-rate

Range: $[0, 1]$

"doc", difference of confidence Defined as the difference in confidence of the rule and the rule $\overline{X} \Rightarrow Y$

See details: https://mhahsler.github.io/arules/docs/measures#difference-of-confidence

Range: $[-1, 1]$

"fishersExactTest", Fisher's exact test p-value of Fisher's exact test used in the analysis of contingency tables where sample sizes are small. By default complementary effects are mined, substitutes can be found by using the parameter `complements = FALSE`.

See details: https://mhahsler.github.io/arules/docs/measures#fishers-exact-test

Note that it is equal to hyper-confidence with `significance=TRUE`. Correction for multiple comparisons can be done using `p.adjust`.

Range: $[0, 1]$ (p-value scale)

"gini", Gini index Measures quadratic entropy of a rule.

See details: https://mhahsler.github.io/arules/docs/measures#gini-index

Range: $[0, 1]$ (0 means the rule provides no information for the data set)

"hyperConfidence" Confidence level that the observed co-occurrence count of the LHS and RHS is too high given the expected count using the hypergeometric model.

See details: https://mhahsler.github.io/arules/docs/measures#hyper-confidence

Hyper-confidence reports the confidence level by default and the significance level if `significance=TRUE` is used.

By default complementary effects are mined, substitutes (too low co-occurrence counts) can be found by using the parameter `complements = FALSE`.

Range: $[0, 1]$

"hyperLift" Adaptation of the lift measure which evaluates the deviation from independence using a quantile of the hypergeometric distribution defined by the counts of the LHS and RHS. HyperLift can be used to calculate confidence intervals for the lift measure.

The used quantile can be given as parameter `level` (default: `level = 0.99`).

See details: https://mhahsler.github.io/arules/docs/measures#hyper-lift

Range: $[0, \infty]$ (1 indicates independence)

"imbalance", imbalance ratio, IR IR measures the degree of imbalance between the two events that the lhs and the rhs are contained in a transaction. The ratio is close to 0 if the conditional probabilities are similar (i.e., very balanced) and close to 1 if they are very different. See also: https://mhahsler.github.io/arules/docs/measures#imbalance-ratio

Range: $[0, 1]$ (0 indicates a balanced rule)

"implicationIndex", implication index A variation of the Lerman similarity.

See details: https://mhahsler.github.io/arules/docs/measures#implication-index

Range: $[0, 1]$ (0 means independence)

"importance" Log likelihood of the right-hand side of the rule, given the left-hand side of the rule using Laplace corrected confidence.

See details: https://mhahsler.github.io/arules/docs/measures#importance

Range: $[-Inf, Inf]$

"improvement" The improvement of a rule is the minimum difference between its confidence and the confidence of any more general rule (i.e., a rule with the same consequent but one or more items removed in the LHS).

Special case: We define improvement for a rules with an empty LHS as its confidence.

The idea of improvement can be generalized to other measures than confidence. Other measures like lift can be specified with the extra parameter `improvementMeasure`.

See details: https://mhahsler.github.io/arules/docs/measures#improvement

Range: $[0, 1]$

"jaccard", Jaccard coefficient, sometimes also called Coherence Null-invariant measure of dependence defined as the Jaccard similarity between the two sets of transactions that contain the items in X and Y, respectively.

See details: https://mhahsler.github.io/arules/docs/measures#jaccard-coefficient

Range: $[0, 1]$

"jMeasure", J-measure, J A scaled measures of cross entropy to measure the information content of a rule.

See details: https://mhahsler.github.io/arules/docs/measures#j-measure

Range: $[0, 1]$ (0 indicates X does not provide information for Y)

"kappa" Cohen's Kappa (Tan and Kumar, 2000) Cohen's Kappa of the rule (seen as a classifier) given as the rule's observed rule accuracy (i.e., confidence) corrected by the expected accuracy (of a random classifier).

See details: https://mhahsler.github.io/arules/docs/measures#kappa

Range: $[-1, 1]$ (0 means the rule is not better than a random classifier)

"klosgen" Defined as $\sqrt{supp(X \cup Y)}conf(X \Rightarrow Y) - supp(Y)$

See details: https://mhahsler.github.io/arules/docs/measures#klosgen

Range: $[-1, 1]$ (0 for independence)

"kulczynski", kulc Calculate the null-invariant Kulczynski measure with a preference for skewed patterns.

See details: https://mhahsler.github.io/arules/docs/measures#kulczynski

Range: $[0, 1]$

"lambda", Goodman-Kruskal's λ, predictive association Goodman and Kruskal's lambda to assess the association between the LHS and RHS of the rule.

See details: https://mhahsler.github.io/arules/docs/measures#lambda

Range: $[0, 1]$

"laplace", Laplace corrected confidence/accuracy, L Estimates confidence by increasing each count by 1. Parameter `k` can be used to specify the number of classes (default is 2). Prevents counts of 0 and L decreases with lower support.

See details: https://mhahsler.github.io/arules/docs/measures#laplace-corrected-confidence

Range: $[0, 1]$

"leastContradiction", least contradiction Probability of finding a matching transaction minus the probability of finding a contradicting transaction normalized by the probability of finding a transaction containing Y.

See details: https://mhahsler.github.io/arules/docs/measures#least-contradiction

Range: $[-1, 1]$

"lerman", Lerman similarity Defined as $\sqrt{N}\frac{supp(X \cup Y) - supp(X)supp(Y)}{\sqrt{supp(X)supp(Y)}}$

See details: https://mhahsler.github.io/arules/docs/measures#lerman-similarity

Range: $[0, 1]$

"leverage", Piatetsky-Shapiro Measure, PS PS measures the difference of X and Y appearing together in the data set and what would be expected if X and Y where statistically dependent. It can be interpreted as the gap to independence.

See details: https://mhahsler.github.io/arules/docs/measures#leverage

Range: $[-1, 1]$ (0 indicates independence)

"lift", interest factor Lift quantifies dependence between X and Y by comparing the probability that X and Y are contained in a transaction to the expected probability under independence (i.e., the product of the probabilities that X is contained in a transaction times the probability that Y is contained in a transaction).

See details: https://mhahsler.github.io/arules/docs/measures#lift

Range: $[0, \infty]$ (1 means independence between LHS and RHS)

"maxConfidence" Null-invariant symmetric measure defined as the larger of the confidence of the rule or the rule with X and Y exchanged.

See details: https://mhahsler.github.io/arules/docs/measures#maxconfidence Range: $[0, 1]$

"mutualInformation", uncertainty, M Measures the information gain for Y provided by X.

See details: https://mhahsler.github.io/arules/docs/measures#mutual-information

Range: $[0, 1]$ (0 means that X does not provide information for Y)

"oddsRatio", odds ratio The odds of finding X in transactions which contain Y divided by the odds of finding X in transactions which do not contain Y. For zero counts, Haldane-Anscombe correction (adding .5 to all zells) is applied.

See details: https://mhahsler.github.io/arules/docs/measures#odds_ratio

Range: $[0, \infty]$ (1 indicates that Y is not associated to X)

"oddsRatioCI", odds ratio confidence interval Calculates the lower and upper bounds of the confidence interval around the odds ratio (using a normal approximation). The used confidence level defaults to 0.95, but can be adjusted with the additional parameter `confidenceLevel`.

See details: https://mhahsler.github.io/arules/docs/measures#odds-ratio

Range: $[0, \infty]$

"phi", correlation coefficient ϕ Correlation coefficient between the transactions containing X and Y represented as two binary vectors. Phi correlation is equivalent to Pearson's Product Moment Correlation Coefficient ρ with 0-1 values.

See details: https://mhahsler.github.io/arules/docs/measures#phi-correlation-coefficient
Range: $[-1, 1]$ (0 when X and Y are independent)

"ralambondrainy", Ralambondrainy Measure The measure is defined as the probability that a transaction contains X but not Y. A smaller value is better.

See details: https://mhahsler.github.io/arules/docs/measures#ralambondrainy

Range: $[0, 1]$

"rhsSupport", Support of the rule consequent Range: $[0, 1]$

"RLD", relative linkage disequilibrium RLD is an association measure motivated by indices used in population genetics. It 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.
See details: https://mhahsler.github.io/arules/docs/measures#relative-linkage-disequilibrium
The code was contributed by Silvia Salini.
Range: $[0, 1]$

"rulePowerFactor", rule power factor Product of support and confidence. Can be seen as rule confidence weighted by support.
See details: https://mhahsler.github.io/arules/docs/measures#rule-power-factor
Range: $[0, 1]$

"sebag", Sebag-Schoenauer measure Confidence of a rule divided by the confidence of the rule $X \Rightarrow \overline{Y}$.
See details: https://mhahsler.github.io/arules/docs/measures#sebag-schoenauer
Range: $[0, 1]$

"stdLift", Standardized Lift Standardized lift uses the minimum and maximum lift can reach for each rule to standardize lift between 0 and 1. By default, the measure is corrected for minimum support and minimum confidence. Correction can be disabled by using the argument `correct = FALSE`.
See details: https://mhahsler.github.io/arules/docs/measures#standardized-lift
Range: $[0, 1]$

"support", supp Support is an estimate of $P(X \cup Y)$ and measures the generality of the rule.
See details: https://mhahsler.github.io/arules/docs/measures#support
Range: $[0, 1]$

"table" Returns the counts for the contingency table. The values are labeled n_{XY} where X and Y represent the presence (1) or absence (0) of the LHS and RHS of the rule, respectively. If several measures are specified, then the counts have the prefix `table.`
Range: counts

"varyingLiaison", varying rates liaison Defined as the lift of a rule minus 1 so 0 represents independence.
See details: https://mhahsler.github.io/arules/docs/measures#Varying-Rates-Liaison
Range: $[-1, \infty]$ (0 for independence)

"yuleQ", Yule's Q Defined as $\frac{\alpha-1}{\alpha+1}$ where α is the odds ratio.
See details: https://mhahsler.github.io/arules/docs/measures#yules-q-and-yules-y
Range: $[-1, 1]$

"yuleY", Yule's Y Defined as $\frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}$ where α is the odds ratio.
See details: https://mhahsler.github.io/arules/docs/measures#yules-q-and-yules-y
Range: $[-1, 1]$

Value

If only one measure 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 measures are specified, the result is a data.frame containing the different measures for each association as columns.

`NA` is returned for rules/itemsets for which a certain measure is not defined.

Author(s)

Michael Hahsler

References

A complete list of references for each individual measure is available in the following document:

Hahsler, Michael (2015). A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules, 2015, URL: https://mhahsler.github.io/arules/docs/measures.

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, measure = "hyperConfidence",
transactions = Income))

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

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

## calculate all available measures for the first 5 rules and show them as a
## table with the measures as rows
t(interestMeasure(head(rules, 5), transactions = Income))

## calculate measures on a different set of transactions (I use a sample here)
## Note: reuse = TRUE (default) would just return the stored support on the
## data set used for mining
newTrans <- sample(Income, 100)
m2 <- interestMeasure(rules, "support", transactions = newTrans, reuse = FALSE)
head(m2)

## calculate all available measures for the 5 frequent itemsets with highest support
its <- apriori(Income, parameter = list(target = "frequent itemsets"))
its <- head(its, 5, by = "support")
inspect(its)

interestMeasure(its, transactions = Income)
```

`is.closed` *Find Closed Itemsets*

Description

Provides the generic function and the S4 method `is.closed` for finding closed itemsets. Closed itemsets are used as a concise representation of frequent 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).

Closed frequent itemsets can also be mined directly using `link{apriori}` or `link{eclat}` with target "closed frequent itemsets".

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.

Author(s)

Michael Hahsler

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`, `is.generator`

`is.generator` *Find Generator Itemsets*

Description

Provides the generic function and the S4 method `is.generator` for finding generator itemsets. Generators are part of concise representations for frequent itemsets. A generator in a set of itemsets is an itemset that has no subset with the same support (Liu et al, 2008). Note that the empty set is by definition a generator, but it is typicaly not stored in the itemsets in **arules**.

Usage

```
is.generator(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 generator itemset.

Author(s)

Michael Hahsler

References

Yves Bastide, Niolas Pasquier, Rafik Taouil, Gerd Stumme, Lotfi Lakhal (2000). Mining Minimal Non-redundant Association Rules Using Frequent Closed Itemsets. In *International Conference on Computational Logic*, Lecture Notes in Computer Science (LNCS 1861). pages 972–986. doi: 10.1007/3540449574_65

Guimei Liu, Jinyan Li, Limsoon Wong (2008). A new concise representation of frequent itemsets using generators and a positive border. *Knowledge and Information Systems* 17(1):35-56. doi: 10.1007/s1011500701115

See Also

`itemsets-class`, `is.closed`, `is.maximal`

Examples

```
# Example from Liu et al (2008)
trans_list <- list(
      t1 = c("a","b","c"),
      t2 = c("a","b", "c", "d"),
      t3 = c("a","d"),
      t4 = c("a","c")
```

```
      )

trans <- transactions(trans_list)
its <- apriori(trans, support = 1/4, target = "frequent itemsets")

is.generator(its)
```

`is.maximal` *Find Maximal Itemsets*

Description

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

Maximally frequent itemsets can also be mined directly using `link{apriori}` or `link{eclat}` with target "maximally frequent itemsets".

We define here maximal rules, as the rules generated by maximal itemsets.

Usage

```
is.maximal(x,...)

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

Arguments

`x`	the set of itemsets, rules 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.

Author(s)

Michael Hahsler

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

`itemMatrix-class`, `itemsets-class`, `is.generator`, `is.closed`

`is.redundant` *Find Redundant Rules*

Description

Provides the generic function and the S4 method `is.redundant` to find redundant rules.

Usage

```
is.redundant(x, ...)

## S4 method for signature 'rules'
is.redundant(x, measure = "confidence",
  confint = FALSE, level = 0.95, smoothCounts = 1, ...)
```

Arguments

`x`	a set of rules.
`measure`	measure used to check for redundancy.
`confint`	should confidence intervals be used to the redundancy check?
`level`	confidence level for the confidence interval. Only used when `confint = TRUE`.
`smoothCounts`	adds a "pseudo count" to each count in the used contingency table. This implements addaptive smoothing (Laplace smoothing) for counts and avoids zero counts.
`...`	additional arguments are passed on to `interestMeasure`, or, for `confint = TRUE` to `confint`.

Details

Simple improvement-based redundancy: (`confint = FALSE`) A rule can be defined as redundant if a more general rules with the same or a higher confidence exists. That is, a more specific rule is redundant if it is only equally or even less predictive than a more general rule. A rule is more general if it has the same RHS but one or more items removed from the LHS. Formally, a rule $X \Rightarrow Y$ is redundant if

$$\exists X' \subset X \quad conf(X' \Rightarrow Y) \geq conf(X \Rightarrow Y).$$

This is equivalent to a negative or zero *improvement* as defined by Bayardo et al. (2000).

The idea of improvement can be extended other measures besides confidence. Any other measure availabe for function `interestMeasure` (e.g., lift or the odds ratio) can be specified in `measure`.

Confidence interval-based redundancy: (`confint = TRUE`) Li et al (2014) propose to use the confidence interval (CI) of the odds ratio (OR) of rules to define redundancy. A more specific rule is redundant if it does not provide a significantly higher OR than any more general rule. Using confidence intervals as error bounds, a more specific rule is redundant if its OR CI overlaps with the CI of any more general rule (i.e., the lower bound of the more specific rule's CI is lower than the

upper bound of any more general rule's CI). This type of redundancy detection is more powerful than improvement since it takes differences in counts due to randomness in the dataset into account.

The odds ratio and the CI are based on counts which can be zero and which leads to numerical problems. In addition to the method described by Li et al (2014), we use additive smoothing (Laplace smoothing) to alleviate this problem. The default setting adds 1 to each count (see `confint`). A different pseudocount (smoothing parameter) can be defined using the additional parameter `smoothCounts`. Smoothing can be disabled using `smoothCounts = 0`.

Confidence interval-based redundancy checks can also be used for other measures with a confidence interval like confidence (see `confint`).

Value

returns a logical vector indicating which rules are redundant.

Author(s)

Michael Hahsler and Christian Buchta

References

Bayardo, R. , R. Agrawal, and D. Gunopulos (2000). Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery,* 4(2/3):217–240.

Li, J., Jixue Liu, Hannu Toivonen, Kenji Satou, Youqiang Sun, and Bingyu Sun (2014). Discovering statistically non-redundant subgroups. Knowledge-Based Systems. 67 (September, 2014), 315–327. doi: 10.1016/j.knosys.2014.04.030

See Also

`interestMeasure`, `confint`

Examples

```
data("Income")

## mine some rules with the consequent "language in home=english"
rules <- apriori(Income, parameter = list(support = 0.5),
  appearance = list(rhs = "language in home=english"))

## for better comparison we add Bayado's improvement and sort by improvement
quality(rules)$improvement <- interestMeasure(rules, measure = "improvement")
rules <- sort(rules, by = "improvement")
inspect(rules)
is.redundant(rules)

## find non-redundant rules using improvement of confidence
## Note: a few rules have a very small improvement over the rule {} => {language in home=english}
rules_non_redundant <- rules[!is.redundant(rules)]
inspect(rules_non_redundant)

## use non-overlapping confidence intervals for the confidence measure instead
```

```
## Note: fewer rules have a significantly higher confidence
inspect(rules[!is.redundant(rules, measure = "confidence",
  confint = TRUE, level = 0.95)])

## find non-redundant rules using improvement of the odds ratio.
quality(rules)$oddsRatio <- interestMeasure(rules, measure = "oddsRatio", smoothCounts = .5)
inspect(rules[!is.redundant(rules, measure = "oddsRatio")])

## use the confidence interval for the odds ratio.
## We see that no rule has a significantly better odds ratio than the most general rule.
inspect(rules[!is.redundant(rules, measure = "oddsRatio",
  confint = TRUE, level = 0.95)])

##  use the confidence interval for lift
inspect(rules[!is.redundant(rules, measure = "lift",
  confint = TRUE, level = 0.95)])
```

`is.significant` *Find Significant Rules*

Description

Provides the generic functions and the S4 method `is.significant` to find significant associations and an implementation for rules.

Usage

```
is.significant(x, transactions, method = "fisher",
  alpha = 0.01, adjust = "bonferroni")
```

Arguments

`x`	a set of rules.
`transactions`	set of transactions used to mine the rules.
`method`	test to use. Options are `"fisher"`, `"chisq"`. Note that the contingency table is likely to have cells with low expected values and that thus Fisher's Exact Test might be more appropriate than the chi-squared test.
`alpha`	required significance level.
`adjust`	method to adjust for multiple comparisons. Options are `"none"`, `"bonferroni"`, `"holm"`, `"fdr"`, etc. (see `p.adjust`)

Details

The implementation for association rules uses Fisher's exact test with correction for multiple comparisons to test the null hypothesis that the LHS and the RHS of the rule are independent. Significant rules have a p-value less then the specified significance level alpha (the null hypothesis of independence is rejected.).

Value

returns a logical vector indicating which rules are significant.

Author(s)

Michael Hahsler

References

Hahsler, Michael and Kurt Hornik (2007). New probabilistic interest measures for association rules. *Intelligent Data Analysis*, 11(5):437–455.

See Also

`interestMeasure`, `p.adjust`

Examples

```
data("Income")
rules <- apriori(Income, parameter = list(support = 0.5))
is.significant(rules, Income)

inspect(rules[is.significant(rules, Income)])
```

---

is.superset *Find Super and Subsets*

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

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.
`sparse`	a logical indicating if a sparse (ngCMatrix) rather than a dense logical matrix should be returned. Sparse computation preserves a significant amount of memory and is much faster for large sets.
`...`	currently unused.

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.

For rules, the union of lhs and rhs is used a the set of items.

Value

returns a logical matrix or a sparse ngCMatrix (for sparse=TRUE) 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.

Author(s)

Michael Hahsler and Ian Johnson

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)
is.superset(set, set, sparse = FALSE)
```

---

itemCoding *Item Coding — Conversion between Item Labels and Column IDs*

Description

The order in which items are stored in an itemMatrix is called the *item coding*. The following generic functions and S4 methods are used to translate between the binary representation in the itemMatrix format (used in transactions, rules and itemsets), item labels and numeric item IDs (i.e., the column numbers in the binary representation).

Usage

```
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)
```

```
compatible(x, y)

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

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

Arguments

x	a vector or a list of vectors of character strings (for encode) or of numeric (for decode), or an object of class itemMatrix (for recode).
itemLabels	a vector of character strings used for coding where the position of an item label in the vector gives the item's column ID. Alternatively, a itemMatrix, transactions or associations object can be specified and the item labels or these objects are used.
itemMatrix	return an object of class itemMatrix otherwise an object of the same class as x is returned.
y	an object of class itemMatrix, transactions or associations to compare item coding to x.
match	deprecated: used itemLabels instead.
...	further arguments.

Details

Item compatibility: If you deal with several datasets or different subsets of the same dataset and want to combine or compate the found itemsets or rules, then you need to make sure that all transaction sets have a compatible item coding. That is, the sparse matrices representing the items have columns for the same items in exactly the same order. The coercion to transactions with as(x,"transactions") will create the item coding by adding items when they are encountered in the dataset. This can lead to different item codings (different order, missing items) for even only slightly different datasets. You can use the method compatible to check if two sets have the same item coding.

If you work with many sets, then you should first define a common item coding by creating a vector with all possible item labels and then use either encode to create transactions or recode to make a different set compatible.

The following function help with creating and changing the item coding to make them compatible.

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.

decode converts from the column IDs used in the itemMatrix representation to item labels. decode is used by LIST.

recode recodes an itemMatrix object so its coding is compatible with another itemMatrix object specified in itemLabels (i.e., the columns are reordered to match).

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.

Author(s)

Michael Hahsler

See Also

LIST, associations-class, itemMatrix-class

Examples

```
data("Adult")

## Example 1: Manual decoding
## Extract the item coding as a vector of item labels.
iLabels <- itemLabels(Adult)
head(iLabels)

## get undecoded list (itemIDs)
list <- LIST(Adult[1:5], decode = FALSE)
list

## decode itemIDs by replacing them with the appropriate item label
decode(list, itemLabels = iLabels)


## Example 2: Manually create an itemMatrix using iLabels as the common item coding
data <- list(
    c("income=small", "age=Young"),
    c("income=large", "age=Middle-aged")
    )

# Option a: encode to match the item coding in Adult
iM <- encode(data, itemLabels = Adult)
iM
inspect(iM)
compatible(iM, Adult)
```

```
# Option b: coercion plus recode to make it compatible to Adult
#           (note: the coding has 115 item columns after recode)
iM <- as(data, "itemMatrix")
iM
compatible(iM, Adult)

iM <- recode(iM, itemLabels = Adult)
iM
compatible(iM, Adult)


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

## After choosing only a subset of items (columns), the item coding is now
## no longer compatible with the Adult dataset
compatible(sub, Adult)

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


## Example 4: manually create 2 new transaction for the Adult data set
##            Note: check itemLabels(Adult) to see the available labels for items
twoTransactions <- as(
    encode(list(
        c("age=Young", "relationship=Unmarried"),
        c("age=Senior")
      ), itemLabels = Adult),
    "transactions")

twoTransactions
inspect(twoTransactions)

## the same using the transactions constructor function instead
twoTransactions <- transactions(
    list(
        c("age=Young", "relationship=Unmarried"),
        c("age=Senior")
    ), itemLabels = Adult)

twoTransactions
inspect(twoTransactions)

## Example 5: Use a common item coding

# Creation of transactions separately will produce different item codings
trans1 <- transactions(
    list(
```

```
        c("age=Young", "relationship=Unmarried"),
        c("age=Senior")
    ))
trans1

trans2 <- transactions(
    list(
        c("age=Middle-aged", "relationship=Married"),
        c("relationship=Unmarried", "age=Young")
    ))
trans2

compatible(trans1, trans2)

# produce common item coding (all item labels in the two sets)
commonItemLabels <- union(itemLabels(trans1), itemLabels(trans2))
commonItemLabels

trans1 <- recode(trans1, itemLabels = commonItemLabels)
trans1
trans2 <- recode(trans2, itemLabels = commonItemLabels)
trans2

compatible(trans1, trans2)


## Example 6: manually create a rule using the item coding in Adult
## and calculate interest measures
aRule <- new("rules",
  lhs = encode(list(c("age=Young", "relationship=Unmarried")),
    itemLabels = Adult),
  rhs = encode(list(c("income=small")),
    itemLabels = Adult)
)

## shorter version using the rules constructor
aRule <- rules(
  lhs = list(c("age=Young", "relationship=Unmarried")),
  rhs = list(c("income=small")),
  itemLabels = Adult
)

quality(aRule) <- interestMeasure(aRule,
  measure = c("support", "confidence", "lift"), transactions = Adult)

inspect(aRule)
```

---

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, weighted = FALSE)
```

Arguments

`x`	an object.
`...`	further arguments are passed on.
`type`	a character string specifying if `"relative"` frequency/support or `"absolute"` frequency/support (item counts) is returned. (default: `"relative"`).
`weighted`	should support be weighted by transactions weights stored as column `"weight"` in transactionInfo?

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`.

Author(s)

Michael Hahsler

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"),
    weighted = FALSE, support = NULL, topN = NULL,
    population = NULL, popCol = "black", popLwd = 1,
    lift = FALSE, horiz = FALSE,
    names = TRUE, cex.names =  graphics::par("cex.axis"),
    xlab = NULL, ylab = NULL, mai = NULL, ...)
```

Arguments

`x`	the object to be plotted.
`...`	further arguments are passed on (see `barplot` from possible arguments).
`type`	a character string indicating whether item frequencies should be displayed relative of absolute.
`weighted`	should support be weighted by transactions weights stored as column `"weight"` in transactionInfo?
`support`	a numeric value. Only display items which have a support of at least `support`. If no population is given, support is calculated from `x` otherwise from the population. Support is interpreted relative or absolute according to the setting of `type`.
`topN`	a integer value. Only plot the `topN` items with the highest item frequency or lift (if `lift = TRUE`). The items are plotted ordered by descending support.
`population`	object of same class as `x`; if `x` is a segment of a population, the population mean frequency for each item can be shown as a line in the plot.
`popCol`	plotting color for population.
`popLwd`	line width for population.
`lift`	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 `x` than in `population`.
`horiz`	a logical. If `horiz = FALSE` (default), the bars are drawn vertically. If `TRUE`, the bars are drawn horizontally.
`names`	a logical indicating if the names (bar labels) should be displayed?

cex.names	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.

Author(s)

Michael Hahsler

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.

Details

Sets of itemsets (or transactions) are represented as a compressed sparse binary matrix. Columns represent items and rows are the set/transactions. In the compressed form, each itemset is a vector of column indices (called item IDs) representing the items.

Note: If you work with several itemMatrices at the same time (e.g., several transaction sets, lhs and rhs of a rule, etc.), then the encoding (itemLabes and order of the items in the binary matrix) in the different itemMatrices is important and needs to conform. See `itemCoding` to learn how to encode and recode itemMatrix objects.

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

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

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

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

coerce `signature(from = "ngCMatrix", to = "itemMatrix")`; Note, the `ngCMatrix` has to be transposed with items as rows!

c `signature(object = "itemMatrix")`; combine.

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

dimnames, rownames, colnames `signature(x = "itemMatrix")`; returns row (itemsetID) and column (item) names.

dimnames `signature(x = "itemMatrix")`; returns dimnames.

dimnames<- `signature(x = "itemMatrix", value = "list")`; replace dimnames.

%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`.

%oin% `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 *only* 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 labels for the itemsets. The following arguments can be used to customize the representation of the labels: `itemSep`, `setStart` and `setEnd`.

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

toLongFormat `signature(object = "itemMatrix")`; convert the transactions to long format (a data.frame with two columns, tid and item). Column names can be specified as a character vector of length 2 called `cols`.

show `signature(object = "itemMatrix")`

summary `signature(object = "itemMatrix")`

Author(s)

Michael Hahsler

See Also

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

Examples

```
set.seed(1234)

## Generate a logical matrix with 5000 random itemsets for 20 items
m <- matrix(runif(5000*20)>0.8, ncol=20,
            dimnames = list(NULL, paste("item", c(1:20), sep="")))
head(m)

## Coerce the logical matrix into an itemMatrix object
imatrix <- as(m, "itemMatrix")
imatrix

## An itemMatrix contains a set of itemsets (each row is an itemset).
## The length of the set is the number of rows.
length(imatrix)

## The sparese matrix also has regular matrix  dimensions.
dim(imatrix)
nrow(imatrix)
ncol(imatrix)

## Subsetting: Get first 5 elements (rows) of the itemMatrix. This can be done in
## several ways.
imatrix[1:5]            ### get elements 1:5
imatrix[1:5, ]          ### Matrix subsetting for rows 1:5
head(imatrix, n = 5)    ### head()

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

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

## Get first 5 elements (rows) of the itemMatrix as sparse ngCMatrix.
## Warning: For efficiency reasons, the ngCMatrix is transposed!
as(imatrix[1:5], "ngCMatrix")

## Get labels for the first 5 itemsets (first default and then with
## custom formating)
labels(imatrix[1:5])
labels(imatrix[1:5], itemSep = " + ", setStart = "", setEnd = "")

## Create itemsets manually from an itemMatrix. Itemsets contain items in the form of
## an itemMatrix and additional quality measures (not supplied in the example).
is <- new("itemsets", items = imatrix)
is
inspect(head(is, n = 3))


## Create rules manually. I use imatrix[4:6] for the lhs of the rules and
## imatrix[1:3] for the rhs. Rhs and lhs cannot share items so I use
## itemSetdiff here. I also assign missing values for the quality measures support
```

```
## and confidence.
rules <- new("rules",
             lhs = itemSetdiff(imatrix[4:6], imatrix[1:3]),
             rhs = imatrix[1:3],
             quality = data.frame(support = c(NA, NA, NA),
                                  confidence =  c(NA, NA, NA)
          ))
rules
inspect(rules)

## Manually create a itemMatrix with an item encoding that matches imatrix (20 items in order
## item1, item2, ..., item20)
itemset_list <- list(c("item1","item2"),
                     c("item3"))

imatrix_new <- encode(itemset_list, itemLabels = imatrix)
imatrix_new
compatible(imatrix_new, imatrix)
```

`itemSetOperations` *Itemwise Set Operations*

Description

Provides the generic functions and the S4 methods for itemwise set operations on items in an itemMatrix. The regular set operations regard each itemset in an itemMatrix as an element. Itemwise operations regard each item as an element and operate on the items of pairs if corresponding itemsets (first itemset in x with first itemset in y, second with second, etc.).

Usage

```
itemUnion(x, y)
itemSetdiff(x, y)
itemIntersect(x, y)
```

Arguments

`x,y`	two itemMatrix objects with the same number of rows (itemsets).

Value

An object of class `itemMatrix` is returned.

Author(s)

Michael Hahsler

See Also

`itemMatrix-class`

Examples

```
data("Adult")

fsets <- eclat(Adult, parameter = list(supp = 0.5))
inspect(fsets[1:4])
inspect(itemUnion(items(fsets[1:2]), items(fsets[3:4])))
inspect(itemSetdiff(items(fsets[1:2]), items(fsets[3:4])))
inspect(itemIntersect(items(fsets[1:2]), items(fsets[3:4])))
```

---

itemsets-class *Class itemsets — A Set of Itemsets*

Description

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

Details

Itemsets are usually created by calling an association rule mining algorithm like `apriori`. Itemsets store the items as an object of class `itemMatrix`.

To create itemsets manually, the itemMatrix for the items of the itemsets can be created using `itemCoding`. An example is in the Example section below.

Mined itemsets sets typically contain several interest measures accessible with the `quality` method. Additional measures can be calculated via `interestMeasure`.

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",...)`

or by using the constructor function

`itemsets(items,itemLabels,quality = data.frame())`.

`items` need to be a list describing the items (using labels or item ids) and `itemLabels` needs to be a vector of all possible item labels (character) or a transactions object to copy the item coding (see `itemCoding` for details).

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.

nitems signature(x = "itemsets"); number of all possible items in the binary matrix representation of the object.

summary signature(object = "itemsets")

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

Author(s)

Michael Hahsler

See Also

associations-class, [-methods, apriori, c, duplicated, eclat, inspect, is.maximal, itemCoding length, match, sets, size, subset, 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")

## Manually create itemsets with the item coding in the Adult dataset
## and calculate some interest measures
twoitemsets <- itemsets(
  items = list(
    c("age=Young", "relationship=Unmarried"),
    c("age=Old")
  ), itemLabels = Adult)

quality(twoitemsets) <- data.frame(support = interestMeasure(twoitemsets,
  measure = c("support"), transactions = Adult))

inspect(twoitemsets)
```

`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.

Author(s)

Michael Hahsler

LIST *List Representation for Objects Based on Class 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.

Author(s)

Michael Hahsler

See Also

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

Examples

```
data(Adult)

### default coercions (same as as(Adult[1:5], "list"))
LIST(Adult[1:5])

### coercion without item decoding
LIST(Adult[1:5], decode = FALSE)
```

---

match *Value Matching*

Description

Provides the generic function match and the S4 methods for associations, transactions and itemMatrices. 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 the items in the itemsets (left operand) with the items in the table (right operand).

arules defines additional binary operators for matching itemsets: %pin% uses *partial* matching on the table; %ain% itemsets have to match/include *all* items in the table; %oin% itemsets can *only* match/include the items in the table. The binary matching operators or often used in subset.

Usage

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

x %in% table
x %pin% table
x %ain% table
x %oin% table
```

Arguments

x	an object of class itemMatrix, transactions or associations.
table	a set of associations or transactions to be matched against.
nomatch	the value to be returned in the case when no match is found.
incomparables	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%, %pin%, %ain%, %oin%: A logical vector, indicating if a match was located for each element of x.

Author(s)

Michael Hahsler

See Also

subset, 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])

## for transactions we can also match directly with itemLabels.
## Find in the first 10 transactions the ones which
## contain age=Middle-aged (see help page for class itemMatrix)
Adult[1:10] %in% "age=Middle-aged"

## find all transactions which contain items that partially match "age=" (all here).
Adult[1:10] %pin% "age="

## find all transactions that only include the item "age=Middle-aged" (none here).
Adult[1:10] %oin% "age=Middle-aged"

## find al transaction which contain both items "age=Middle-aged" and "sex=Male"
Adult[1:10] %ain% c("age=Middle-aged", "sex=Male")
```

merge *Adding Items to Data*

Description

Provides the generic function merge and the S4 methods for itemMatrix and transactions. The methods are used to add new items to existing data.

Usage

```
merge(x, y, ...)
```

Arguments

x	an object of class itemMatrix or transactions.
y	an object of the same class as x (or something which can be coerced to that class).
...	further arguments; unused.

Value

Returns a new object of the same class as x with the items in y added.

Author(s)

Michael Hahsler

See Also

transactions-class, itemMatrix-class, addComplement

Examples

```
data("Groceries")

## create a random item as a matrix
randomItem <- sample(c(TRUE, FALSE), size=length(Groceries),replace=TRUE)
randomItem <- as.matrix(randomItem)
colnames(randomItem) <- "random item"
head(randomItem, 3)

## add the random item to Groceries
g2 <- merge(Groceries, randomItem)
nitems(Groceries)
nitems(g2)
inspect(head(g2,3))
```

Mushroom *Mushroom Data Set*

Description

The Mushroom data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. It contains information about 8124 mushrooms (transactions). 4208 (51.8%) are edible and 3916 (48.2%) are poisonous. The data contains 22 nominal features plus the class attribute (edible or not). These features were translated into 114 items.

Usage

```
data(Mushroom)
```

Format

Object of class transactions.

Author(s)

Michael Hahsler

Source

The data set was obtained from the UCI Machine Learning Repository at https://archive.ics.uci.edu/ml/datasets/Mushroom.

References

Alfred A. Knopf (1981). Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms. G. H. Lincoff (Pres.), New York.

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

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

Value

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

Author(s)

Michael Hahsler

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.

Author(s)

Michael Hahsler

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

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

Details

The function generates a nitems times ntrans transaction database.

Currently two simulation methods are implemented:

method "independent" **(see Hahsler et al., 2006)** All items are treated as independent. The transaction size is determined by $rpois(lambda - 1) + 1$, where lambda can be specified (defaults to 3). Note that one subtracted from lambda and added to the size to avoid empty transactions. The items in the transactions are randomly chosen using the numeric probability 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.

Author(s)

Michael Hahsler

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,
   lambda = 5, iProb = seq(0.2,0.0001, length=200))

## size distribution
summary(size(trans))

## 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 (we just create 100 transactions here to speed things up).
patterns <- random.patterns(nItems = 1000)
summary(patterns)

trans2 <- random.transactions(nItems = 1000, nTrans = 100,
   method = "agrawal", patterns = patterns)
image(trans2)

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

read.PMML *Read and Write PMML*

Description

This function reads and writes PMML representations (version 4.1) of associations (itemsets and rules).

Usage

```
write.PMML(x, file)
read.PMML(file)
```

Arguments

x	a rules or itemsets object.
file	name of the PMML file (for read.PMML also a XML root node can be supplied).

Details

Write delegates to package pmml.

Author(s)

Michael Hahsler

References

PMML 4.4 - Association Rules. http://dmg.org/pmml/v4-4/AssociationRules.html

See Also

pmml.

Examples

```
data("Groceries")

rules <- apriori(Groceries, parameter=list(support=0.001))
rules <- head(rules, by="lift")
rules

### save rules as PMML
write.PMML(rules, file = "rules.xml")

### read rules back
rules2 <- read.PMML("rules.xml")
rules2
```

```
### compare rules
inspect(rules[1])
inspect(rules2[1])

### clean up
unlink("rules.xml")
```

`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"),
                  header = FALSE, sep = "",
                  cols = NULL, rm.duplicates = FALSE,
                  quote = "\"'", skip = 0,
                  encoding = "unknown")
```

Arguments

`file`	the file name or connection.
`format`	a character string indicating the format of the data set. One of `"basket"` or `"single"`, can be abbreviated.
`header`	a logical value indicating whether the file contains the names of the variables as its first line.
`sep`	a character string specifying how fields are separated in the data file. The default (`""`) splits at whitespaces.
`cols`	For the 'single' format, `cols` is a numeric or character vector of length two giving the numbers or names of the columns (fields) with the transaction and item ids, respectively. If character, the first line of `file` is assumed to be a header with column names. For the 'basket' format, `cols` can be a numeric scalar giving the number of the column (field) with the transaction ids. If `cols = NULL`, the data do not contain transaction ids.
`rm.duplicates`	a logical value specifying if duplicate items should be removed from the transactions.
`quote`	a list of characters used as quotes when reading.
`skip`	number of lines to skip in the file before start reading data.
`encoding`	character string indicating the encoding which is passed to `readLines` or `scan` (see `Encoding`).

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.

Author(s)

Michael Hahsler and Kurt Hornik

See Also

transactions-class

Examples

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

## read demo data (skip the comment in the first line)
tr <- read.transactions("demo_basket.txt", format = "basket", sep=",", skip = 1)
inspect(tr)
## make always sure that the items were properly separated
itemLabels(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.txt")

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

## create a demo file using single format with column headers
data <- paste(
  "item_id;trans_id",
```

```
  "item1;trans1",
  "item1;trans2",
  "item2;trans2",
  sep ="\n")
cat(data)
write(data, file = "demo_single.txt")

## read demo data
tr <- read.transactions("demo_single.txt", format = "single",
  header = TRUE, sep = ";", cols = c("trans_id","item_id"))
inspect(tr)

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

`ruleInduction` *Rule Induction from Itemsets*

Description

Provides the generic function and the needed S4 method to induce all rules which can be generated by the given set of itemsets from a transactions dataset. This method can be used to create closed association rules.

Usage

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

Arguments

`x`	the set of itemsets from which rules will be induced.
`...`	further arguments.
`transactions`	the transaction dataset used to mine the itemsets. Can be omitted if `x` contains a lattice (complete set) of frequent itemsets together with their support counts.
`confidence`	a numeric value giving the minimum confidence for the rules.
`control`	a named list with elements `method` indicating the method (`"apriori"` or `"ptree"`), and the logical arguments `reduce` and `verbose` 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, for `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.

For `transactions`, a set different to the data used for creating the original itemsets can be used, however, the new set has to conform in terms of items and their order.

This method can be used to produce closed association rules defined by Pei et al. (2000) as rules $X -> Y$ where both X and Y are closed frequent itemsets. See Example section for code.

Value

An object of class `rules`.

Author(s)

Christian Buchta and Michael Hahsler

References

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

Jian Pei, Jiawei Han, Runying Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 2000).

See Also

`itemsets-class`, `rules-class` `transactions-class`

Examples

```
data("Adult")

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

```
## use rule induction to produce all closed association rules
closed_rules <- ruleInduction(closed_is, Adult,
  control = list(verbose = TRUE))

## X&Y are already closed, check that X is also closed
closed_rules[is.element(lhs(closed_rules), items(closed_is))]

## inspect the resulting closed rules
summary(closed_rules)
inspect(head(closed_rules, by = "lift"))

## use lattice of frequent itemsets
ec  <- eclat(Adult, parameter = list(support = 0.4))
rec <- ruleInduction(ec)
rec
inspect(head(rec))
```

`rules-class` *Class rules — A Set of Rules*

Description

The `rules` class represents a set of rules.

Details

Rules are usually created by calling an association rule mining algorithm like `apriori`. Rules store the LHS and the RHS separately as objects of class `itemMatrix`.

To create rules manually, the itemMatrix for the LHS and the RHS of the rules can be created using `itemCoding`. Note the two matrices need to have the itemLabels (i.e., columns of the sparse matrix) in the same order. An example is in the Example section below.

Mined rule sets typically contain several interest measures accessible with the `quality` method. Additional measures can be calculated via `interestMeasure`.

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",...)`

or by using the constructor function

`rules(lhs,rhs,itemLabels,quality = data.frame())`.

`lhs` and `rhs` need to be a list describing the items (using labels or item ids) and `itemLabels` needs to be a vector of all possible item labels (character) or a transactions object to copy the item coding (see `itemCoding` for details).

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; typically contains measures like support, confidence and count (i.e., the absolute support count)

Extends

Class `associations`, directly.

Methods

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

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. Technically this method produces the same as the result of method `items()`, but wrapped into an `itemsets` object with support information.

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`

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.

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`

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)

nitems `signature(x = "rules")`; number of all possible items in the binary matrix representation of the object.

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")`

Author(s)

Michael Hahsler

See Also

associations-class, [-methods, apriori, c, duplicated, inspect, itemCoding length, match, sets, size, subset,

Examples

```
data("Adult")

## Mine rules
rules <- apriori(Adult, parameter = list(support = 0.3))
rules

## 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 the top 3 support rules
inspect(head(rules.sub, n = 3, by = "support"))

## Display the first 3 rules
inspect(rules.sub[1:3])

## Get labels for the first 3 rules
labels(rules.sub[1:3])
labels(rules.sub[1:3], itemSep = " + ", setStart = "", setEnd="",
  ruleSep = " ---> ")

## Manually create rules using the item coding in Adult and calculate some interest measures
twoRules <- rules(
  lhs = list(
    c("age=Young", "relationship=Unmarried"),
    c("age=Old")
  ),
  rhs = list(
    c("income=small"),
    c("income=large")
  ),
  itemLabels = Adult
)

quality(twoRules) <- interestMeasure(twoRules,
  measure = c("support", "confidence", "lift"), transactions = Adult)

inspect(twoRules)
```

---

sample *Random Samples and Permutations*

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.
...	further arguments.

Value

An object of the same class as x.

Author(s)

Michael Hahsler

See Also

associations-class, transactions-class, itemMatrix-class.

Examples

```
data("Adult")

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

---

setOperations *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.
...	Other arguments are unused.

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.

Author(s)

Michael Hahsler

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 *Number of Items*

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.

Author(s)

Michael Hahsler

See Also

`itemMatrix-class`, `transactions-class`

Examples

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

sort *Sort Associations*

Description

Provides the method sort 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", order = FALSE, ...)

## S4 method for signature 'associations'
head(x, n = 6L, by = NULL, decreasing = TRUE, ...)
## S4 method for signature 'associations'
tail(x, n = 6L, by = NULL, decreasing = TRUE, ...)
```

Arguments

x	an object to be sorted.
decreasing	a logical. Should the sort be increasing or decreasing? (default is decreasing)
na.last	na.last is not supported for associations. NAs are always put last.
by	a character string specifying the quality measure stored in x to be used to sort x. If a vector of character strings is specified then the additional strings are used to sort x in case of ties.
order	should a order vector be returned instead of the sorted associations?
n	a single integer indicating the number of associations returned.
...	Further arguments are ignored.

Details

sort is relatively slow for large sets of associations since it has to copy and rearrange a large data structure. Note that sorting creates a second copy of the set of associations which can be slow and memory consuming for large sets. With order = TRUE a integer vector with the order is returned instead of the reordered associations.

If only the top n associations are needed then head using by performs this faster than calling sort and then head since it does it without copying and rearranging all the data. tail works in the same way.

Value

An object of the same class as x.

Author(s)

Michael Hahsler

See Also

associations-class

Examples

```
data("Adult")

## Mine rules with APRIORI
rules <- apriori(Adult, parameter = list(supp = 0.6))

rules_by_lift <- sort(rules, by = "lift")

inspect(head(rules))
inspect(head(rules_by_lift))

## A faster/less memory consuming way to get the top 5 rules according to lift
## (see Details section)
inspect(head(rules, n = 5, by = "lift"))
```

---

subset *Subsetting Itemsets, Rules and Transactions*

Description

Provides the generic function subset and S4 methods to subset associations or transactions (itemMatrix) 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

`x`	object to be subsetted.
`subset`	logical expression indicating elements to keep.
`...`	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), `%oin%` (select only itemsets matching *only* the 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.

Author(s)

Michael Hahsler

See Also

`%in%`, `%pin%`, `%ain%`, `%oin%`, `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"))
```

SunBai *The SunBai Data Set*

Description

A small example database for weighted association rule mining provided as an object of class transactions.

Usage

```
data(SunBai)
```

Details

The data set contains the example database described in the paper by K. Sun and F.Bai for illustration of the concepts of weighted association rule mining. `weight` stored as transaction information denotes the transaction weights obtained using the HITS algorithm.

Source

K. Sun and F. Bai (2008). Mining Weighted Association Rules without Preassigned Weights. *IEEE Transactions on Knowledge and Data Engineering*, 4 (30), 489–495.

See Also

Class transactions, method transactionInfo, function hits.

Examples

```
data(SunBai)
summary(SunBai)
inspect(SunBai)

transactionInfo(SunBai)
```

support *Support Counting for Itemsets*

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"), weighted = FALSE, control = NULL)
## S4 method for signature 'associations'
support(x, transactions,
    type= c("relative", "absolute"), weighted = FALSE, control = NULL)
```

Arguments

x	the set of itemsets for which support should be counted.
...	further arguments are passed on.
transactions	the transaction data set used for mining.
type	a character string specifying if "relative" support or "absolute" support (counts) are returned for the itemsets in x. (default: "relative")
weighted	should support be weighted by transactions weights stored as column "weight" in transactionInfo?
control	a named list with elements method indicating the method ("tidlists" or "ptree"), and the logical arguments reduce and verbose 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.

Author(s)

Michael Hahsler and Christian Buchta

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)
```

---

supportingTransactions

Supporting Transactions

Description

Find transactions which support each of a set of associations and return this information as a transaction ID list.

Usage

```
supportingTransactions(x, transactions, ...)
```

Arguments

x	a set of associations (itemsets, rules, etc.)
transactions	an object of class transactions used to mine the associations in x.
...	currently unused.

Details

The supporting transactions are all transactions of which the itemset representing the association is a subset of.

Value

An object of class tidLists containing one transaction ID list per association in x.

Author(s)

Michael Hahsler

See Also

tidLists-class

Examples

```
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")
)
data <- as(data, "transactions")

## mine itemsets
f <- eclat(data, parameter = list(support = .2, minlen=3))
inspect(f)

## find supporting Transactions
st <- supportingTransactions(f, data)
st

as(st, "list")
```

---

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 = "list")`

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

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

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

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

coerce `signature(from = "itemMatrix", to = "tidLists")`; this also coerces from `transactions`.

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

c `signature(x = "tidLists")`; combine.

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

dimnames, rownames, colnames `signature(x = "transactions")`; returns row (items/itemsets) and column (transactionIDs if available) names.

labels `signature(x = "transactions")`; returns the labels for the itemsets in each transaction (see `itemMatrix`).

inspect inspect the transaction ID lists.

itemInfo returns the slot `itemInfo`.

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

labels `signature(x = "transactions")`; returns the labels (transaction IDs).

show `signature(object = "tidLists")`

summary `signature(object = "tidLists")`

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

Author(s)

Michael Hahsler

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")
  )
data <- as(data, "transactions")
data

## convert transactions to transaction ID lists
tl <- as(data, "tidLists")
tl

inspect(tl)
dim(tl)
dimnames(tl)

## inspect visually
image(tl)

## mine itemsets with transaction ID lists
f <- eclat(data, parameter = list(support = 0, tidLists = TRUE))
tl2 <- tidLists(f)
inspect(tl2)
```

---

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

Description

The transactions class represents transaction data used for mining itemsets or rules.

Details

Transactions are a direct extension of class itemMatrix to store a binary incidence matrix, item labels, and optionally transaction IDs and user IDs.

Transactions can be created from a list containing transactions or a matrix or data.frames using

- the constructor function `transactions(x,itemLabels = NULL,transactionInfo = NULL)`, or
- S4 coercion with `as(x,"transactions")`).

`itemLabels` and `transactionInfo` are by default created from information in `x` (e.g., from row and column names). In the constructor function, the user can specify for itemLabels a vector of all possible item labels (character) or another transactions object to copy the item coding (see `itemCoding` for details).

Note that you will need to prepare your data first (see coercion methods in the Methods Section and the Example Section below for details on the needed format).

Continuous variables: Association rule mining can only use items and does not work with continuous variables. Continuous variables need to be discretized first. An item resulting from discretization might be *age>18* and the column contains only `TRUE` or `FALSE`. Alternatively it can be a factor with levels *age<=18*, *50=>age>18* and *age>50*. These will be automatically converted into 3 items, one for each level. Have a look at the function `discretize` for automatic discretization.

Logical variables: A logical variable describing a person could be *tall* indicating if the person is tall using the values `TRUE` and `FALSE`. The fact that the person is tall would be encoded in the transaction containing the item *tall* while not tall persons would not have this item. Therefore, for logical variables, the `TRUE` value is converted into an item with the name of the variable and for the `FALSE` values no item is created.

Factors: The function also can convert columns with nominal values (i.e., factors) into a series of binary items (one for each level constructed as `` `variable name`=`level` ``). Note that nominal variables need to be encoded as factors (and not characters or numbers). This can be done with

`data[,"a_nominal_var"] <-factor(data[,"a_nominal_var"])`.

Complete examples for how to prepare data can be found in the man pages for `Income` and `Adult`.

Transactions are represented as sparse binary matrices of class `itemMatrix`. If you work with several transaction sets at the same time, then the encoding (order of the items in the binary matrix) in the different sets is important. See `itemCoding` to learn how to encode and recode transaction sets.

Objects from the Class

Objects are created by coercion from objects of other classes (see Examples section) or by calls of the form

`new("transactions",...)`

or by using the constructor function

`transactions(x,itemLabels = NULL,transactionInfo = NULL,format = "wide",cols = NULL)`.

Format "wide" is a regular data.frame where each row contains an object. Format "long" is a data.frame with one column with transaction IDs and one with an item. cols is a numeric or character vector of length two giving the numbers or names of the columns (fields) with the transaction and item ids, respectively.

See Examples Section for creating transactions from data.

Slots

`itemsetInfo`: a data.frame with one row per transaction (each transaction is considered an itemset). The data.frame can hold columns with additional information, e.g., transaction IDs or user IDs for each transaction. **Note:** this slot is inherited from class `itemMatrix`, but should be accessed in transactions with the method `transactionInfo()`.

`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 column names are used as item labels and the row names are stores as transaction IDs.

coerce `signature(from = "transactions", to = "matrix")`; coerces the transactions data set into a binary incidence matrix.

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.

coerce `signature(from = "transactions", to = "list")`; coerces the transactions data set into a list of transactions. Each transaction is a vector of character strings (names of the contained items).

coerce `signature(from = "data.frame", to = "transactions")`; recodes the data frame containing only categorical variables (factors) or logicals all into a binary transaction data set. For binary variables only TRUE values are converted into items and the item label is the variable name. For factors, a dummy item for each level is automatically generated. Item labels are generated by concatenating variable names and levels with '`"="`'. The original variable names and levels are stored in the itemInfo data frame as the components `variables` and `levels`. Note that `NA`s are ignored (i.e., do not generate an item).

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`.

coerce `signature(from = "ngCMatrix", to = "transactions")`; Note that the data is stored transposed in the ngCMatrix. Items are stored as rows and transactions are columns!

dimnames, rownames, colnames `signature(x = "transactions")`; returns row (transactionID) and column (item) names.

items `signature(x = "transactions")`; returns the items in the transactions as an `itemMatrix`.

labels `signature(x = "transactions")`; returns the labels for the itemsets in each transaction (see `itemMatrix`).

transactionInfo<- `signature(x = "transactions")`; replaces the transaction information with a new data.frame.

transactionInfo `signature(x = "transactions")`; returns the transaction information as a data.frame.

toLongFormat `signature(object = "transactions")`; convert the transactions to long format (a data.frame with two columns, tid and item). Column names can be specified as a character vector of length 2 called `cols`.

show `signature(object = "transactions")`

summary `signature(object = "transactions")`

Author(s)

Michael Hahsler

See Also

`[-methods`, `discretize`, `LIST`, `write`, `c`, `image`, `inspect`, `itemCoding`, `read.transactions`, `random.transactions`, `sets`, `itemMatrix-class`

Examples

```
## Example 1: creating transactions form a list (each element is a transaction)
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

## Use the constructor to create transactions
## Note: S4 coercion does the same trans1 <- as(a_list, "transactions")
trans1 <- transactions(a_list)
trans1

## Analyze the transactions
summary(trans1)
image(trans1)

## Example 2: creating transactions from a 0-1 matrix with 5 transactions (rows) and
##            5 items (columns)
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 item names (columns) and transaction labels (rows)
colnames(a_matrix) <- c("a", "b", "c", "d", "e")
```

```
rownames(a_matrix) <- paste("Tr", c(1:5), sep = "")

a_matrix

## Create transactions
trans2 <- transactions(a_matrix)
trans2
inspect(trans2)

## Example 3: creating transactions from data.frame (wide format)
a_df <- data.frame(
age   = as.factor(c(6, 8, NA, 9, 16)),
grade = as.factor(c("A", "C", "F", NA, "C")),
  pass  = c(TRUE, TRUE, FALSE, TRUE, TRUE))
## Note: factors are translated differently than logicals and NAs are ignored
a_df

## Create transactions
trans3 <- transactions(a_df)
inspect(trans3)

## Note that coercing the transactions back to a data.frame does not recreate the
## original data.frame, but represents the transactions as sets of items
as(trans3, "data.frame")

## Example 4: creating transactions from a data.frame with
## transaction IDs and items (long format)
a_df3 <- data.frame(
  TID =  c( 1,   1,   2,   2,   2,   3 ),
  item = c("a", "b", "a", "b", "c", "b")
)
a_df3
trans4 <- transactions(a_df3, format = "long", cols = c("TID", "item"))
trans4
inspect(trans4)

## convert transactions back into long format.
toLongFormat(trans4)

## Example 5: create transactions from a dataset with numeric variables
## using discretization.
data(iris)

irisDisc <- discretizeDF(iris)
head(irisDisc)

trans5 <- transactions(irisDisc)
trans5
inspect(head(trans5))

## Note, creating transactions without discretizing numeric variables will apply the
## default discretization and also create a warning.
```

```
## Example 6: create transactions manually (with the same item coding as in trans5)
trans6 <- transactions(
  list(
    c("Sepal.Length=[4.3,5.4)", "Species=setosa"),
    c("Sepal.Length=[4.3,5.4)", "Species=setosa")
  ), itemLabels = trans5)
trans6

inspect(trans6)
```

---

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.

Author(s)

Michael Hahsler

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
```

---

weclat *Mining Associations from Weighted Transaction Data with Eclat (WARM)*

Description

Find frequent itemsets with the Eclat algorithm. This implementation uses optimized tidlist joins and transaction weights to implement weighted association rule mining (WARM).

Usage

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

Arguments

`data`	an object that can be coerced into an object of class `transactions`.
`parameter`	an object of class `ASparameter` (default values: `support = 0.1`, `minlen = 1L`, and `maxlen = 5L`) or a named list with corresponding components.
`control`	an object of class `AScontrol` (default values: `verbose = TRUE`) or a named list with corresponding components.

Details

Transaction weights are stored in the transaction as a column called `weight` in `transactionInfo`.

The weighted support of an itemset is the sum of the weights of the transactions that contain the itemset. An itemset is frequent if its weighted support is equal or greater than the threshold specified by `support` (assuming that the weights sum to one).

Note that ECLAT only mines (weighted) frequent itemsets. Weighted association rules can be created using `ruleInduction`.

Value

Returns an object of class `itemsets`. Note that weighted support is returned in `quality` as column `support`.

Note

The C code can be interrupted by CTRL-C. This is convenient but comes at the price that the code cannot clean up its internal memory.

Author(s)

Christian Buchta

References

G.D. Ramkumar, S. Ranka, and S. Tsur (1998). Weighted Association Rules: Model and Algorithm, *Proceedings of ACM SIGKDD*

See Also

Class transactions, function ruleInduction, eclat

Examples

```
## Example 1: SunBai data
data(SunBai)
SunBai

## weights are stored in transactionInfo
transactionInfo(SunBai)

## mine weighted support itemsets using transaction support in SunBai
s <- weclat(SunBai, parameter = list(support = 0.3),
       control = list(verbose = TRUE))
inspect(sort(s))

## create rules using weighted support (satisfying a minimum
## weighted confidence of 90%).
r <- ruleInduction(s, confidence = .9)
inspect(r)

## Example 2: Find association rules in weighted data
trans <-  list(
    c("A", "B", "C", "D", "E"),
    c("C", "F", "G"),
    c("A", "B"),
    c("A"),
    c("C", "F", "G", "H"),
    c("A", "G", "H")
)

weight <- c(5, 10, 6, 7, 5, 1)

## convert list to transactions
trans <-  transactions(trans)

## add weight information
```

```
transactionInfo(trans) <- data.frame(weight = weight)
inspect(trans)

## mine weighed support itemsets
s <- weclat(trans, parameter = list(support = 0.3),
       control = list(verbose = TRUE))
inspect(sort(s))

## create association rules
r <- ruleInduction(s, confidence = .5)
inspect(r)
```

`write` *Write Transactions or Associations to a File*

Description

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

Usage

```
write(x, file = "",...)
## S4 method for signature 'transactions'
write(x, file="", format = c("basket", "single"),
                 sep=" ", quote=TRUE, ...)
## S4 method for signature 'associations'
write(x, file="", sep=" ", quote=TRUE, ...)
```

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.
`format`	format to write transactions.
`sep`	the field separator string. Values within each row of x are separated by this string. Use `quote=TRUE` and `sep=","` for saving data as in csv format.
`quote`	a logical value. Quote fields?
`...`	further arguments passed on to `write.table` or `write`. Use `fileEncoding` to set the encoding used for writing the file.

Details

For associations (rules and itemsets) `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.

Transactions can be saved in basket (one line per transaction) or in single (one line per item) format.

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) via `write.PMML`. This requires package **pmml**.

Author(s)

Michael Hahsler

See Also

`read.transactions` for reading transactions from a file, `read.PMML` and `write.PMML` for reading/writing associations in PMML format, `write.table` (in **base**), `transactions-class`, `associations-class`

Examples

```
data("Epub")

## write the formated transactions to screen (basket format)
write(head(Epub))

## write the formated transactions to screen (single format)
write(head(Epub), format="single")

## write the formated result to file in CSV format
write(Epub, file = "data.csv", format="single", sep = ",")

## write rules in CSV format
rules <- apriori(Epub, parameter=list(support=0.0005, conf=0.8))
write(rules, file = "data.csv", sep = ",")

unlink("data.csv") # 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.

Author(s)

Michael Hahsler

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