Package ‘arules’

November 29, 2023

Version 1.7-7

Date 2023-11-28

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 (>= 4.0.0), Matrix (>= 1.4-0)

Imports stats, methods, generics, graphics, utils

Suggests pmml, XML, proxy, 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.

Encoding UTF-8

RoxygenNote 7.2.3

Collate 'AAADefs.R' 'Adult.R' 'itemMatrix.R' 'associations.R' 'transactions.R' 'tidLists.R' 'itemsets.R' 'rules.R' 'DATAFRAME.R' 'Epub.R' 'Groceries.R' 'Income.R' 'LIST.R' 'Matrix.R' 'Mushroom.R' 'SunBai.R' 'abbreviate.R' 'addComplement.R' 'appearance.R' 'apriori.R' 'c.R' 'check_installed.R' 'confint.R' 'control.R' 'coverage.R' 'crossTable.R' 'discretize.R' 'dissimilarity.R' 'duplicated.R' 'eclat.R' 'extract.R' 'fim4r.R' 'hierarchy.R' 'image.R' 'inspect.R' 'interestMeasures.R' 'is.closed.R' 'is.generator.R' 'is.maximal.R' 'is.redundant.R' 'is.significant.R' 'is.superset.R' 'itemCoding.R' 'itemFrequency.R' 'match.R' 'merge.R' 'parameter.R' 'plot.R' 'pmml.R' 'predict.R' 'random.transactions.R' 'read_write.R' 'ruleInduction.R' 'sample.R' 'sets.R' 'setsItemwise.R' 'size.R' 'sort.R' 'subset.R' 'support.R' 'supportingTransactions.R' 'unique.R' 'warm.R'

NeedsCompilation yes

Author Michael Hahsler [aut, cre, cph] (<https://orcid.org/0000-0003-2716-1405>), Christian Buchta [aut, cph], Bettina Gruen [aut, cph], Kurt Hornik [aut, cph] (<https://orcid.org/0000-0003-4198-9911>), Christian Borgelt [ctb, cph], Ian Johnson [ctb], Makhlouf Ledmi [ctb]

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

Repository CRAN

Date/Publication 2023-11-29 09:00:07 UTC

R topics documented:

abbreviate *Abbreviate 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 `base::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 'transactions'
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 'tidLists'
abbreviate(names.arg, minlength = 4, ..., method = "both.sides")
```

Arguments

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

Author(s)

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

See Also

`base::abbreviate()`

Other associations functions: `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

Other itemMatrix and transactions functions: `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

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 a method for transactions to add complement items. That is, it adds an artificial item to each transaction which does not contain the original item. Such items are also called negative items (Antonie et al, 2014).

Usage

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

## S4 method for signature 'transactions'
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

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**.

Format

`Adult` is an object of class transactions with 48842 transactions and 115 items. See below for details.

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://archive.ics.uci.edu/

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 methods to compute and return a similarity matrix with the affinities between items for a set itemsets stored in a matrix or in transactions via its superclass itemMatrix.

Usage

```
affinity(x)

## S4 method for signature 'matrix'
affinity(x)

## S4 method for signature 'itemMatrix'
affinity(x)
```

Arguments

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

Details

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

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

where $supp(.)$ is the support measure. Note that affinity is equivalent to 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

Other proximity classes and functions: `dissimilarity()`, `predict()`, `proximity-classes`

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).

Details

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

Slots

`labels` 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.

`set` used internally.

`items` used internally.

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 transactions.

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.

Coercions

- `as("NULL", "APappearance")`
- `as("list", "APappearance")`

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

Other mining algorithms: `AScontrol-classes`, `ASparameter-classes`, `apriori()`, `eclat()`, `fim4r()`, `ruleInduction()`, `weclat()`

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 the Apriori Algorithm*

Description

Mine frequent itemsets, association rules or association hyperedges using the Apriori algorithm.

Usage

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

Arguments

`data`	object of class transactions. Any data structure which can be coerced into transactions (e.g., a binary matrix, a data.frame or a tibble) can also be specified and will be internally coerced to transactions.
`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 for convenience added to the parameter list.

Details

The Apriori algorithm (Agrawal et al, 1993) employs level-wise search for frequent itemsets. The used C implementation of Apriori by Christian Borgelt (2003) includes some improvements (e.g., a prefix tree and item sorting).

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.

Apriori only creates rules with one item in the RHS (Consequent). The default value in APparameter for `minlen` is 1. This meains 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

Other mining algorithms: `APappearance-class`, `AScontrol-classes`, `ASparameter-classes`, `eclat()`, `fim4r()`, `ruleInduction()`, `weclat()`

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)

# since ... gets automatically added to parameter, we can also write the
#  same call shorter:
apriori(Adult, supp = 0.5, conf = 0.9, target = "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).

Slots

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

`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`)

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

Available Slots by Subclass

- `APcontrol`: `filter`, `tree`, `heap`, `memopt`, `load`, `sort`, `verbose`
- `ECcontrol`: `sparse`, `sort`, `verbose`

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 the `control` argument for `apriori()` or `eclat()`, default values can be replaced with the values in the list.

Objects can also be created via coercion.

Coercions

- `as("NULL", "APcontrol")`
- `as("list", "APcontrol")`
- `as("NULL", "ECcontrol")`
- `as("list", "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

Other mining algorithms: `APappearance-class`, `ASparameter-classes`, `apriori()`, `eclat()`, `fim4r()`, `ruleInduction()`, `weclat()`

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 `apriori()` (`APparameter`) or `eclat()` (`ECparameter`).

Slots

`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. Partial names are matched. Available targets are:

- `"frequent itemsets"`
- `"maximally frequent itemsets"`
- `"generator 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`)

`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 the support of the LHS (also called coverage of the rule) is returned as support. The minimum support threshold is applied to this support. (default: `TRUE`)

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

`tidLists` a logical indicating whether `eclat()` should return also a list of supporting transactions IDs. (default: `FALSE`)

Available Slots by Subclass

- `APparameter`: `confidence`, `minval`, `smax`, `arem`, `aval`, `originalSupport`, `maxtime`, `support`, `minlen`, `maxlen`, `target`, `ext`
- `ECparameter`: `tidLists`, `support`, `minlen`, `maxlen`, `target`, `ext`

Objects from the Class

A suitable default parameter object will be automatically created by `apriori()` or `eclat()`. By specifying a named list (names equal to slots) as `parameter` argument for `apriori()` or `eclat()`, the default values can be replaced with the values in the list.

Objects can also be created via coercion.

Coercions

- `as("NULL", "APparameter")`
- `as("list", "APparameter")`
- `as("NULL", "ECparameter")`
- `as("list", "ECparameter")`

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

Other mining algorithms: `APappearance-class`, `AScontrol-classes`, `apriori()`, `eclat()`, `fim4r()`, `ruleInduction()`, `weclat()`

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 defines some common methods for its subclasses.

Usage

```
## S4 method for signature 'associations'
quality(x)

## S4 replacement method for signature 'associations'
quality(x) <- value

## S4 method for signature 'associations'
info(x)

## S4 replacement method for signature 'associations'
info(x) <- value

## 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, ...)

## S4 method for signature 'associations'
items(x)

## S4 method for signature 'associations'
length(x)

## S4 method for signature 'associations'
labels(object)
```

Arguments

x, object	the object.
value	the replacement value.
n	number of elements
by	sort by this interest measure
decreasing	sort in decreasing order?
...	further arguments.

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()`.

See Sections Functions and See Also to see all available methods.

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

Functions

- `quality(associations)`: returns the quality data.frame.
- `quality(associations) <- value`: replaces the quality data.frame. The lengths of the vectors in the data.frame have to equal the number of associations in the set.
- `info(associations)`: returns the info list.
- `info(associations) <- value`: replaces the info list.
- `head(associations)`: returns the first n associations.
- `tail(associations)`: returns the last n associations.
- `items(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.
- `length(associations)`: dummy method. This method has to be implemented by all subclasses of associations and return the number of elements in the association.
- `labels(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.

Slots

`quality` a data.frame

`info` a list

Objects from the Class

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

Author(s)

Michael Hahsler

See Also

Subclasses: rules, itemsets

Other associations functions: `abbreviate()`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

c *Combining Association and Transaction Objects*

Description

Provides the methods to combine several associations or transactions objects into a single object.

Usage

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

## S4 method for signature 'transactions'
c(x, ..., recursive = FALSE)

## S4 method for signature 'tidLists'
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.

Details

Combining arules objects is done by combining the rows of itemMatrix objects representing the associations or transactions.

Note that `c()` can result in duplicates. Use `union()` rather than `c()` to combine several mined itemsets or rules into a single set without duplicates.

Value

An object of the same class as `x`.

Author(s)

Michael Hahsler

See Also

Other associations functions: abbreviate(), associations-class, duplicated(), extract, inspect(), is.closed(), is.generator(), is.maximal(), is.redundant(), is.significant(), is.superset(), itemsets-class, match(), rules-class, sample(), sets, size(), sort(), unique()

Other itemMatrix and transactions functions: abbreviate(), crossTable(), duplicated(), extract, hierarchy, image(), inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), itemMatrix-class, match(), merge(), random.transactions(), sample(), sets, size(), supportingTransactions(), tidLists-class, transactions-class, unique()

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 Interest Measures for Association Rules*

Description

Defines a method to compute confidence intervals for interest measures for association rules.

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 (see `interestMeasure()`). `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.
`replications`	number of replications for method `"simulation"`. Ignored for other methods.
`smoothCounts`	pseudo count for addaptive smoothing (Laplace smoothing). Often a pseudo counts of .5 is used for smoothing (see Detail Section).
`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 named `"LL"` and `"UL"` with the interval boundaries. The matrix has the following 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`	description 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

Other interest measures: `coverage()`, `interestMeasure()`, `is.redundant()`, `is.significant()`, `support()`

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 a method to calculate the coverage (support of the left-hand-side) of rules.

Usage

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

## S4 method for signature 'rules'
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 $X => Y$, i.e., $supp(X)$. It represents a measure of to how often the rule can be applied.

Coverage can be quickly calculated from the rule's 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 `interestMeasure()`.

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

Other interest measures: `confint()`, `interestMeasure()`, `is.redundant()`, `is.significant()`, `support()`

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 a 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).
`...`	additional arguments.
`measure`	measure to return. Default is co-occurrence counts.
`sort`	sort the items by support.

Value

A symmetric matrix of n x 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

Other itemMatrix and transactions functions: abbreviate(), c(), duplicated(), extract, hierarchy, image(), inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), itemMatrix-class, match(), merge(), random.transactions(), sample(), sets, size(), supportingTransactions(), tidLists-class, transactions-class, unique()

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 methods to create a data.frame representation from some arules objects. These methods are used for the coercion to a data.frame, but offer more control over the coercion process (item separators, etc.).

Usage

```
DATAFRAME(from, ...)

## S4 method for signature 'rules'
DATAFRAME(from, separate = TRUE, ...)

## S4 method for signature 'itemsets'
DATAFRAME(from, ...)

## S4 method for signature 'itemMatrix'
DATAFRAME(from, ...)
```

Arguments

from	the object to be converted into a data.frame.
...	further arguments are passed on to the labels() method defined for the object in from.
separate	logical; separate LHS and RHS in separate columns? (only for rules)

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 `labels()` method for the object specified in `from`.

Value

a data.frame.

Author(s)

Michael Hahsler

See Also

Other import/export: `LIST()`, `pmml`, `read`, `write()`

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 = NULL,
  ...
)

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`	either number of categories or a vector with boundaries for discretization (all values outside the boundaries will be set to NA). `categories` **is deprecated, use** `breaks` **instead.**
`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 lists of discretization parameters (see parameters of `discretize()`) for each numeric column (see details). If no 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 `base::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 `arulesCBA::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

`discretize()` returns 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

`base::cut()`, `arulesCBA::discretizeDF.supervised()`.

Other preprocessing: `hierarchy`, `itemCoding`, `merge()`, `sample()`

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 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 'matrix'
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")
```

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 element `"transactions"`.
- `"cosine"`: the Cosine distance.
- `"dice"`: 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" A distance calculated by $1 - r$ if $r > 1$ and 1 otherwise, where r is the Pearson's correlation coefficient.
- "phi": same as "pearson". Pearson's correlation coefficient reduces to the phi coefficient for the 2x2 contingency tables used here.
- "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.

... — further arguments.

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

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

Other proximity classes and functions: affinity(), predict(), proximity-classes

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 methods to find duplicated elements in itemMatrix, associations and their subclasses.

Usage

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

## S4 method for signature 'itemMatrix'
duplicated(x, incomparables = FALSE)

## S4 method for signature 'rules'
duplicated(x, incomparables = FALSE)

## S4 method for signature 'itemsets'
duplicated(x, incomparables = FALSE)
```

Arguments

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

Value

A logical vector indicating duplicated elements.

Author(s)

Michael Hahsler

See Also

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

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 for convenience 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

Other mining algorithms: APappearance-class, AScontrol-classes, ASparameter-classes, apriori(), fim4r(), ruleInduction(), weclat()

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 frequent itemsets
rules <- ruleInduction(itemsets, confidence = .9)
rules
```

Epub — *The Epub Transactions 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.

Format

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

Author(s)

Michael Hahsler

Source

Provided by Michael Hahsler from the custom information system ePub-WU at https://epub.wu-wien.ac.at (which has been replaced by eprint).

Examples

```
data(Epub)
inspect(head(Epub))
```

---

extract *Methods for "[": Extraction or Subsetting arules Objects*

Description

Methods for "[", i.e., extraction or subsetting for arules objects.

Usage

```
## S4 method for signature 'itemMatrix,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'transactions,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'tidLists,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'rules,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'itemsets,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

x	an object of class itemMatrix, transactions or associations.
i	select rows/sets using an integer vector containing row numbers or a logical vector.
j	select columns/items using an integer vector containing column numbers (i.e., item IDs), a logical vector or a vector of strings containing parts of item labels.
...	further arguments are ignored.
drop	ignored.

Author(s)

Michael Hahsler

See Also

Other associations functions: abbreviate(), associations-class, c(), duplicated(), inspect(), is.closed(), is.generator(), is.maximal(), is.redundant(), is.significant(), is.superset(), itemsets-class, match(), rules-class, sample(), sets, size(), sort(), unique()

Other itemMatrix and transactions functions: abbreviate(), crossTable(), c(), duplicated(), hierarchy, image(), inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), itemMatrix-class, match(), merge(), random.transactions(), sample(), sets, size(), supportingTransactions(), tidLists-class, transactions-class, unique()

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")]
```

---

fim4r *Interface to Mining Algorithms from fim4r*

Description

Interfaces the algorithms implemented in fim4r. The algorithms include: Apriori, Eclat, FPgrowth, Carpenter, IsTa, RElim and SaM.

Usage

```
fim4r(
  transactions,
  method = NULL,
  target = "frequent",
  support = 0.1,
  confidence = 0.8,
  originalSupport = TRUE,
  appear = NULL,
  report = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

- `transactions` — a transactions object
- `method` — the algorithm to be used. One of:
  - `"apriori"`, `"eclat"`, `"fpgrowth"` can mine itemsets and rules.
  - `"relim"`, `"sam"` can mine itemsets.
  - `"carpenter"`, `"ista"` can only mine closed itemset.
- `target` — the target type. One of: `"frequent"`, `"closed"`, `"maximal"`, `"generators"` or `"rules"`.
- `support` — a numeric value for the minimal support in the range $[0, 1]$.
- `confidence` — a numeric value for the minimal confidence of rules in the range $[0, 1]$.
- `originalSupport` — logical; Use the support threshold on the support of the whole rule (LHS and RHS). If `FALSE`, then LHS support (i.e., coverage) is used by the support threshold.
- `appear` — Specify item appearance in rules (only for apriori, eclat, fpgrowth and the target `"rules"`) Specify a list with two vectors (item labels and appearance modifiers) of the same length. Appearance modifiers are:
  - `"-"` (item may not appear in a rule),
  - `"a"` (item may only appear a rule antecedent/LHS),
  - `"c"` (item may only appear a rule consequent/RHS),
  - `"x"` (item may appear anywhere).
- `report` — cannot be used via the interface.
- `verbose` — logical; print used parameters?
- `...` — further arguments are passed on to the function the `fim4r::fim4r` function for the given method. Examples are: `zmin`, `zmax`, `wgts`.

Details

Installation: The package **fim4r** is not available via CRAN. If needed, the `fim4r()` function downloads and installs the current version of the package automatically. Your system needs to have build tools installed.

Build tools: You need to be able to install source packages. For Windows users this means that you need to install the RTools with a version matching your R version.

Additional Notes:

- Support and confidence are specified here in the range $[0, 1]$. This is different from the use in `fim4r` package where `supp` and `conf` have the range $[0, 100]$. `arules::fim4r()` automatically converts support and confidence internally.
- `fim4r` methods also return the empty itemset while `arules` methods do not.
- See `? fim4r::fim4r` for help on additional available arguments. This is only available after package `fim4r` is installed.
- Algorithm descriptions and references can be found on the fim4r web page in the References Section.

Value

An object of class itemsets or rules.

References

Christian Borgelt, fimi4r: Frequent Item Set Mining and Association Rule Induction for R. https://borgelt.net/fim4r.html

See Also

Other mining algorithms: APappearance-class, AScontrol-classes, ASparameter-classes, apriori(), eclat(), ruleInduction(), weclat()

Examples

```
## Not run:
data(Adult)

# list available algorithms
fim4r()

# mine association rules with FPgrowth
r <- fim4r(Adult, method = "fpgrowth",
  target = "rules", supp = .7, conf = .8)
r
inspect(head(r, by = "lift"))

# mine closed itemsets with Carpenter or IsTa
fim4r(Adult, method = "carpenter",
  target = "closed", supp = .7)
fim4r(Adult, method = "ista",
  target = "closed", supp = .7)

# mine frequent itemset of length 2 (zmin and zmax = 2)
freq_2 <- fim4r(Adult, method = "relim", target = "frequent", supp = .7,
  zmin = 2, zmax = 2)
inspect(freq_2)

# mine maximal frequent itemsets
mfis <- fim4r(Adult, method = "sam", target = "maximal", supp = .7)
inspect(mfis)

# Examples for how to use item appearance with apriori, eclat,
#   fpgrowth in fim4r. We first mine all rules.
inspect(fim4r(Adult, method = "fpgrowth",
  target = "rules", supp = .8))

# ignore item "capital-gain=None"
inspect(fim4r(Adult, method = "fpgrowth",
  target = "rules", supp = .8,
  appear = list(c("capital-gain=None"), c("-"))))
```

```
# "capital-gain=None" cannot appear in consequent (antecedent only)
inspect(fim4r(Adult, method = "fpgrowth",
  target = "rules", supp = .8,
  appear = list(c("capital-gain=None"), c("a"))))

# "capital-gain=None" cannot appear in the antecedent
inspect(fim4r(Adult, method = "fpgrowth",
  target = "rules", supp = .8,
  appear = list(c("capital-gain=None"), c("c"))))

# restrict the consequent to the item "capital-gain=None".
# That is, "" = all items can only appear in the antecedent with the
# exception that "capital-gain=None" can only appear in the consequent.
inspect(fim4r(Adult, method = "fpgrowth",
  target = "rules", supp = .8,
  appear = list(c("", "capital-gain=None"), c("a", "c"))))

## End(Not run)
```

`Groceries` *The Groceries Transactions 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.

Format

Object of class transactions.

Details

If you use this data set in your paper, please cite to the paper in the References Section.

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

Functions to use item hierarchies to aggregate items at different group levels, to perform multi-level transaction analysis.

Usage

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

filterAggregate(x)

aggregate(x, ...)

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

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

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

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.
`...`	further arguments.

Details

Often an item hierarchy is available for transactions 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."

Transactions can store item hierarchies as additional columns in the itemInfo data.frame (`"labels"` cannot be used since it 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:
`item A => group of item A`

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

Other preprocessing: `discretize()`, `itemCoding`, `merge()`, `sample()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

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 transaction weights for 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

Other weighted association mining functions: `SunBai`, `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 `image()` methods 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() in package **Matrix**

Other itemMatrix and transactions functions: abbreviate(), crossTable(), c(), duplicated(), extract, hierarchy, inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), itemMatrix-class, match(), merge(), random.transactions(), sample(), sets, size(), supportingTransactions(), tidLists-class, transactions-class, unique()

Examples

```
data("Epub")

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

Income *The Income Data Set*

Description

Survey example data from the book *The Elements of Statistical Learning*.

Format

The data is provided in two formats:

1. Income is an object of class transactions with 6876 transactions (complete cases) and 50 items. See below for details.
2. 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

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.

To create transactions for Income, 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 methods to display associations and transactions plus additional information formatted for online inspection.

Usage

```
inspect(x, ...)

## S4 method for signature 'itemsets'
inspect(x, itemSep = ", ", setStart = "{", setEnd = "}", linebreak = NULL, ...)

## S4 method for signature 'rules'
inspect(
  x,
  itemSep = ", ",
  setStart = "{",
  setEnd = "}",
  ruleSep = "=>",
  linebreak = NULL,
  ...
)

## S4 method for signature 'transactions'
inspect(x, itemSep = ", ", setStart = "{", setEnd = "}", linebreak = NULL, ...)

## S4 method for signature 'itemMatrix'
inspect(x, itemSep = ", ", setStart = "{", setEnd = "}", linebreak = NULL, ...)

## S4 method for signature 'tidLists'
inspect(x, ...)
```

Arguments

`x`	a set of associations or transactions or an itemMatrix.
`...`	additional arguments. can be used to customize the output:
`itemSep`	item separator
`setStart`	set start symbol
`setEnd`	set end symbol
`linebreak`	print only one element per line in case the output lines get very long?
`ruleSep`	rule separator

Details

`inspect()` prints the results directly. If you need to create a data.frame with a human readable version, then you can use `DATAFRAME()`.

Value

Nothing is returned (see the Details Section).

Author(s)

Michael Hahsler and Kurt Hornik

See Also

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

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 methods to calculate various additional interest measures for existing sets of itemsets or rules.

Usage

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

## S4 method for signature 'itemsets'
interestMeasure(x, measure, transactions = NULL, reuse = TRUE, ...)

## S4 method for signature 'rules'
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 the Details section for available measures). If measure is missing then all available measures are calculated.
`transactions`	the transactions 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 the 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 cell in the contingency table. Use `smoothCounts = 1` or larger values for Laplace smoothing. Use `smoothCounts = .5` for Haldane-Anscombe correction (Haldan, 1940; Anscombe, 1956) which is often used for chi-squared, phi correlation and related measures.

Details

A searchable list of definitions, equations and references for all available interest measures can be found at https://mhahsler.github.io/arules/docs/measures. The descriptions are also linked in the list below.

The following measures are implemented for **itemsets**:

- "support": Support.
- "count": Support Count.
- "allConfidence": All-Confidence.
- "crossSupportRatio": Cross-Support Ratio.
- "lift": Lift.

The following measures are implemented for **rules**:

- "support": Support.
- "confidence": Confidence.
- "lift": Lift.
- "count": Support Count.
- "addedValue": Added Value.
- "boost": Confidence Boost.
- "casualConfidence": Casual Confidence.
- "casualSupport": Casual Support.
- "centeredConfidence": Centered Confidence.
- "certainty": Certainty Factor.
- "chiSquared": Chi-Squared. Additional parameters are: `significance = TRUE` returns the p-value of the test for independence instead of the chi-squared statistic. For p-values, substitution effects (the occurrence of one item makes the occurrence of another item less likely) can be tested using the parameter `complements = FALSE`. Note: Correction for multiple comparisons can be done using `stats::p.adjust()`.
- "collectiveStrength": Collective Strength.
- "confirmedConfidence": Descriptive Confirmed Confidence.
- "conviction": Conviction.
- "cosine": Cosine.
- "counterexample": Example and Counter-Example Rate.
- "coverage": Coverage.
- "doc": Difference of Confidence.
- "fishersExactTest": Fisher's Exact Test. By default complementary effects are mined, substitutes can be found by using the parameter `complements = FALSE`. Note that Fisher's exact test is equal to hyper-confidence with `significance = TRUE`. Correction for multiple comparisons can be done using `stats::p.adjust()`.
- "gini": Gini Index.
- "hyperConfidence": 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`.
- "hyperLift": Hyper-Lift. The used quantile can be changed using parameter `level` (default: `level = 0.99`).

- "imbalance": Imbalance Ratio.
- "implicationIndex": Implication Index.
- "importance": Importance.
- "improvement": Improvement. The additional parameter `improvementMeasure` (default: `'confidence'`) can be used to specify the measure used for the improvement calculation. See Generalized improvement.
- "jaccard": Jaccard Coefficient.
- "jMeasure": J-Measure.
- "kappa": Kappa.
- "kulczynski": Kulczynski.
- "lambda": Lambda.
- "laplace": Laplace Corrected Confidence. Parameter `k` can be used to specify the number of classes (default is 2).
- "leastContradiction": Least Contradiction.
- "lerman": Lerman Similarity.
- "leverage": Leverage.
- "LIC": Lift Increase. The additional parameter `improvementMeasure` (default: `'lift'`) can be used to specify the measure used for the increase calculation. See Generalized increase ratio.
- "maxconfidence": MaxConfidence.
- "mutualInformation": Mutual Information.
- "oddsRatio": Odds Ratio.
- "phi": Phi Correlation Coefficient.
- "ralambondrainy": Ralambondrainy.
- "relativeRisk": Relative Risk.
- "rhsSupport": Right-Hand-Side Support.
- "RLD": Relative Linkage Disequilibrium.
- "rulePowerFactor": Rule Power Factor.
- "sebag": Sebag-Schoenauer.
- "stdLift": Standardized Lift.
- "table": Contingency Table. Returns the four counts for the contingency table. The entries are labeled `n11`, `n01`, `n10`, and `n00` (the first subscript is for X and the second is for Y; 1 indicated presence and 0 indicates absence). If several measures are specified, then the counts have the prefix `table.`
- "varyingLiaison": Varying Rates Liaison.
- "yuleQ": Yule's Q.
- "yuleY": Yule's Y.

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

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

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

itemsets, rules

Other interest measures: confint(), coverage(), is.redundant(), is.significant(), support()

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 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).

Usage

```
is.closed(x)

## S4 method for signature 'itemsets'
is.closed(x)
```

Arguments

`x`	a set of itemsets.

Details

Closed frequent itemsets can also be mined directly using `apriori()` or `eclat()` with target `"closed frequent 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

Other postprocessing: `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

`is.generator` *Find Generator Itemsets*

Description

Provides the generic function and the 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 typically not stored in the itemsets in **arules**.

Usage

```
is.generator(x)

## S4 method for signature 'itemsets'
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

Other postprocessing: `is.closed()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

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 `is.maximal()` and methods 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).

Usage

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

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

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

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

Arguments

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

Details

Maximally frequent itemsets can also be mined directly using `apriori()` or `eclat()` with target "maximally frequent itemsets".

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

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

Other postprocessing: `is.closed()`, `is.generator()`, `is.redundant()`, `is.significant()`, `is.superset()`

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

`is.redundant` *Find Redundant Rules*

Description

Provides the generic function `is.redundant()` and the method to find redundant rules based on any interest measure.

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.
`...`	additional arguments are passed on to `interestMeasure()`, or, for `confint = TRUE` to `confint()`.
`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.

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 available 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 defined as redundant if its OR CI overlaps with the CI of any more general rule. This type of redundancy detection removes more rules 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`.

Warning: This approach of redundancy checking is flawed since rules with non-overlapping CIs are non-redundant (same result as for a 2-sample t-test), but overlapping CIs do not automatically mean that there is no significant difference between the two measures which leads to a higher type II error. At the same time, multiple comparisons are performed leading to an increased type I error. If we are more worried about missing important rules, then the type II error is more concerning.

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

Other postprocessing: `is.closed()`, `is.generator()`, `is.maximal()`, `is.significant()`, `is.superset()`

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

Other interest measures: `confint()`, `coverage()`, `interestMeasure()`, `is.significant()`, `support()`

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 `is.significant()` and the method to find significant rules.

Usage

```
is.significant(x, ...)
```

```
## S4 method for signature 'rules'
is.significant(
  x,
  transactions = NULL,
  method = "fisher",
  alpha = 0.01,
  adjust = "none",
  reuse = TRUE,
  ...
)
```

Arguments

`x`	a set of rules.
`...`	further arguments are passed on to `interestMeasure()`.
`transactions`	optional set of transactions. Only needed if not sufficient interest measures are available in `x`. If the test should be performed on a transaction set different then the one used for mining (use `reuse = FALSE`).
`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. Some options are `"none"`, `"bonferroni"`, `"holm"`, `"fdr"`, etc. (see `stats::p.adjust()` for more methods)
`reuse`	logical indicating if information in the quality slot should be reuse for calculating the measures.

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). See Hahsler and Hornik (2007) for details.

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. doi:10.3233/IDA200711502

See Also

stats::p.adjust()

Other interest measures: confint(), coverage(), interestMeasure(), is.redundant(), support()

Other postprocessing: is.closed(), is.generator(), is.maximal(), is.redundant(), is.superset()

Other associations functions: abbreviate(), associations-class, c(), duplicated(), extract, inspect(), is.closed(), is.generator(), is.maximal(), is.redundant(), is.superset(), itemsets-class, match(), rules-class, sample(), sets, size(), sort(), unique()

Examples

```
data("Income")
rules <- apriori(Income, support = 0.2)
is.significant(rules)

rules[is.significant(rules)]

# Adjust P-values for multiple comparisons
rules[is.significant(rules, adjust = "bonferroni")]
```

---

is.superset *Find Super and Subsets*

Description

Provides the generic functions is.subset() and is.superset(), and the methods for finding super or subsets in associations and itemMatrix objects.

Usage

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

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

## S4 method for signature 'itemMatrix'
is.superset(x, y = NULL, proper = FALSE, sparse = TRUE)

## S4 method for signature 'associations'
is.superset(x, y = NULL, proper = FALSE, sparse = TRUE)

## S4 method for signature 'itemMatrix'
is.subset(x, y = NULL, proper = FALSE, sparse = TRUE)

## S4 method for signature 'associations'
is.subset(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 requires a significantly smaller amount of memory and is much faster for large sets.
`...`	currently unused.

Details

Determines 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` are very dense (contain many items).

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

Value

returns a logical matrix or a sparse `ngCMatrix` 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

Other postprocessing: `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

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 methods are used to translate between the representation in the itemMatrix format (used in transactions, rules and itemsets), item labels and numeric item IDs (i.e., the column numbers in the itemMatrix representation).

Usage

```
decode(x, ...)

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

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

encode(x, ...)

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

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

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

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)

compatible(x, y)

## S4 method for signature 'itemMatrix'
compatible(x, y)
```

```
## S4 method for signature 'associations'
compatible(x, y)
```

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()`).
`...`	further arguments.
`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.
`match`	deprecated: used `itemLabels` instead.
`y`	an object of class itemMatrix, transactions or associations to compare item coding to `x`.

Details

Item coding compatibility: When working with several datasets or different subsets of the same dataset, combining or compare the found itemsets or rules requires a compatible item coding. That is, the sparse matrices representing the items (the itemMatrix objects) have columns for the same items in exactly the same order. The coercion to transactions with `transactions()` or `as(x, "transactions")` will create the item coding by adding items in the order they are encountered in the dataset. This can lead to different item codings (different order, missing items) for even only slightly different datasets or versions of a dataset. Method `compatible()` can be used to check if two sets have the same item coding.

Defining a common item coding: When working with many sets, then first a common item coding should be defined by creating a vector with all possible item labels and then specify them as `itemLabels` to create transactions with `transactions()`. Compatible itemMatrix objects can be created using `encode()`.

Recoding and Decoding: Two incompatible objects can be made compatible using `recode()`. Recode one object by specifying the other object in `itemLabels`.

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

Value

`recode()` always returns an object of the same class as `x`.

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, itemMatrix

Other preprocessing: discretize(), hierarchy, merge(), sample()

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 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 = c("relative", "absolute"), weighted = FALSE)

## S4 method for signature 'tidLists'
itemFrequency(x, type = c("relative", "absolute"))
```

Arguments

`x`	an object of class itemMatrix or tidLists.
`...`	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()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

Examples

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

`itemFrequencyPlot` *Creating a Item Frequencies/Support Bar Plot*

Description

Provides the generic function `itemFrequencyPlot()` and the 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 `graphics::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()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

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, and associations. The class contains a sparse Matrix representation of a set of itemsets and the corresponding item labels.

Usage

```
## S4 method for signature 'itemMatrix'
summary(object, maxsum = 6, ...)

## S4 method for signature 'itemMatrix'
dim(x)

nitems(x, ...)

## S4 method for signature 'itemMatrix'
nitems(x)

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

toLongFormat(from, ...)

## S4 method for signature 'itemMatrix'
toLongFormat(from, cols = c("ID", "item"), decode = TRUE)

## S4 method for signature 'itemMatrix'
labels(object, itemSep = ",", setStart = "{", setEnd = "}")

itemLabels(object, ...)

itemLabels(object) <- value

## S4 method for signature 'itemMatrix'
itemLabels(object)

## S4 replacement method for signature 'itemMatrix'
itemLabels(object) <- value

itemInfo(object)

itemInfo(object) <- value
```

```
## S4 method for signature 'itemMatrix'
itemInfo(object)

## S4 replacement method for signature 'itemMatrix'
itemInfo(object) <- value

itemsetInfo(object)

itemsetInfo(object) <- value

## S4 method for signature 'itemMatrix'
itemsetInfo(object)

## S4 replacement method for signature 'itemMatrix'
itemsetInfo(object) <- value

## S4 method for signature 'itemMatrix'
dimnames(x)

## S4 replacement method for signature 'itemMatrix,list'
dimnames(x) <- value
```

Arguments

`object, x, from`	the object.
`maxsum`	integer, how many items should be shown for the summary?
`...`	further parameters
`cols`	columns for the long format.
`decode`	decode item IDs to item labels.
`itemSep`	item separator symbol.
`setStart`	set start symbol.
`setEnd`	set end symbol.
`value`	replacement value

Details

Representation

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

Warning: Ideally, we would store the matrix as a row-oriented sparse matrix (`ngRMatrix`), but the **Matrix** package provides better support for column-oriented sparse classes (`ngCMatrix`). The matrix is therefore internally stored in transposed form.

Working with several `itemMatrix` objects

If you work with several `itemMatrix` objects 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.

Functions

- `summary(itemMatrix)`: show a summary.
- `dim(itemMatrix)`: returns the number of rows (itemsets) and columns (items in the encoding).
- `nitems(itemMatrix)`: returns the number of items in the encoding.
- `length(itemMatrix)`: returns the number of itemsets (rows) in the matrix.
- `toLongFormat(itemMatrix)`: convert the sets to long format (a data.frame with two columns, ID and item). Column names can be specified as a character vector of length 2 called `cols`.
- `labels(itemMatrix)`: returns labels for the itemsets. The following arguments can be used to customize the representation of the labels: `itemSep`, `setStart` and `setEnd`.
- `itemLabels(itemMatrix)`: returns the item labels used for encoding as a character vector.
- `itemLabels(itemMatrix) <- value`: replaces the item labels used for encoding.
- `itemInfo(itemMatrix)`: returns the whole item/column information data.frame including labels.
- `itemInfo(itemMatrix) <- value`: replaces the item/column info by a data.frame.
- `itemsetInfo(itemMatrix)`: returns the item set/row information data.frame.
- `itemsetInfo(itemMatrix) <- value`: replaces the item set/row info by a data.frame.
- `dimnames(itemMatrix)`: returns a list with the dimname vectors.
- `dimnames(x = itemMatrix) <- value`: replace the dimnames.

Slots

`data` a sparse matrix of class ngCMatrix representing the itemsets. **Warning:** the matrix is stored in transposed form for efficiency reasons!.

`itemInfo` a data.frame

`itemsetInfo` a data.frame

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.

Coercions

- `as("matrix", "itemMatrix")`
- `as("itemMatrix", "matrix")`
- `as("list", "itemMatrix")`
- `as("itemMatrix", "list")`
- `as("itemMatrix", "ngCMatrix")`
- `as("ngCMatrix", "itemMatrix")`

Warning: the `ngCMatrix` representation is transposed!

Author(s)

Michael Hahsler

See Also

Other itemMatrix and transactions functions: abbreviate(), crossTable(), c(), duplicated(), extract, hierarchy, image(), inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), match(), merge(), random.transactions(), sample(), sets, size(), supportingTransactions(), tidLists-class, transactions-class, unique()

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! You
## can transpose it again to get the expected format.
as(imatrix[1:5], "ngCMatrix")
t(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)
```

---

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

Description

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

Usage

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

## S4 method for signature 'itemsets'
summary(object, ...)

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

```
## S4 method for signature 'itemsets'
nitems(x)

## S4 method for signature 'itemsets'
labels(object, ...)

## S4 method for signature 'itemsets'
itemLabels(object)

## S4 replacement method for signature 'itemsets'
itemLabels(object) <- value

## S4 method for signature 'itemsets'
itemInfo(object)

## S4 method for signature 'itemsets'
items(x)

## S4 replacement method for signature 'itemsets'
items(x) <- value

## S4 method for signature 'itemsets'
tidLists(x)
```

Arguments

`items`	an itemMatrix or an object that can be converted using `encode()`.
`itemLabels`	item labels used for `encode()`.
`quality`	a data.frame with quality information (one row per itemset).
`object, x`	the object
`...`	further argments
`value`	replacement value

Details

Itemsets are usually created by calling an association rule mining algorithm like `apriori()`. 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 contain several interest measures accessible with the `quality()` method. Additional measures can be calculated via `interestMeasure()`.

Functions

- `summary(itemsets)`: create a summary
- `length(itemsets)`: get the number of itemsets.
- `nitems(itemsets)`: get the number of items (columns) in the current encoding.

- `labels(itemsets)`: get the itemset labels.
- `itemLabels(itemsets)`: get the item labels.
- `itemLabels(itemsets) <- value`: replace the item labels.
- `itemInfo(itemsets)`: get item info data.frame.
- `items(itemsets)`: get items as an itemMatrix.
- `items(itemsets) <- value`: with a different itemMatrix.
- `tidLists(itemsets)`: get tidLists stored in the object (if any).

Slots

`items` an itemMatrix object representing the itemsets.

`tidLists` a tidLists or `NULL`.

`quality` a data.frame with quality information

`info` a list with mining information.

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()`.

Coercions

- `as("itemsets", "data.frame")`

Author(s)

Michael Hahsler

See Also

Superclass: associations

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

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)
```

`itemwiseSetOps` *Itemwise Set Operations*

Description

Provides the generic functions and the 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 of 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)

## S4 method for signature 'itemMatrix,itemMatrix'
itemUnion(x, y)

## S4 method for signature 'itemMatrix,itemMatrix'
itemSetdiff(x, y)

## S4 method for signature 'itemMatrix,itemMatrix'
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

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])))
```

LIST *List Representation for Objects Based on Class itemMatrix*

Description

Provides the generic function `LIST()` and the methods to create a list representation from objects of the classes itemMatrix, transactions, and tidLists.

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).

Value

a list primitive.

Author(s)

Michael Hahsler

See Also

Other import/export: `DATAFRAME()`, `pmml`, `read`, `write()`

Examples

```
data(Adult)

### default coercion (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 methods for associations, transactions and itemMatrix objects. `match()` returns a vector of the positions of (first) matches of its first argument in its second.

Usage

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

## S4 method for signature 'itemMatrix,itemMatrix'
match(x, table, nomatch = NA_integer_, incomparables = NULL)

## S4 method for signature 'rules,rules'
match(x, table, nomatch = NA_integer_, incomparables = NULL)

## S4 method for signature 'itemsets,itemsets'
match(x, table, nomatch = NA_integer_, incomparables = NULL)
```

```
## S4 method for signature 'itemMatrix,itemMatrix'
x %in% table

## S4 method for signature 'itemMatrix,character'
x %in% table

## S4 method for signature 'associations,associations'
x %in% table

## S4 method for signature 'itemMatrix,character'
x %pin% table

## S4 method for signature 'itemMatrix,character'
x %ain% table

## S4 method for signature 'itemMatrix,character'
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.

Details

`%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()`.

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

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `rules-class`, `sample()`, `sets`, `size()`, `sort()`, `unique()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

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 methods for itemMatrix and transactions to add new items to existing data.

Usage

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

```
## S4 method for signature 'itemMatrix'
merge(x, y, ...)

## S4 method for signature 'transactions'
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

Other preprocessing: discretize(), hierarchy, itemCoding, sample()

Other itemMatrix and transactions functions: abbreviate(), crossTable(), c(), duplicated(), extract, hierarchy, image(), inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), itemMatrix-class, match(), random.transactions(), sample(), sets, size(), supportingTransactions(), tidLists-class, transactions-class, unique()

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 — *The Mushroom Data Set as Transactions*

Description

The Mushroom transactions data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family.

Format

Object of class transactions with 8124 transactions and 114 items.

Details

The transaction set 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.

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.

pmml — *Read and Write PMML*

Description

This function reads and writes PMML representations (version 4.1) of associations (itemsets and rules). Write delegates to package **pmml**.

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).

Author(s)

Michael Hahsler

References

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

See Also

`pmml::pmml()`.

Other import/export: `DATAFRAME()`, `LIST()`, `read`, `write()`

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")
```

---

predict *Model Predictions*

Description

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

Usage

```
predict(object, ...)

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

Arguments

- `object` — clustered examples as an itemMatrix with cluster label specified in `labels` or medoids as an itemMatrix (use `labels = NULL`).
- `...` — further arguments passed on to `dissimilarity()`. E.g., `method`.
- `newdata` — an itemMatrix containing the objects to predict labels for.
- `labels` — an integer vector containing the labels for the examples in `object`. The cluster labels need to be contiguous integers starting with 1.
- `blocksize` — a numeric scalar indicating how much memory predict can use for big `x` and/or `y` (approx. in MB). 200 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.

Value

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

Author(s)

Michael Hahsler

See Also

Other proximity classes and functions: `affinity()`, `dissimilarity()`, `proximity-classes`

Examples

```
data("Adult")

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

## cluster a small sample and extract the cluster lael vector
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.

Details

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.

Author(s)

Michael Hahsler

See Also

`stats::dist()`, `proxy::dist()`

Other proximity classes and functions: `affinity()`, `dissimilarity()`, `predict()`

`random.transactions` *Simulate a Random Transactions*

Description

Simulate random transactions using different methods.

Usage

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

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

Arguments

`nItems`	an integer. Number of items to simulate
`nTrans`	an integer. Number of transactions to simulate
`method`	name of the simulation method used (see Details Section).
`...`	further arguments used for the specific simulation method (see details).
`verbose`	report progress?
`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 distribution).
`cvar`	variance of the corruption level.
`iWeight`	item selection weights to build patterns.

Details

Currently two simulation methods are implemented:

- `"independent"` (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).

- "agrawal" (see Agrawal and Srikant, 1994): This method creates transactions with correlated items using random.patters(). 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 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 a ntrans x nitems transactions object.

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

Other itemMatrix and transactions functions: abbreviate(), crossTable(), c(), duplicated(), extract, hierarchy, image(), inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), itemMatrix-class, match(), merge(), sample(), sets, size(), supportingTransactions(), tidLists-class, transactions-class, unique()

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 *Read Transaction Data*

Description

Reads transaction data from a file 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 a 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 for character 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

Other import/export: `DATAFRAME()`, `LIST()`, `pmml`, `write()`

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` *Association Rule Induction from Itemsets*

Description

Provides the generic function `ruleInduction()` and the method to induce all association rules which can be generated by the given set of itemsets from a transactions dataset.

Usage

```
ruleInduction(x, ...)

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

```
  method = c("ptree", "apriori"),
  reduce = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

`x`	the set of itemsets from which rules will be induced.
`...`	further arguments.
`transactions`	the transactions used to mine the itemsets. Can be omitted for method `"ptree"`, if `x` contains a (complete set) of frequent itemsets together with their support counts.
`confidence`	a numeric value between 0 and 1 giving the minimum confidence threshold for the rules.
`method`	`"ptree"` or `"apriori"`
`reduce`	remove unused items to speed up the counting process?
`verbose`	report progress?

Details

All rules that can be created using the supplied itemsets and that surpass the specified minimum confidence threshold are returned. `ruleInduction()` 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 the code example in the Example section.

Rule induction implements several induction methods. The default method is `"ptree"`

- `"ptree"` **method without transactions:** No transactions are need to be specified if `x` contains a complete set of frequent or itemsets. The itemsets' support counts are stored in a ptree and then retrieved to create rules and calculate rules confidence. This is very fast, but fails because of missing support values if `x` is not a complete set of frequent itemsets.
- `"ptree"` **method with transactions:** If transactions are specified then all transactions are counted into a prefix tree and later retrieved to create rules from the itemsets and calculate confidence values. This is slower, but necessary if `x` is not a complete set of frequent itemsets. To improve speed, unused items are removed from the transaction data before creating the prefix tree (this behavior can be changed using the argument `reduce`). This might be slower for large transaction data sets. However, this is highly recommended as the items are also reordered to reduce the counting time.
- `"apriori"` **method (always needs transactions):** All association rules are mined from the transactions data set using `apriori()` with the smallest support found in the itemsets. In a second step, all rules which cannot be generated from one of the itemsets are removed. This procedure is very slow, especially for itemsets with many elements or very low support.

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

Other mining algorithms: `APappearance-class`, `AScontrol-classes`, `ASparameter-classes`, `apriori()`, `eclat()`, `fim4r()`, `weclat()`

Examples

```
data("Adult")

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

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

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

## get rules from frequent itemsets. Here, transactions does not need to be
## specified for rule induction.
frequent_is  <- eclat(Adult, support = 0.4)
assoc_rules <- ruleInduction(frequent_is)
assoc_rules
inspect(head(assoc_rules))

## for itemsets that are not a complete set of frequent itemsets,
## transactions need to be specified.
some_is <- sample(frequent_is, 10)
some_rules <- ruleInduction(some_is, transactions = Adult)
some_rules
```

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

Description

Defines the `rules` class to represent a set of association rules and methods to work with `rules`.

Usage

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

## S4 method for signature 'rules'
summary(object, ...)

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

## S4 method for signature 'rules'
nitems(x)

## S4 method for signature 'rules'
labels(object, ruleSep = " => ", ...)

## S4 method for signature 'rules'
itemLabels(object)

## S4 replacement method for signature 'rules'
itemLabels(object) <- value

## S4 method for signature 'rules'
itemInfo(object)

lhs(x)

## S4 method for signature 'rules'
lhs(x)

lhs(x) <- value

## S4 replacement method for signature 'rules'
lhs(x) <- value

rhs(x)

rhs(x) <- value

## S4 replacement method for signature 'rules'
rhs(x) <- value

## S4 method for signature 'rules'
rhs(x)
```

```
## S4 method for signature 'rules'
items(x)

generatingItemsets(x)

## S4 method for signature 'rules'
generatingItemsets(x)
```

Arguments

`rhs, lhs`	itemMatrix objects or objects that can be converted using `encode()`.
`itemLabels`	a vector of all possible item labels (character) or a transactions object to copy the item coding used for `encode()` (see itemCoding for details).
`quality`	a data.frame with quality information (one row per rule).
`object, x`	the object
`...`	further arguments
`ruleSep`	rule separation symbol
`value`	replacement value

Details

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

To create rules manually, the itemMatrix for the LHS and the RHS of the rules need to be compatible. See itemCoding for details.

Functions

- `summary(rules)`: create a summary
- `length(rules)`: returns the number of rules.
- `nitems(rules)`: returns the number of items used in the current encoding.
- `labels(rules)`: labels for the rules.
- `itemLabels(rules)`: returns item labels for the current encoding.
- `itemLabels(rules) <- value`: change the item labels in the current encoding.
- `itemInfo(rules)`: returns the item info data.frame.
- `lhs(rules)`: returns the LHS of the rules as an itemMatrix.
- `lhs(rules) <- value`: replaces the LHS of the rules with an itemMatrix.
- `rhs(rules) <- value`: replaces the RHS of the rules with an itemMatrix.
- `rhs(rules)`: returns the RHS of the rules as an itemMatrix.
- `items(rules)`: returns all items in a rule (LHS and RHS) an itemMatrix.
- `generatingItemsets(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. This method produces the same as the result as calling `items()`, but wrapped into an itemsets object with support information.

Slots

lhs,rhs itemMatrix representing the left-hand-side and right-hand-side of the rules.

quality the quality data.frame

info a list with mining information.

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().

Coercions

- as("rules", "data.frame")

Author(s)

Michael Hahsler

See Also

Superclass: associations

Other associations functions: abbreviate(), associations-class, c(), duplicated(), extract, inspect(), is.closed(), is.generator(), is.maximal(), is.redundant(), is.significant(), is.superset(), itemsets-class, match(), sample(), sets, size(), sort(), unique()

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 methods to sample from transactions and associations.

Usage

```
## S4 method for signature 'itemMatrix'
sample(x, size, replace = FALSE, prob = NULL, ...)

## S4 method for signature 'associations'
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

Other preprocessing: `discretize()`, `hierarchy`, `itemCoding`, `merge()`

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sets`, `size()`, `sort()`, `unique()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

Examples

```
data("Adult")

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

---

sets *Set Operations*

Description

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

Usage

```
## S3 method for class 'itemMatrix'
union(x, y, ...)

## S3 method for class 'associations'
union(x, y, ...)

## S4 method for signature 'associations'
union(x, y, ...)

## S4 method for signature 'itemMatrix'
union(x, y, ...)

## S3 method for class 'itemMatrix'
intersect(x, y, ...)

## S3 method for class 'associations'
intersect(x, y, ...)
```

```
## S4 method for signature 'associations'
intersect(x, y, ...)

## S4 method for signature 'itemMatrix'
intersect(x, y, ...)

## S3 method for class 'itemMatrix'
setequal(x, y, ...)

## S3 method for class 'associations'
setequal(x, y, ...)

## S4 method for signature 'associations'
setequal(x, y, ...)

## S4 method for signature 'itemMatrix'
setequal(x, y, ...)

## S3 method for class 'itemMatrix'
setdiff(x, y, ...)

## S3 method for class 'associations'
setdiff(x, y, ...)

## S4 method for signature 'associations'
setdiff(x, y, ...)

## S4 method for signature 'itemMatrix'
setdiff(x, y, ...)

## S3 method for class 'itemMatrix'
is.element(el, set, ...)

## S3 method for class 'associations'
is.element(el, set, ...)

## S4 method for signature 'associations'
is.element(el, set, ...)

## S4 method for signature 'itemMatrix'
is.element(el, set, ...)
```

Arguments

x, y, el, set	sets of associations or itemMatrix objects.
...	Other arguments are unused.

Details

Technical note: 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

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `size()`, `sort()`, `unique()`

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `size()`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

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 in Sets*

Description

Provides the generic function `size()` and methods to get the size of each itemset in an itemMatrix or associations. For example, `size()` can be used to get a vector with the number of items in each transaction.

Usage

```
size(x, ...)

## S4 method for signature 'itemMatrix'
size(x)

## S4 method for signature 'tidLists'
size(x)

## S4 method for signature 'itemsets'
size(x)

## S4 method for signature 'rules'
size(x)
```

Arguments

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

Value

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

Author(s)

Michael Hahsler

See Also

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `supportingTransactions()`, `tidLists-class`, `transactions-class`, `unique()`

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `sort()`, `unique()`

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, ...)
```

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 (a permutation like `order()`) be returned instead of the sorted associations?
`...`	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. With `order = TRUE` an 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` or a permutation vector.

Author(s)

Michael Hahsler

See Also

Other associations functions: `abbreviate()`, `associations-class`, `c()`, `duplicated()`, `extract`, `inspect()`, `is.closed()`, `is.generator()`, `is.maximal()`, `is.redundant()`, `is.significant()`, `is.superset()`, `itemsets-class`, `match()`, `rules-class`, `sample()`, `sets`, `size()`, `unique()`

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 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, ...)
```

Arguments

`x`	object to be subsetted.
`...`	further arguments to be passed to or from other methods.
`subset`	logical expression indicating elements to keep.

Details

`subset()` finds the rows/itemsets/rules of `x` that match the expression given in `subset`. Parts of `x` like items, lhs, rhs and the columns in the quality data.frame (e.g., support and lift) can be directly referred to by their names in `subset`.

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
- %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

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 Weighted Transactions Data Set*

Description

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

Format

Object of class transactions with 6 transactions and 8 items. Weights are stored as transaction information.

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

Other weighted association mining functions: `hits()`, `weclat()`

Examples

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

transactionInfo(SunBai)
```

`support` *Support Counting for Itemsets*

Description

Provides the generic function `support()` and the methods to count support for given itemMatrix and associations in a given transactions data.

Usage

```
support(x, transactions, ...)

## S4 method for signature 'itemMatrix'
support(
  x,
  transactions,
  type = c("relative", "absolute"),
  method = c("ptree", "tidlists"),
  reduce = FALSE,
  weighted = FALSE,
  verbose = FALSE,
  ...
)

## S4 method for signature 'associations'
support(
  x,
  transactions,
  type = c("relative", "absolute"),
  method = c("ptree", "tidlists"),
  reduce = FALSE,
```

```
  weighted = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

x	the set of itemsets for which support should be counted.
transactions	the transaction data set used for mining.
...	further arguments.
type	a character string specifying if "relative" support or "absolute" support (counts) are returned for the itemsets in x. (default: "relative")
method	use "ptree" or "tidlists". See Details Section.
reduce	should unused items are removed before counting?
weighted	should support be weighted by transactions weights stored as column "weight" in transactionInfo?
verbose	report progress?

Details

Normally, the support of frequent itemsets is very efficiently counted during mining process using a set minimum support. However, if only the support for specific itemsets (maybe itemsets with very low support) is needed, or the support of a set of itemsets needs to be recalculated on different transactions than they were mined on, then `support()` can be used.

Several methods for support counting are available:

- "ptree" (default method): 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 transaction ID list intersection.
- "tidlists": 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.

To speed up counting, `reduce = TRUE` can be specified in control. Unused items are removed from the transactions before counting.

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

Other interest measures: confint(), coverage(), interestMeasure(), is.redundant(), is.significant()

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 for each itemset in an associations object which transactions support (i.e., contains all items in the itemset) it. The information is returned as a tidLists object.

Usage

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

## S4 method for signature 'associations'
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.

Value

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

Author(s)

Michael Hahsler

See Also

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `tidLists-class`, `transactions-class`, `unique()`

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

Class to represent transaction ID lists and associated methods.

Usage

```
tidLists(x)

## S4 method for signature 'tidLists'
summary(object, maxsum = 6, ...)

## S4 method for signature 'tidLists'
dim(x)
```

```
## S4 method for signature 'tidLists'
dimnames(x)

## S4 replacement method for signature 'tidLists,list'
dimnames(x) <- value

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

## S4 method for signature 'tidLists'
t(x)

## S4 method for signature 'tidLists'
transactionInfo(x)

## S4 replacement method for signature 'tidLists'
transactionInfo(x) <- value

## S4 method for signature 'tidLists'
itemInfo(object)

## S4 replacement method for signature 'tidLists'
itemInfo(object) <- value

## S4 method for signature 'tidLists'
itemLabels(object)

## S4 method for signature 'tidLists'
labels(object)
```

Arguments

`x, object`	the object
`maxsum`	maximum numbers of itemsets shown in the summary
`...`	further arguments
`value`	replacement value

Details

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 Matrix::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.

Functions

- `summary(tidLists)`: create a summary
- `dim(tidLists)`: get dimensions. The rows represent the itemsets and the columns are the transactions.
- `dimnames(tidLists)`: get dimnames
- `dimnames(x = tidLists) <- value`: replace dimnames
- `length(tidLists)`: get the number of itemsets.
- `t(tidLists)`: this object is not transposable. `t()` results in an error.
- `transactionInfo(tidLists)`: get the transaction info data.frame
- `transactionInfo(tidLists) <- value`: replace the the transaction info data.frame
- `itemInfo(tidLists)`: get the item info data.frame
- `itemInfo(tidLists) <- value`: replace the item info data.frame
- `itemLabels(tidLists)`: get the item labels
- `labels(tidLists)`: convert the tid lists into a text representation.

Slots

`data` an object of class ngCMatrix from package **Matrix**.

`itemInfo` a data.frame

`transactionInfo` a data.frame

Objects from the Class

Objects are created

- as part of the itemsets mined by `eclat()` with `tidLists = TRUE` in the ECparameter object.
- by `supportingTransactions()`.
- by coercion from an object of class transactions.
- by calls of the form `new("tidLists", ...)`.

Coercions

- `as("tidLists", "list")`
- `as("list", "tidLists")`
- `as("tidLists", "ngCMatrix")`
- `as("tidLists", "transactions")`
- `as("transactions", "tidLists")`
- `as("tidLists", "itemMatrix")`
- `as("itemMatrix", "tidLists")`

Author(s)

Michael Hahsler

See Also

Other itemMatrix and transactions functions: abbreviate(), crossTable(), c(), duplicated(), extract, hierarchy, image(), inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), itemMatrix-class, match(), merge(), random.transactions(), sample(), sets, size(), supportingTransactions(), transactions-class, unique()

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 is a subclass of itemMatrix and represents transaction data used for mining associations.

Usage

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

## S4 method for signature 'transactions'
summary(object)

## S4 method for signature 'transactions'
toLongFormat(from, cols = c("TID", "item"), decode = TRUE)

## S4 method for signature 'transactions'
items(x)

transactionInfo(x)

## S4 method for signature 'transactions'
transactionInfo(x)

transactionInfo(x) <- value

## S4 replacement method for signature 'transactions'
transactionInfo(x) <- value

## S4 method for signature 'transactions'
dimnames(x)

## S4 replacement method for signature 'transactions,list'
dimnames(x) <- value
```

Arguments

`x, object, from`	the object
`itemLabels`	a vector with labels for the items
`transactionInfo`	a transaction information data.frame with one row per transaction.
`format`	`"wide"` or `"long"` format? 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 (see `cols` below).

`cols`	a numeric or character vector of length two giving the index or names of the columns (fields) with the transaction and item ids in the long format.
`decode`	translate item IDs to item labels?
`value`	replacement value

Details

Transactions store the presence of items in each individual transaction as binary matrix where rows represent the transactions and columns represent the items. `transactions` direct extends class itemMatrix to store the sparse binary incidence matrix, item labels, and optionally transaction IDs and user IDs. 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.

Data Preparation

Data typically starts as a data.frame or a matrix and needs to be prepared before it can be converted into `transactions` (see coercion methods in the Methods Section and the Example Section below for details on the needed format).

Columns need to represent items which is different depending on the data type of the column:

- **Continuous variables:** Continuous variables cannot directly be represented as items and 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. Discretization is described in functions `discretize()` and `discretizeDF()`.
- **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:** Columns with nominal values (i.e., factor, ordered) are translated into a series of binary items (one for each level constructed as `variable name = level`). Items cannot represent order and this ordered factors lose the order information. 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.

Functions

- `summary(transactions)`: produce a summary
- `toLongFormat(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`.
- `items(transactions)`: get the transactions as an itemMatrix
- `transactionInfo(transactions)`: get the transaction info data.frame
- `transactionInfo(transactions) <- value`: replace the transaction info data.frame

- `dimnames(transactions)`: get the dimnames
- `dimnames(x = transactions) <- value`: set the dimnames

Slots

Slots are inherited from itemMatrix.

Objects from the Class

Objects are created by:

- coercion from objects of other classes. `itemLabels` and `transactionInfo` are by default created from information in `x` (e.g., from row and column names).
- the constructor function `transactions()`
- by calling `new("transactions", ...)`.

See Examples Section for creating transactions from data.

Coercions

- `as("transactions", "matrix")`
- `as("matrix", "transactions")`
- `as("list", "transactions")`
- `as("transactions", "list")`
- `as("data.frame", "transactions")`
- `as("transactions", "data.frame")`
- `as("ngCMatrix", "transactions")`

Author(s)

Michael Hahsler

See Also

Superclass: itemMatrix

Other itemMatrix and transactions functions: `abbreviate()`, `crossTable()`, `c()`, `duplicated()`, `extract`, `hierarchy`, `image()`, `inspect()`, `is.superset()`, `itemFrequencyPlot()`, `itemFrequency()`, `itemMatrix-class`, `match()`, `merge()`, `random.transactions()`, `sample()`, `sets`, `size()`, `supportingTransactions()`, `tidLists-class`, `unique()`

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 methods for itemMatrix transactions, and associations.

Usage

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

```
## S4 method for signature 'itemMatrix'
unique(x, incomparables = FALSE)

## S4 method for signature 'associations'
unique(x, incomparables = FALSE, ...)
```

Arguments

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

Details

unique() uses duplicated() to return an object with the duplicate elements removed.

Value

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

Author(s)

Michael Hahsler

See Also

Other associations functions: abbreviate(), associations-class, c(), duplicated(), extract, inspect(), is.closed(), is.generator(), is.maximal(), is.redundant(), is.significant(), is.superset(), itemsets-class, match(), rules-class, sample(), sets, size(), sort()

Other itemMatrix and transactions functions: abbreviate(), crossTable(), c(), duplicated(), extract, hierarchy, image(), inspect(), is.superset(), itemFrequencyPlot(), itemFrequency(), itemMatrix-class, match(), merge(), random.transactions(), sample(), sets, size(), supportingTransactions(), tidLists-class, transactions-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 transaction ID list 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 transactions 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

Other mining algorithms: APappearance-class, AScontrol-classes, ASparameter-classes, apriori(), eclat(), fim4r(), ruleInduction()

Other weighted association mining functions: SunBai, hits()

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 methods to write transactions or associations 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.
`...`	further arguments passed on to `write.table()`. Use `fileEncoding` to set the encoding used for writing the file.
`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?

Details

For associations (rules and itemsets) `write()` first uses coercion to data.frame to obtain a printable form of `x` and then uses `utils::write.table()` to write the data to disk. This is just a method to export the rules in human-readable form. These exported associations cannot be read back in as rules. 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**.

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

Author(s)

Michael Hahsler

See Also

Other import/export: `DATAFRAME()`, `LIST()`, `pmml`, `read`

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
```

Index