Package 'arules'

April 7, 2018

Version 1.6-1

Date 2018-04-04

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.

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

URL https://github.com/mhahsler/arules, http://lyle.smu.edu/IDA/arules

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

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

Imports stats, methods, graphics, utils

Suggests pmml, XML, arulesViz, testthat

License GPL-3

Copyright The code for apriori and eclat in src/rapriori.c 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.

RoxygenNote 6.0.1

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2018-04-07 03:48:04 UTC

R topics documented:

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

Description

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

Usage

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

Arguments

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

Author(s)

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

See Also

abbreviate in base.

Examples

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

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

---

addComplement *Add Complement-items to Transactions*

Description

Provides the generic function addComplement and the S4 methods for transactions. This function adds for given items complement items. That is it adds an artificial item to each transactions which does not contain the original item.

Usage

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

Arguments

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

Value

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

Author(s)

Michael Hahsler

See Also

transactions-class, merge

Examples

```
data("Groceries")

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

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

`Adult` *Adult Data Set*

Description

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

Usage

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

Format

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

age a numeric vector.

workclass a factor with levels `Federal-gov`, `Local-gov`, `Never-worked`, `Private`, `Self-emp-inc`, `Self-emp-not-inc`, `State-gov`, and `Without-pay`.

education an ordered factor with levels `Preschool` < `1st-4th` < `5th-6th` < `7th-8th` < `9th` < `10th` < `11th` < `12th` < `HS-grad` < `Prof-school` < `Assoc-acdm` < `Assoc-voc` < `Some-college` < `Bachelors` < `Masters` < `Doctorate`.

education-num a numeric vector.

marital-status a factor with levels `Divorced`, `Married-AF-spouse`, `Married-civ-spouse`, `Married-spouse-absent`, `Never-married`, `Separated`, and `Widowed`.

occupation a factor with levels `Adm-clerical`, `Armed-Forces`, `Craft-repair`, `Exec-managerial`, `Farming-fishing`, `Handlers-cleaners`, `Machine-op-inspct`, `Other-service`, `Priv-house-serv`, `Prof-specialty`, `Protective-serv`, `Sales`, `Tech-support`, and `Transport-moving`.

relationship a factor with levels `Husband`, `Not-in-family`, `Other-relative`, `Own-child`, `Unmarried`, and `Wife`.

race a factor with levels `Amer-Indian-Eskimo`, `Asian-Pac-Islander`, `Black`, `Other`, and `White`.

sex a factor with levels `Female` and `Male`.

capital-gain a numeric vector.

capital-loss a numeric vector.

fnlwgt a numeric vector.

hours-per-week a numeric vector.

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

income an ordered factor with levels `small` < `large`.

Details

The "Adult" database was extracted from the census bureau database found at http://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

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

References

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

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

Examples

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

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

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

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

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

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

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

---

affinity *Computing Affinity Between Items*

Description

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

Usage

```
affinity(x)
```

Arguments

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

Details

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

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

where $sup(.)$ is the support measure. This means that affinity is the *Jaccard similarity* between items.

Value

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

Author(s)

Michael Hahsler

References

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

See Also

`dissimilarity`, `ar_similarity-class`, `itemMatrix-class`

Examples

```
data("Adult")

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

a <- affinity(s)
image(a)
```

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

Description

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

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

Objects from the Class

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

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

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

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

Slots

set: an integer scalar indicating how many items are specified for each of lhs, rhs, items, both and none

items: an integer vector of item IDs (column numbers)

labels: a character vector of item labels

default: a character scalar indicating the value for default appearance

Author(s)

Michael Hahsler and Bettina Gruen

References

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

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

See Also

apriori

Examples

```
data("Adult")

## find only frequent itemsets which do not contain small or large income
is <- apriori(Adult, parameter = list(support= 0.1, target="frequent"),
```

```
  appearance = list(none = c("income=small", "income=large")))
itemFrequency(items(is))["income=small"]
itemFrequency(items(is))["income=large"]

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

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

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

`apriori` *Mining Associations with Apriori*

Description

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

Usage

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

Arguments

`data`	object of class `transactions` or any data structure which can be coerced into `transactions` (e.g., a binary matrix or data.frame).
`parameter`	object of class `APparameter` or named list. The default behavior is to mine rules with 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.)

Details

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

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

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

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

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

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

Value

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

Author(s)

Michael Hahsler and Bettina Gruen

References

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

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: `www.borgelt.net/apriori.html`

See Also

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

Examples

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

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

Description

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

Objects from the Class

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

Slots

Common slots defined in AScontrol:

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

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

verbose: a logical indicating whether to report progress

Additional slots for Apriori in APcontrol:

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

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

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

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

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

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

Additional slots for Eclat in `ECcontrol`:

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

Methods

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

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

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

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

Author(s)

Michael Hahsler and Bettina Gruen

References

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

See Also

`apriori`, `eclat`

`ASparameter-classes` *Classes ASparameter, APparameter, ECparameter — Specifying the parameter Argument of apriori() and eclat()*

Description

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

Objects from the Class

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

Slots

Common slots defined in `ASparameter`:

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

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

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

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

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

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

Additional slots for Apriori in `APparameter`:

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

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

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

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

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

Additional slots for Eclat in `ECparameter`:

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

Methods

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

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

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

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

show signature(object = "ASparameter")

Author(s)

Michael Hahsler and Bettina Gruen

References

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

See Also

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

---

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

Description

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

Objects from the Class

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

Slots

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

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

Methods

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

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

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

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

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

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

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

show `signature(object = "associations")`

Subclasses

`itemsets-class`, `rules-class`

Author(s)

Michael Hahsler

See Also

`sort`, `write`, `length`, `is.subset`, `is.superset`, `sets`, `unique`, `itemMatrix-class`

---

combine *Combining Objects*

Description

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

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

Usage

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

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

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

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

Arguments

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

Value

An object of the same class as x.

Author(s)

Michael Hahsler

See Also

itemMatrix-class, transactions-class, rules-class, itemsets-class

Examples

```
data("Adult")

## combine transactions
a1 <- Adult[1:10]
a2 <- Adult[101:110]

aComb <- c(a1, a2)
summary(aComb)

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

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

coverage *Calculate coverage for rules*

Description

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

Usage

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

Arguments

x	the set of rules.
transactions	the data set used to generate 'x'. Only needed if the quality slot of 'x' does not contain support and confidence.
reuse	reuse support and confidence stored in 'x' or recompute from transactions?

Details

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

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

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

Value

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

Author(s)

Michael Hahsler

See Also

`interestMeasure`, `rules-class`

Examples

```
data("Income")

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

inspect(rules)
```

`crossTable` *Cross-tabulate joint occurrences across pairs of items*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Michael Hahsler

See Also

`transactions-class`, `itemMatrix-class`.

Examples

```
data("Groceries")

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

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

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

chi2 <- crossTable(Groceries, measure="chiSquared", sort=TRUE)
chi2[1:5,1:5]
```

DATAFRAME *Data.frame Representation for arules Objects*

Description

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

Usage

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

Arguments

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

Details

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

Value

a data.frame.

Author(s)

Michael Hahsler

See Also

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

Examples

```
data(Adult)

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

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


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

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

---

discretize *Convert a Continuous Variable into a Categorical Variable*

Description

This function implements several basic unsupervised methods to convert continuous variables into a categorical variables (factor) suitable for association rule mining. For convenience, a whole data.frame can be discretized (i.e., all numeric columns are discretized).

Usage

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

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

Arguments

x	a numeric vector (continuous variable).
method	discretization method. Available are: "interval" (equal interval width), "frequency" (equal frequency), "cluster" (k-means clustering) and "fixed" (categories specifies interval boundaries). Note that equal frequency does not achieve perfect equally sized groups if the data contains duplicated values.

`breaks, categories`
: `categories` **is deprecated, use** `breaks`**.** either number of categories or a vector with boundaries for discretization (all values outside the boundaries will be set to NA).

`labels`
: character vector; labels for the levels of the resulting category. By default, labels are constructed using "(a,b]" interval notation. If `labels = FALSE`, simple integer codes are returned instead of a factor..

`include.lowest`
: logical; should the first interval be closed to the left?

`right`
: logical; should the intervals be closed on the right (and open on the left) or vice versa?

`dig.lab`
: integer; number of digits used to create labels.

`ordered_result`
: logical; return a ordered factor?

`infinity`
: logical; should the first/last break boundary changed to +/-Inf?

`onlycuts`
: logical; return only computed interval boundaries?

`...`
: for method "cluster" further arguments are passed on to `kmeans`. .

`df`
: data.frame; each numeric column in the data.frame is discretized.

`methods`
: named list of lists or a data.frame; the named list contains list of discretization parameters (see parameters of `discretize`) for each numeric column (see details). If no specific discretization is specified for a column, then the default settings for `discretize` are used. Note: the names have to match exactly. If a data.frame is specified, then the discretization breaks in this data.frame are applied to `df`.

`default`
: named list; parameters for `discretize` used for all columns not specified in `methods`.

Details

`discretize` only implements unsupervised discretization. See packages **arulesCBA**, **discretization** or **RWeka** 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(...))`.

Value

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

`discretizeDF` returns a discretized data.frame.

Author(s)

Michael Hahsler

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)

### specify discretization for the petal columns
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"))
  ))
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 Computation*

Description

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

Usage

```
dissimilarity(x, y = NULL, method = NULL, args = NULL, ...)

## S4 method for signature 'itemMatrix'
dissimilarity(x, y = NULL, method = NULL, args = NULL,
which = "transactions")

## S4 method for signature 'associations'
dissimilarity(x, y = NULL, method = NULL, args = NULL,
which = "associations")

## S4 method for signature 'matrix'
dissimilarity(x, y = NULL, method = NULL, args = NULL)
```

Arguments

`x`	the set of elements (e.g., `matrix`, `itemMatrix`, `transactions`, `itemsets`, `rules`).
`y`	`NULL` or a second set to calculate cross dissimilarities.
`method`	the distance measure to be used. Implemented measures are (defaults to `"jaccard"`):

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

`"cosine"`: the *cosine* distance.

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

`"euclidean"`: the *euclidean* distance.

"jaccard": the number of items which occur in both elements divided by the total number of items in the elements (Sneath, 1957). This measure is often also called: *binary, asymmetric binary,* etc.

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

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

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

For associations the following additional measures are available:

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

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

args — a list of additional arguments for the methods.

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

... — further arguments.

Value

returns an object of class dist.

Author(s)

Michael Hahsler

References

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

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

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

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

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

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

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

See Also

affinity, dist-class, itemMatrix-class, associations-class.

Examples

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

s <- Groceries[,itemFrequency(Groceries)>0.05]
d_jaccard <- dissimilarity(s, which = "items")
plot(hclust(d_jaccard, method = "ward.D2"), main = "Dendrogram for items")

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

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

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

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

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

## use affinity to cluster rules
## Note: we need to supply the transactions (or affinities) from the
## dataset (sample).
d_affinity <- dissimilarity(rules, method = "affinity",
  args = list(transactions = s))
hc <- hclust(d_affinity, method = "ward.D2")
plot(hc, main = "Dendrogram for Rules (Affinity)")

## create 4 groups and inspect the rules in the first group.
```

```
assign <- cutree(hc, k = 3)
inspect(rules[assign == 1])
```

`duplicated` *Find Duplicated Elements*

Description

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

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

Usage

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

Arguments

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

Value

A logical vector indicating duplicated elements.

Author(s)

Michael Hahsler

See Also

`unique`, `rules-class`, `itemsets-class`, `itemMatrix-class`

Examples

```
data("Adult")

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

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

`eclat` *Mining Associations with Eclat*

Description

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

Usage

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

Arguments

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

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*. Technical Report 651, Computer Science Department, University of Rochester, Rochester, NY 14627.

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

ECLAT Implementation: http://www.borgelt.net/eclat.html

See Also

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

Examples

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

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

Epub *Epub Data Set*

Description

The Epub data set contains the download history of documents from the electronic publication platform of the Vienna University of Economics and Business Administration. The data was recorded between Jan 2003 and Dec 2008.

Usage

```
data(Epub)
```

Format

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

Author(s)

Michael Hahsler

Source

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

`Groceries` *Groceries Data Set*

Description

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

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

Usage

```
data(Groceries)
```

Format

Object of class `transactions`.

Author(s)

Michael Hahsler

Source

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

References

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

`hierarchy` *Support for Item Hierarchies*

Description

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

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

Usage

```
## S4 method for signature 'itemMatrix'
aggregate(x, by)
## S4 method for signature 'itemsets'
aggregate(x, by)
## S4 method for signature 'rules'
aggregate(x, by)
addAggregate(x, by, postfix = "*")
filterAggregate(x)
```

Arguments

`x`	an transactions, itemsets or rules object.
`by`	name of a field (hierarchy level) available in itemInfo or a vector of character strings (factor) of the same length as items in `x` by which should be aggregated. Items receiving the same label in `by` will be aggregated into a single, higher-level item.
`postfix`	characters added to mark group-level items.

Details

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

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

Multi-level analysis: To analyze relationships between individual items and item groups, `addAggregate()` creates a new transactions object which contains both, the original items and group-level items (marked with a given postfix). In association rule mining, all items are handled the same, which means that we will produce a large number of rules of the type

$$itemA => groupofitemA$$

with a confidence of 1. This happens also to 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

Examples

```
data("Groceries")
Groceries

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

## aggregate by level2: items will become labels at level2
## 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 orginal and aggregated transactions
inspect(head(Groceries, 2))
inspect(head(Groceries_level2, 2))

## create lables manually (organize items by the first letter)
mylevels <- toupper(substr(itemLabels(Groceries), 1, 1))
head(mylevels)

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

## 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])

## mine multi-level rules:
## (1) add aggregate items. These items are followed by 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"))
## this contains many spurous rules of type 'item X => aggregare of item X'
## with a confidence of 1 and high lift.

## filter spurious rules resulting from the aggregation
rules <- filterAggregate(rules)
inspect(head(rules, by = "lift"))
```

`hits` *Computing Transaction Weights With HITS*

Description

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

Usage

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

Arguments

`data`	an object of or coercible to class `transactions`.
`iter`	an integer value specifying the maximum number of iterations to use.
`tol`	convergence tolerance (default `FLT_EPSILON`).
`type`	a string value specifying the norming of the hub weights. For `"normed"` scale the weights to unit length (L2 norm), and for `"relative"` to unit sum.
`verbose`	a logical specifying if progress and runtime information should be displayed.

Details

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

Value

A `numeric` vector with transaction weights for `data`.

Author(s)

Christian Buchta

References

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

See Also

Class `transactions`, function `weclat`

Examples

```
data(SunBai)

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

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

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

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

image *Visual Inspection of Binary Incidence Matrices*

Description

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

Usage

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

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

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

Arguments

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

Author(s)

Michael Hahsler

See Also

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

Examples

```
data("Epub")

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

Income *Income Data Set*

Description

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

Usage

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

Format

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

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

sex a factor with levels male female

marital status a factor with levels married cohabitation divorced widowed single

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

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

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

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

dual incomes a factor with levels not married yes no

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

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

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

type of home a factor with levels house condominium apartment mobile Home other

ethnic classification a factor with levels american indian asian black east indian hispanic pacific islander white other

language in home a factor with levels english spanish other

Details

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

Author(s)

Michael Hahsler

Source

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

Obtained from the web site of the book: Hastie, T., Tibshirani, R. \& Friedman, J. (2001) *The Elements of Statistical Learning*. Springer-Verlag.

Examples

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

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

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

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

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

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

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

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

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

inspect *Display Associations and Transactions in Readable Form*

Description

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

Usage

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

Arguments

`x`	a set of associations or transactions or an itemMatrix.
`...`	additional arguments can be used to customize the output: `setStart`, `setEnd`, `itemSep` and `ruleSep`. Items are printed only one per line in case the output lines get very long. This can also be directly controlled using `linebreak`.

Author(s)

Michael Hahsler and Kurt Hornik

See Also

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

Examples

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

inspect(rules[1000], ruleSep = "---->", itemSep = " + ", setStart = "", setEnd ="",
  linebreak = FALSE)
```

interestMeasure *Calculate Additional Interest Measures*

Description

Provides the generic function `interestMeasure` and the needed S4 method to calculate various additional interest measures for existing sets of itemsets or rules. Definitions and equations can be found in Hahsler (2015).

Usage

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

Arguments

x	a set of itemsets or rules.
measure	name or vector of names of the desired interest measures (see details for available measures). If measure is missing then all available measures are calculated.
transactions	the transaction data set used to mine the associations or a set of different transactions to calculate interest measures from (Note: you need to set reuse=FALSE in the later case).
reuse	logical indicating if information in quality slot should be reuse for calculating the measures. This speeds up the process significantly since only very little (or no) transaction counting is necessary if support, confidence and lift are already available. Use reuse=FALSE to force counting (might be very slow but is necessary if you use a different set of transactions than was used for mining).
...	further arguments for the measure calculation.

Details

For itemsets X the following measures are implemented:

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

Range: $[0, 1]$

"crossSupportRatio", cross-support ratio (Xiong et al., 2003) Defined on itemsets as the ratio of the support of the least frequent item to the support of the most frequent item, i.e., $\frac{min(supp(x \in X))}{max(supp(x \in X))}$. Cross-support patterns have a ratio smaller than a set threshold. Normally many found patterns are cross-support patterns which contain frequent as well as rare items. Such patterns often tend to be spurious.

Range: $[0, 1]$

"lift" Probability (support) of the itemset over the product of the probabilities of all items in the itemset, i.e., $\frac{supp(X)}{\prod_{x \in X} supp(X)}$. This is a measure of dependence similar to lift for rules.

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

"support", supp (Agrawal et al., 1996) Support is an estimate of $P(X)$ a measure of generality of the itemset.

Range: $[0, 1]$

"count" Absolute support count of the itemset.

Range: $[0, \infty]$

For rules $X \Rightarrow Y$ the following measures are implemented. In the following we use the notation $supp(X \Rightarrow Y) = supp(X \cup Y)$ to indicate the support of the union of the itemsets X and Y, i.e., the proportion of the transactions that contain both itemsets. We also use $\overline{X}$ as the complement itemset to X with $supp(\overline{X}) = 1 - supp(X)$, i.e., the proportion of transactions that do not contain X.

"addedValue", added Value, AV, Pavillon index, centered confidence (Tan et al., 2002) Defined as $conf(X \Rightarrow Y) - supp(Y)$

Range: $[-.5, 1]$

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

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

Called with `significance=TRUE`, the p-value of the test for independence is returned instead of the chi-squared statistic. For p-values, substitutes effects can be tested using the parameter `complements = FALSE`.

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

"certainty", certainty factor, CF, Loevinger (Berzal et al., 2002) The certainty factor is a measure of variation of the probability that Y is in a transaction when only considering transactions with X. An inreasing CF means a decrease of the probability that Y is not in a transaction that X is in. Negative CFs have a similar interpretation.

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

"collectiveStrength" Collective strength (S).

Range: $[0, \infty]$

"confidence", conf (Agrawal et al., 1996) Rule confidence is an estimate of $P(Y|X)$ calculated as $\frac{supp(X \Rightarrow Y)}{supp(X)}$. Confidence is a measure of validity.

Range $[0, 1]$

"conviction" (Brin et al. 1997) Defined as $\frac{supp(X)supp(\overline{Y})}{supp(X \cup \overline{Y})}$.

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

"cosine" (Tan et al., 2004) Defined as $\frac{supp(X \cup Y)}{\sqrt{(supp(X)supp(Y))}}$

Range: $[0, 1]$

"count" Absolute support count of the rule.

Range: $[0, \infty]$

"coverage", cover, LHS-support Support of the left-hand-side of the rule, i.e., $supp(X)$. A measure of to how often the rule can be applied.

Range: $[0, 1]$

"confirmedConfidence", descriptive confirmed confidence (Kodratoff, 1999) Confidence confirmed by its negative as $conf(X \Rightarrow Y) - conf(X \Rightarrow \overline{Y})$.

Range: $[-1, 1]$

"casualConfidence", casual confidence (Kodratoff, 1999) Confidence reinforced by negatives given by $\frac{1}{2}(conf(X \Rightarrow Y) + conf(\overline{Y} \Rightarrow \overline{X}))$.

Range: $[0, 1]$

"casualSupport", casual support (Kodratoff, 1999) Support improved by negatives given by $supp(X \cup Y) - supp(\overline{X} \cup \overline{Y})$.

Range: $[-1, 1]$

"counterexample", example and counter-example rate $\frac{supp(X \cup Y) - supp(X \cup \overline{Y})}{supp(X \cup Y)}$

Range: $[0, 1]$

"descriptiveConfirm", descriptive-confirm (Kodratoff, 1999) Defined by $supp(X \cup Y) - supp(X \cup \overline{Y})$.

Range: $[0, 1]$

"doc", difference of confidence (Hofmann and Wilhelm, 2001) Defined as $conf(X \Rightarrow Y) - conf(\overline{X} \Rightarrow Y)$.

Range: $[-1, 1]$

"fishersExactTest", Fisher's exact test (Hahsler and Hornik, 2007) p-value of Fisher's exact test used in the analysis of contingency tables where sample sizes are small. By default complementary effects are mined, substitutes can be found by using the parameter `complements = FALSE`.

Note that it is equal to hyper-confidence with `significance=TRUE`.

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

"gini", Gini index (Tan et al., 2004) Measures quadratic entropy.

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

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

Hyper-lift is defined as:

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

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

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

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

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

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

By default complementary effects are mined, substitutes can be found by using the parameter `complements = FALSE`.

Range: $[0, 1]$

"imbalance", imbalance ratio, IR (Wu, Chen and Han, 2010) IR is defined as $\frac{|supp(X) - supp(Y)|}{supp(X) + supp(Y) - supp(X \Rightarrow Y))}$ gauges the degree of imbalance between two events that the lhs and the rhs are contained in a transaction. The ratio is close to 0 if the conditional probabilities are similar (i.e., very balanced) and close to 1 if they are very different.

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

"implicationIndex", implication index (Gras, 1996) Defined as $\sqrt{N}\frac{supp(X\cup\overline{Y})-supp(X)supp(\overline{Y})}{\sqrt{supp(X)supp(\overline{Y})}}$.
Represents a variation of the Lerman similarity.
Range: $[0, 1]$ (0 means independence)

"importance" (MS Analysis Services) Log likelihood of the right-hand side of the rule, given the left-hand side of the rule.
$log_{10}(L(X \Rightarrow Y)/L(X \Rightarrow \bar{Y}))$
where L is the Laplace corrected confidence.
Range: $[-Inf, Inf]$

"improvement" (Bayardo et al., 2000) The improvement of a rule is the minimum difference between its confidence and the confidence of any more general rule (i.e., a rule with the same consequent but one or more items removed in the LHS). Defined as $min_{X' \subset X}(conf(X \Rightarrow Y) - conf(X' \Rightarrow Y)$
Range: $[0, 1]$

"jaccard", Jaccard coefficient (Tan and Kumar, 2000) sometimes also called Coherence (Wu et al., 2010) Null-invariant measure defined as $\frac{supp(X\cup Y)}{supp(X)+supp(Y)-supp(X\cup Y)}$
Range: $[-1, 1]$ (0 for independence)

"jMeasure", J-measure, J (Smyth and Goodman, 1991) Measures cross entrophy.
Range: $[0, 1]$ (0 for independence)

"kappa" (Tan and Kumar, 2000) Defined as $\frac{supp(X\cup Y)+supp(\overline{X}\cup\overline{Y})-supp(X)supp(Y)-supp(\overline{X})supp(\overline{Y})}{1-supp(X)supp(Y)-supp(\overline{X})supp(\overline{Y})}$
Range: $[-1, 1]$ (0 means independence)

"klosgen", Klosgen (Tan and Kumar, 2000) Defined as $\sqrt{supp(X \cup Y)}conf(X \Rightarrow Y)-supp(Y)$
Range: $[-1, 1]$ (0 for independence)

"kulczynski" (Wu, Chen and Han, 2010; Kulczynski, 1927) Calculate the null-invariant Kulczynski measure with a preference for skewed patterns.
Range: $[0, 1]$

"lambda", Goodman-Kruskal λ, predictive association (Tan and Kumar, 2000) Range: $[0, 1]$

"laplace", Laplace corrected confidence, L (Tan and Kumar 2000) Estimates confidence with increasing each count by 1. Prevents counts of 0 and L decreases with lower support.
Range: $[0, 1]$

"leastContradiction", least contradiction (Aze and Kodratoff, 2004 $\frac{supp(X\cup Y)-supp(X\cup\overline{Y})}{supp(Y)}$.
Range: $[-1, 1]$

"lerman", Lerman similarity (Lerman, 1981) Defined as $\sqrt{N}\frac{supp(X\cup Y)-supp(X)supp(Y)}{\sqrt{supp(X)supp(Y)}}$
Range: $[0, 1]$

"leverage", PS (Piatetsky-Shapiro 1991) PS is defined as $supp(X \Rightarrow Y) - supp(X)supp(Y)$. It measures the difference of X and Y appearing together in the data set and what would be expected if X and Y where statistically dependent. It can be interpreted as the gap to independence.
Range: $[-1, 1]$ (0 indicates intependence)

"lift", interest factor (Brin et al. 1997) Lift quantifies dependence between X and Y by $\frac{supp(X\cup Y)}{supp(X)supp(Y)}$.
Range: $[0, \infty]$ (1 means independence)

"maxConfidence" (Wu et al. 2010) Null-invariant measure defined as $max(conf(X \Rightarrow Y), conf(X \Rightarrow Y))$.

Range: $[0, 1]$

"mutualInformation", uncertainty, M (Tan et al., 2002) Measures the information gain for Y provided by X.

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

"oddsRatio", odds ratio α (Tan et al., 2004) The odds of finding X in transactions which contain Y divided by the odds of finding X in transactions which do not contain Y.

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

"phi", correlation coefficient ϕ (Tan et al., 2004 Equivalent to Pearsons Product Moment Correlation Coefficient ρ.

Range: $[-1, 1]$ (0 when X and Y are independent)

"ralambrodrainy", Ralambrodrainy Measure (Diatta et al., 2007) Range: $[0, 1]$

"RLD", relative linkage disequilibrium (Kenett and Salini, 2008) RLD evaluates the deviation of the support of the whole rule from the support expected under independence given the supports of the LHS and the RHS. The code was contributed by Silvia Salini.

Range: $[0, 1]$

"rulePowerFactor", rule power factor (Ochin et al., 2016) Product of support and confidence. Can be seen as rule confidence weighted by support.

Range: $[0, 1]$

"sebag", Sebag measure (Sebag and Schoenauer, 1988) Defined as $\frac{supp(X \cup Y)}{supp(X \cup \overline{Y})}$

Range: $[0, 1]$

"support", supp (Agrawal et al., 1996) Support is an estimate of $P(X \cup Y)$ and measures the generality of the rule.

Range: $[0, 1]$

"varyingLiaison", varying rates liaison (Bernard and Charron, 1996) Defined as $\frac{supp(X \cup Y)}{supp(X)supp(Y)} - 1$. Is equivalent to $lift(X \Rightarrow Y) - 1$

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

"yuleQ", Yule's Q (Tan and Kumar, 2000) Defined as $\frac{\alpha-1}{\alpha+1}$ where α is the odds ratio.

Range: $[-1, 1]$

"yuleY", Yule's Y (Tan and Kumar, 2000) Defined as $\frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}$ where α is the odds ratio.

Range: $[-1, 1]$

Value

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

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

`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: http://michael.hahsler.net/research/association_rules/measures.html.

Agrawal, R., H Mannila, R Srikant, H Toivonen, AI Verkamo (1996). Fast Discovery of Association Rules. *Advances in Knowledge Discovery and Data Mining* 12 (1), 307–328.

Aze, J. and Y. Kodratoff (2004). Extraction de pepites de connaissances dans les donnees: Une nouvelle approche et une etude de sensibilite au bruit. In *Mesures de Qualite pour la fouille de donnees. Revue des Nouvelles Technologies de l'Information, RNTI.*

Bernard, Jean-Marc and Charron, Camilo (1996). L'analyse implicative bayesienne, une methode pour l'etude des dependances orientees. II : modele logique sur un tableau de contingence Mathematiques et Sciences Humaines, Volume 135 (1996), p. 5–18.

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.

Berzal, Fernando, Ignacio Blanco, Daniel Sanchez and Maria-Amparo Vila (2002). Measuring the accuracy and interest of association rules: A new framework. *Intelligent Data Analysis* 6, 221–235.

Brin, Sergey, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur (1997). Dynamic itemset counting and implication rules for market basket data. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 255–264, Tucson, Arizona, USA.

Diatta, J., H. Ralambondrainy, and A. Totohasina (2007). Towards a unifying probabilistic implicative normalized quality measure for association rules. In *Quality Measures in Data Mining,* 237–250, 2007.

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

Hofmann, Heike and Adalbert Wilhelm (2001). Visual comparison of association rules. *Computational Statistics*, 16(3):399–415.

Kenett, Ron and Silvia Salini (2008). Relative Linkage Disequilibrium: A New measure for association rules. In *8th Industrial Conference on Data Mining ICDM 2008,* July 16–18, 2008, Leipzig/Germany.

Kodratoff, Y. (1999). Comparing Machine Learning and Knowledge Discovery in Databases: An Application to Knowledge Discovery in Texts. *Lecture Notes on AI (LNAI) - Tutorial series*.

Kulczynski, S. (1927). Die Pflanzenassoziationen der Pieninen. *Bulletin International de l'Academie Polonaise des Sciences et des Lettres, Classe des Sciences Mathematiques et Naturelles B,* 57–203.

Lerman, I.C. (1981). Classification et analyse ordinale des donnees. Paris.

Liu, Bing, Wynne Hsu, and Yiming Ma (1999). Pruning and summarizing the discovered associations. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 125–134. ACM Press, 1999.

Ochin, Suresh Kumar, and Nisheeth Joshi (2016). Rule Power Factor: A New Interest Measure in Associative Classification. *6th International Conference On Advances In Computing and Communications, ICACC 2016,* 6-8 September 2016, Cochin, India.

Omiecinski, Edward R. (2003). Alternative interest measures for mining associations in databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):57–69, Jan/Feb 2003.

Piatetsky-Shapiro, G. (1991). Discovery, analysis, and presentation of strong rules. In: *Knowledge Discovery in Databases*, pages 229–248.

Sebag, M. and M. Schoenauer (1988). Generation of rules with certainty and confidence factors from incomplete and incoherent learning bases. In *Proceedings of the European Knowledge Acquisition Workshop (EKAW'88)*, Gesellschaft fuer Mathematik und Datenverarbeitung mbH, 28.1–28.20.

Smyth, Padhraic and Rodney M. Goodman (1991). Rule Induction Using Information Theory. *Knowledge Discovery in Databases*, 159–176.

Tan, Pang-Ning and Vipin Kumar (2000). Interestingness Measures for Association Patterns: A Perspective. TR 00-036, Department of Computer Science and Engineering University of Minnesota.

Tan, Pang-Ning, Vipin Kumar, and Jaideep Srivastava (2002). Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '02)*, ACM, 32–41.

Tan, Pang-Ning, Vipin Kumar, and Jaideep Srivastava (2004). Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313.

Wu, T., Y. Chen, and J. Han (2010). Re-examination of interestingness measures in pattern mining: A unified framework. *Data Mining and Knowledge Discovery*, 21(3):371-397, 2010.

Xiong, Hui, Pang-Ning Tan, and Vipin Kumar (2003). Mining strong affinity association patterns in data sets with skewed support distribution. In Bart Goethals and Mohammed J. Zaki, editors, *Proceedings of the IEEE International Conference on Data Mining*, November 19–22, 2003, Melbourne, Florida, pages 387–394.

See Also

itemsets-class, rules-class

Examples

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

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

`is.closed` *Find Closed Itemsets*

Description

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

Usage

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

Arguments

`x`	a set of itemsets.

Value

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

Author(s)

Michael Hahsler

References

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

See Also

`itemsets-class`

`is.maximal` *Find Maximal Itemsets*

Description

Provides the generic function and the S4 method `is.maximal` for finding maximal itemsets.

Usage

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

Arguments

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

Details

An itemset is maximal in a set if no proper superset of the itemset is contained in the set (Zaki et al., 1997).

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

`is.superset`, `itemMatrix-class`, `itemsets-class`

`is.redundant` *Find Redundant Rules*

Description

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

Usage

```
is.redundant(x, ...)
## S4 method for signature 'rules'
is.redundant(x, measure = "confidence")
```

Arguments

`x`	a set of rules.
`measure`	measure used to check for redundancy.
`...`	additional arguments.

Details

A rule is 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). In this implementation other measures than confidence, e.g. improvement of lift, can be used as well.

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.

See Also

`interestMeasure`

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", default = "lhs"))

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

## redundant rules
inspect(rules[is.redundant(rules)])

## non-redundant rules
inspect(rules[!is.redundant(rules)])
```

`is.significant` *Find Significant Rules*

Description

Provides the generic functions and the S4 method `is.significant` to find rules where the LHS and the RHS depend on each other. This uses Fisher's exact test and corrects for multiple comparisons.

Usage

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

Arguments

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

Value

returns a logical vector indicating which rules are significant.

Author(s)

Michael Hahsler

See Also

interestMeasure, p.adjust

Examples

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

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

---

is.superset *Find Super and Subsets*

Description

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

Usage

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

Arguments

x, y	associations or itemMatrix objects. If y = NULL, the super or subset structure within set x is calculated.
proper	a logical indicating if all or just proper super or subsets.
sparse	a logical indicating if a sparse (ngCMatrix) rather than a dense logical matrix sgould be returned. Sparse computation preserves a significant amount of memory and is much faster for large sets.
...	currently unused.

Details

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

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

Value

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

Author(s)

Michael Hahsler and Ian Johnson

See Also

`associations-class`, `itemMatrix-class`

Examples

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

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

---

itemCoding *Item Coding — Handling Item Labels and Column IDs*

Description

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

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

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

`recode` recodes an `itemMatrix` object so its coding is compatible with another object or the matrix follows a certain order of items.

Usage

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

encode(x, ...)
## S4 method for signature 'list'
```

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

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

Arguments

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

Value

recode always returns an object of class itemMatrix.

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

Author(s)

Michael Hahsler

See Also

LIST, associations-class, itemMatrix-class

Examples

```
data("Adult")

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

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

```
decode(list, itemLabels = iLabels)

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

iM <- encode(data, iLabels)
iM

inspect(iM)

## use the itemMatrix to create transactions
as(iM, "transactions")


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

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

`itemFrequency` *Getting Frequency/Support for Single Items*

Description

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

Usage

```
itemFrequency(x, ...)

## S4 method for signature 'itemMatrix'
itemFrequency(x, type, weighted = FALSE)
```

Arguments

`x`	an object.
`...`	further arguments are passed on.

type	a character string specifying if "relative" frequency/support or "absolute" frequency/support (item counts) is returned. (default: "relative").
weighted	should support be weighted by transactions weights stored as column "weight" in transactionInfo?

Value

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

Author(s)

Michael Hahsler

See Also

itemFrequencyPlot, itemMatrix-class, transactions-class

Examples

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

---

itemFrequencyPlot *Creating a Item Frequencies/Support Bar Plot*

Description

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

Usage

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

Arguments

x	the object to be plotted.
...	further arguments are passed on (see barplot from possible arguments).
type	a character string indicating whether item frequencies should be displayed relative of absolute.
weighted	should support be weighted by transactions weights stored as column "weight" in transactionInfo?
support	a numeric value. Only display items which have a support of at least support. If no population is given, support is calculated from x otherwise from the population. Support is interpreted relative or absolute according to the setting of type.
topN	a integer value. Only plot the topN items with the highest item frequency or lift (if lift = TRUE). The items are plotted ordered by descending support.
population	object of same class as x; if x is a segment of a population, the population mean frequency for each item can be shown as a line in the plot.
popCol	plotting color for population.
popLwd	line width for population.
lift	a logical indicating whether to plot the lift ratio between instead of frequencies. The lift ratio is gives how many times an item is more frequent in x than in population.
horiz	a logical. If horiz = FALSE (default), the bars are drawn vertically. If TRUE, the bars are drawn horizontally.
names	a logical indicating if the names (bar labels) should be displayed?
cex.names	a numeric value for the expansion factor for axis names (bar labels).
xlab	a character string with the label for the x axis (use an empty string to force no label).
ylab	a character string with the label for the y axis (see xlab).
mai	a numerical vector giving the plots margin sizes in inches (see '? par').

Value

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

Author(s)

Michael Hahsler

See Also

itemFrequency, itemMatrix-class

Examples

```
data(Adult)

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

## simple plot
itemFrequencyPlot(Adult.largeIncome)

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

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

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

Description

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

Details

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

Objects from the Class

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

Slots

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

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

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

Methods

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

labels `signature(x = "transactions")`; returns labels for the itemsets. The following arguments can be used to customize the representation of the labels: `itemSep`, `setStart` and `setEnd`.

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

show `signature(object = "itemMatrix")`

summary `signature(object = "itemMatrix")`

Author(s)

Michael Hahsler

See Also

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

Examples

```
set.seed(1234)

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

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

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

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

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

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

## create itemsets from itemMatrix
is <- new("itemsets", items = i[1:3])
inspect(is)

## create rules (rhs and lhs cannot share items so I use
## itemSetdiff here). Also assign (random) support.
rules <- new("rules", lhs=itemSetdiff(i[4:6],i[1:3]), rhs=i[1:3],
  quality = data.frame(support = runif(3)))
inspect(rules)
```

`itemSetOperations` *Itemwise Set Operations*

Description

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

Usage

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

Arguments

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

Value

An object of class `itemMatrix` is returned.

Author(s)

Michael Hahsler

See Also

`itemMatrix-class`

Examples

```
data("Adult")

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

---

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

Description

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

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

Objects from the Class

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

Slots

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

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

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

Extends

Class `associations`, directly.

Methods

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

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

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

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

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

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

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

summary `signature(object = "itemsets")`

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

Author(s)

Michael Hahsler

See Also

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

Examples

```
data("Adult")

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

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

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

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

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

`length` *Getting the Number of Elements*

Description

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

Usage

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

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

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

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

Arguments

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

Details

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

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

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

Value

An integer scalar giving the "length" of x.

Author(s)

Michael Hahsler

LIST *List Representation for Objects Based on Class itemMatrix*

Description

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

Usage

```
LIST(from, ...)

## S4 method for signature 'itemMatrix'
LIST(from, decode = TRUE)

## S4 method for signature 'transactions'
LIST(from, decode = TRUE)

## S4 method for signature 'tidLists'
LIST(from, decode = TRUE)
```

Arguments

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

Details

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

Value

a list primitive.

Author(s)

Michael Hahsler

See Also

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

Examples

```
data(Adult)

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

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

match *Value Matching*

Description

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

%in% is a more intuitive interface as a binary operator, which returns a logical vector indicating if there is a match or not for the items in the itemsets (left operand) with the items in the table (right operand).

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

Usage

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

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

Arguments

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

Value

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

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

Author(s)

Michael Hahsler

See Also

subset, rules-class, itemsets-class, itemMatrix-class

Examples

```
data("Adult")

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

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

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

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

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

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

`merge` *Adding Items to Data*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Michael Hahsler

See Also

transactions-class, itemMatrix-class, addComplement

Examples

```
data("Groceries")

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

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

Mushroom *Mushroom Data Set*

Description

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

Usage

```
data(Mushroom)
```

Format

Object of class transactions.

Author(s)

Michael Hahsler

Source

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

References

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

`predict` *Model Predictions*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Michael Hahsler

See Also

`dissimilarity`, `itemMatrix-class`

Examples

```
data("Adult")

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

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

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


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

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

Description

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

Objects from the Class

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

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

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

Slots

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

Author(s)

Michael Hahsler

See Also

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

`random.transactions` *Simulate a Random Transaction Data Set*

Description

Simulates a random `transactions` object using different methods.

Usage

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

Arguments

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

Details

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

Currently two simulation methods are implemented:

method `"independent"` **(see Hahsler et al., 2006)** All items are treated as independent. The transaction size is determined by $rpois(lambda - 1) + 1$, where `lambda` can be specified (defaults to 3). Note that one subtracted from lambda and added to the size to avoid empty transactions. The items in the transactions are randomly chosen using the numeric probability vector `iProb` of length `nItems` (default: 0.01 for each item).

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

lTrans average length of transactions.

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

lPats average length of patterns.

corr correlation between consecutive patterns.

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

cvar variance of the corruption level.

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

The patterns are created using the following function:

```
random.patterns(nItems, nPats = 2000,  method = "agrawal", lPats = 4, corr = 0.5, cmean = 0.5, cva
```

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

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

Value

Returns an object of class `transactions`.

Author(s)

Michael Hahsler

References

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

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

See Also

`transactions-class`.

Examples

```
## generate random 1000 transactions for 200 items with
## a success probability decreasing from 0.2 to 0.0001
## using the method described in Hahsler et al. (2006).
trans <- random.transactions(nItems = 200, nTrans = 1000,
   lambda = 5, iProb = seq(0.2,0.0001, length=200))

## size distribution
summary(size(trans))

## display random data set
image(trans)

## use the method by Agrawal and Srikant (1994) to simulate transactions
## which contains correlated items. This should create data similar to
## T10I4D100K (we just create 100 transactions here to speed things up).
```

```
patterns <- random.patterns(nItems = 1000)
summary(patterns)

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

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

`read.PMML` *Read and Write PMML*

Description

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

Usage

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

Arguments

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

Details

Write delegates to package pmml.

Author(s)

Michael Hahsler

References

PMML home page: http://www.dmg.org

See Also

pmml.

Examples

```
data("Groceries")

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

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

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

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

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

---

read.transactions *Read Transaction Data*

Description

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

Usage

```
read.transactions(file, format = c("basket", "single"), sep = "",
                  cols = NULL, rm.duplicates = FALSE,
                  quote = "\"'", skip = 0,
                  encoding = "unknown")
```

Arguments

`file`	the file name.
`format`	a character string indicating the format of the data set. One of `"basket"` or `"single"`, can be abbreviated.
`sep`	a character string specifying how fields are separated in the data file. The default (`""`) splits at whitespaces.
`cols`	For the 'single' format, `cols` is a numeric or character vector of length two giving the numbers or names of the columns (fields) with the transaction and item ids, respectively. If character, the first line of `file` is assumed to be a header with column names. For the 'basket' format, `cols` can be a numeric scalar giving the number of the column (field) with the transaction ids. If `cols = NULL`, the data do not contain transaction ids.

rm.duplicates	a logical value specifying if duplicate items should be removed from the transactions.
quote	a list of characters used as quotes when reading.
skip	number of lines to skip in the file before start reading data.
encoding	character string indicating the encoding which is passed to readLines or scan (see Encoding).

Details

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

Value

Returns an object of class transactions.

Author(s)

Michael Hahsler and Kurt Hornik

See Also

transactions-class

Examples

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

## read demo data (skip comment line)
tr <- read.transactions("demo_basket", format = "basket", sep=",", skip = 1)
inspect(tr)


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

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


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

`ruleInduction` *Rule Induction from Itemsets*

Description

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

Usage

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

Arguments

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

Details

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

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

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

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

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

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

Value

An object of class `rules`.

Author(s)

Christian Buchta and Michael Hahsler

References

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

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

See Also

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

Examples

```
data("Adult")

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

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

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

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

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

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

Description

The `rules` class represents a set of rules.

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

Objects from the Class

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

Slots

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

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

`quality`: a data.frame; typically contains measures like support, confidence and count (i.e., the absolute support count)

Extends

Class `associations`, directly.

Methods

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

generatingItemsets `signature(x = "rules")`; returns a collection of the itemsets which generated the rules, one itemset for each rule. Note that the collection can be a multiset and contain duplicated elements. Use `unique` to remove duplicates and obtain a proper set. Technically this method produces the same as the result of method `items()`, but wrapped into an `itemsets` object with support information.

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

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

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

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

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

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

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

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

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

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

summary `signature(object = "rules")`

Author(s)

Michael Hahsler

See Also

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

Examples

```
data("Adult")

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

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

## Display 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 = " ---> ")
```

`sample` *Random Samples and Permutations*

Description

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

Usage

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

Arguments

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

Value

An object of the same class as `x`.

Author(s)

Michael Hahsler

See Also

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

Examples

```
data("Adult")

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

setOperations *Set Operations*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

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

Author(s)

Michael Hahsler

See Also

`associations-class`, `itemMatrix-class`

Examples

```
data("Adult")

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

## take 2 subsets
r1 <- r[1:10]
```

```
r2 <- r[6:15]

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

`size` *Number of Items*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Michael Hahsler

See Also

`itemMatrix-class`, `transactions-class`

Examples

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

sort *Sort Associations*

Description

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

Usage

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

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

Arguments

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

Details

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

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

Value

An object of the same class as x.

Author(s)

Michael Hahsler

See Also

associations-class

Examples

```
data("Adult")

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

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

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

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

---

subset *Subsetting Itemsets, Rules and Transactions*

Description

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

Usage

```
subset(x, ...)

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

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

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

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

Arguments

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

Details

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

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

Value

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

Author(s)

Michael Hahsler

See Also

`%in%`, `%pin%`, `%ain%`, `%oin%`, `itemMatrix-class`, `itemsets-class`, `rules-class`, `transactions-class`

Examples

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

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

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

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

SunBai *The SunBai Data Set*

Description

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

Usage

```
data(SunBai)
```

Details

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

Source

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

See Also

Class transactions, method transactionInfo, function hits.

Examples

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

transactionInfo(SunBai)
```

support *Support Counting for Itemsets*

Description

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

Usage

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

Arguments

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

Details

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

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

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

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

Value

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

Author(s)

Michael Hahsler and Christian Buchta

References

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

See Also

itemMatrix-class, associations-class, transactions-class

Examples

```
data("Income")

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

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

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

---

supportingTransactions

Supporting Transactions

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Michael Hahsler

See Also

tidLists-class

Examples

```
data <- list(
c("a","b","c"),
c("a","b"),
c("a","b","d"),
c("b","e"),
c("b","c","e"),
c("a","d","e"),
c("a","c"),
c("a","b","d"),
c("c","e"),
c("a","b","d","e")
)
data <- as(data, "transactions")

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

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

as(st, "list")
```

---

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

Description

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

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

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

Objects from the Class

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

Slots

`data`**:** object of class `ngCMatrix`.

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

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

Methods

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

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

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

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

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

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

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

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

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

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

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

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

inspect inspect the transaction ID lists.

itemInfo returns the slot `itemInfo`.

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

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

show `signature(object = "tidLists")`

summary `signature(object = "tidLists")`

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

Author(s)

Michael Hahsler

See Also

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

Examples

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

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

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

## inspect visually
image(tl)

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

---

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

Description

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

Details

Transactions can be created by coercion from lists containing transactions, but also from matrix and data.frames. However, you will need to prepare your data first (see coercion methods in the Methods Section and the Example Section below for details on the needed format). Association rule mining can only use items and does not work with continuous variables.

For example, an item describing a person (i.e., the considered object called a transaction) could be *tall*. The fact that the person is tall would be encoded in the transaction containing the item *tall*. This is typically encoded in a transaction-by-items matrix by a `TRUE` value. This is why `as.transaction` can deal with logical columns, because it assumes the column stands for an item. The function also can convert columns with nominal values (i.e., factors) into a series of binary items (one for each level). So if you have nominal variables then you need to make sure they are factors (and not characters or numbers) using something like

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

Continuous variables need to be discretized first. An item resulting from discretization might be *age>18* and the column contains only `TRUE` or `FALSE`. Alternatively it can be a factor with levels *age<=18*, *50=>age>18* and *age>50*. These will be automatically converted into 3 items, one for each level. Have a look at the function `discretize` for automatic discretization.

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

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

Objects from the Class

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

Slots

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

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

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

Extends

Class `itemMatrix`, directly.

Methods

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

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

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

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

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

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

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

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

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

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

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

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

show `signature(object = "transactions")`

summary `signature(object = "transactions")`

Author(s)

Michael Hahsler

See Also

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

Examples

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

```
      c("a","b","d"),
      c("c","e"),
      c("a","b","d","e")
      )

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

## coerce into transactions
trans1 <- as(a_list, "transactions")

## analyze transactions
summary(trans1)
image(trans1)

## example 2: creating transactions from a matrix
a_matrix <- matrix(c(
  1,1,1,0,0,
1,1,0,0,0,
1,1,0,1,0,
0,0,1,0,1,
1,1,0,1,1
  ), ncol = 5)

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

a_matrix

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

## example 3: creating transactions from data.frame
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 to logicals and NAs are ignored
a_df

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

## example 4: creating transactions from a data.frame with
## transaction IDs and items (by converting it into a list of transactions first)
a_df3 <- data.frame(
  TID = c(1,1,2,2,2,3),
```

```
  item=c("a","b","a","b","c", "b")
  )
a_df3
trans4 <- as(split(a_df3[,"item"], a_df3[,"TID"]), "transactions")
trans4
inspect(trans4)
## Note: This is very slow for large datasets. It is much faster to
## read transactions in this format from disk using read.transactions()
## with format = "single".

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

irisDisc <- discretizeDF(iris)
head(irisDisc)
trans5 <- as(irisDisc, "transactions")
trans5
inspect(head(trans5))
```

`unique` *Remove Duplicated Elements from a Collection*

Description

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

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

Usage

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

Arguments

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

Value

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

Author(s)

Michael Hahsler

See Also

duplicated, associations-class, itemMatrix-class

Examples

```
data("Adult")

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

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

---

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

Description

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

Usage

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

Arguments

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

Details

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

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

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

Value

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

Note

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

Author(s)

Christian Buchta

References

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

See Also

Class transactions, function ruleInduction, eclat

Examples

```
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)
```

---

write *Write Transactions or Associations to a File*

Description

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

Usage

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

Arguments

x	the transactions or associations (rules, itemsets, etc.) object.
file	either a character string naming a file or a connection open for writing. '""' indicates output to the console.
format	format to write transactions.
sep	the field separator string. Values within each row of x are separated by this string. Use quote=TRUE and sep="," for saving data as in csv format.
quote	a logical value. Quote fields?
...	further arguments passed on to write.table or write. Use fileEncoding to set the encoding used for writing the file.

Details

For associations (rules and itemsets) write first uses coercion to data.frame to obtain a printable form of x and then uses write.table to write the data to disk.

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

Note: To save and load associations in compact form, use save and load from the **base** package. Alternatively, association can be written to disk in PMML (Predictive Model Markup Language) via write.PMML. This requiresvpackages **pmml**.

Author(s)

Michael Hahsler

See Also

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

Examples

```
data("Epub")

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

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

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

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

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

`[-methods` *Methods for "[": Extraction or Subsetting in Package 'arules'*

Description

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

Methods

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

[`signature(x = "itemsets", i = "ANY", j = "ANY", drop= "ANY")`; extracts a subset of itemsets and the associated quality measures. `j` has to be missing.

[`signature(x = "rules", i = "ANY", j = "ANY", drop= "ANY")`; extracts a subset of rules and the associated quality measures. `j` has to be missing.

[`signature(x = "transactions", i = "ANY", j = "ANY", drop= "ANY")`; extracts a subset of transactions/items from a transactions object (a binary incidence matrix). `i` and `j` can be numeric where `i` selects transactions and `j` selects items.

[`signature(x = "tidLists", i = "ANY", j = "ANY", drop= "ANY")`; extracts parts (transaction ID vectors) from `tidLists`. `i` selects the items or itemsets and `j` selects transactions in the lists.

Author(s)

Michael Hahsler

See Also

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

Examples

```
data(Adult)
Adult

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

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

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

Index