Package 'arulesViz'

May 20, 2019

Version 1.3-3

Date 2019-05-20

Title Visualizing Association Rules and Frequent Itemsets

Depends arules (>= 1.4.1), grid

Imports scatterplot3d, vcd, seriation, igraph (>= 1.0.0), graphics,
methods, utils, grDevices, stats, colorspace, DT, plotly,
visNetwork

Suggests graph, Rgraphviz, iplots, shiny, shinythemes, htmlwidgets

Description
Extends package 'arules' with various visualization techniques for association rules and itemsets. The package also includes several interactive visualizations for rule exploration.

License GPL-3

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

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

Copyright (C) 2018 Michael Hahsler, Tyler Giallanza and Sudheer
Chelluboina

NeedsCompilation no

Author Michael Hahsler [aut, cre, cph],
Giallanza Tyler [ctb],
Sudheer Chelluboina [ctb]

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

Repository CRAN

Date/Publication 2019-05-20 17:20:08 UTC

R topics documented:

`inspectDT` *Inspect Associations Interactively Using datatable*

Description

Uses **datatable** to create a HTML table widget using the DataTables library. Rules can be interactively filtered and sorted.

Usage

```
inspectDT(x, ...)
```

Arguments

`x`	an object of class "rules" or "itemsets".
`...`	additional arguments. `precision` controls the precision used to print the quality measures (defaults to 2). All other arguments are passed on to `datatable`.

Value

A datatable htmlwidget.

Author(s)

Michael Hahsler

See Also

`datatable` in **DT**.

Examples

```
## Not run:
data(Groceries)
rules <- apriori(Groceries, parameter=list(support=0.005, confidence=0.5))
rules

inspectDT(rules)

### save table as a html page.
p <- inspectDT(rules)
htmlwidgets::saveWidget(p, "arules.html", selfcontained = FALSE)
browseURL("arules.html")
## End(Not run)
```

plot	*Visualize Association Rules and Itemsets*

Description

Methods (S3) to visualize association rules and itemsets. Implemented are several popular visualization methods including scatter plots with shading (two-key plots), graph based visualizations, doubledecker plots, etc.

Many plots can use different rendering engines including static standard plots (mostly using **grid**), standard plots with interactive manipulation and interactive HTML widget-based visualizations.

Usage

```
## S3 method for class 'rules'
plot(x, method = NULL, measure = "support", shading = "lift",
    interactive = NULL, engine = "default", data = NULL, control = NULL, ...)
## S3 method for class 'itemsets'
plot(x, method = NULL, measure = "support", shading = NA,
    interactive = NULL, engine = "default", data = NULL, control = NULL, ...)
```

Arguments

`x`	an object of class "rules" or "itemsets".
`method`	a string with value "scatterplot", "two-key plot", "matrix", "matrix3D", "mosaic", "doubledecker", "graph", "paracoord" or "grouped", "iplots" selecting the visualization method (see Details). Note that some methods may only be available for rules or itemsets.
`measure`	measure(s) of interestingness (e.g., "support", "confidence", "lift", "order") used in the visualization. Some visualization methods need one measure, others take a vector with two measures (e.g., scatterplot). In some plots (e.g., graphs) `NA` can be used to suppress using a measure.
`shading`	measure of interestingness used for the color of the points/arrows/nodes (e.g., "support", "confidence", "lift"). The default is "lift". `NA` can be often used to suppress shading.
`interactive`	deprecated. See parameter `engine` below.
`control`	a list of control parameters for the plot. The available control parameters depend on the used visualization method (see Details).
`data`	the dataset (class "transactions") used to generate the rules/itemsets. Only "mosaic" and "doubledecker" require the original data.
`engine`	a string indicating the plotting engine used to render the plot. The "default" engine uses (mostly) **grid**, but some plots can produce interactive interactive grid visualizations using engine "interactive", or HTML widgets using engine "htmlwidget". These widgets can be saved as stand-alone web pages (see Examples). Note that HTML widgets tend to get very slow or unresponsive for too many rules. To prevent this situation, the control parameter `max` sets a limit, and the user is warned if the limit is reached.

`...` Further arguments are added for convenience to the `control` list.

Details

Most visualization techniques are described by Bruzzese and Davino (2008), however, we added more color shading, reordering and interactive features (see Hahsler, 2017). Many visualization methods take extra parameters as the `control` parameter list. Although, we have tried to keep control parameters consistent, the available control parameters vary (slightly) from visualization method to visualization method. A complete list of parameters with default values can be obtained using verbose mode. For example,

`plot(rules, method = "graph", control = list(verbose = TRUE))`

prints a complete list of control parameters for method "graph" (default engine).

The following visualization method are available:

"scatterplot", "two-key plot" This visualization method draws a two dimensional scatterplot with different measures of interestingness (parameter "measure") on the axes and a third measure (parameter "shading") is represented by the color of the points. There is a special value for shading called "order" which produces a two-key plot where the color of the points represents the length (order) of the rule.

Interactive manipulations are available. Engine "htmlwidget" is available to produce an interactive web-based visualization (uses **plotly**).

"matrix" Arranges the association rules as a matrix with the itemsets in the antecedents on one axis and the itemsets in the consequents on the other. The measure of interestingness (first element of `measure`) is either visualized by a color (darker means a higher value for the measure) or as the height of a bar (engine "3d"). The control parameter `reorder` takes the values `"none"`, `"measure"`, `"support/confidence"`, or `"similarity"` and can be used to reorder LHS and RHS of the rules differntly. The default reordering average measure (typically lift) pushing the rules with the highest lift value to the top-left corner of the plot. Interactive visualizations using engine "interactive" or "htmlwidget" (via **plotly**) are available.

"grouped matrix" Grouped matrix-based visualization (Hahsler and Karpienko, 2016; Hahsler 2016). Antecedents (columns) in the matrix are grouped using clustering. Groups are represented by the most interesting item (highest ratio of support in the group to support in all rules) in the group. Balloons in the matrix are used to represent with what consequent the antecedents are connected.

Interactive manipulations (zooming into groups and identifying rules) are available.

The list of control parameters for this method includes:

"main" plot title

"k" number of antecedent groups (default: 20)

"rhs_max" maximal number of RHSs to show. The rest are suppressed. (default: 10)

"lhs_items" number of LHS items shown (default: 2)

"aggr.fun" aggregation function can be any function computing a scalar from a vector (e.g., min, mean (default), median, sum, max). It is also used to reorder the balloons in the plot.

"col" color palette (default is 100 heat colors.)

"gp_labels", "gp_main", "gp_labs", "gp_lines" `gpar()` objects used to specify color, font and font size for different elements.

"graph" Represents the rules (or itemsets) as a graph with items as labeled vertices, and rules (or itemsets) represented as vertices connected to items using arrows. For rules, the LHS items are connected with arrows pointing to the vertex representing the rule and the RHS has an arrow pointing to the item.

Several engines are available. The default engine uses **igraph** (`plot.igraph` and `tkplot` for the interactive visualization). `...` arguments are passed on to the respective plotting function (use for color, etc.).

Alternatively, the engines "graphviz" (**Rgraphviz**) and "htmlwidget" (**visNetwork**) are available. Note that Rgraphviz has to be installed separately from `http://www.bioconductor.org/`.

"doubledecker", "mosaic" Represents a single rule as a doubledecker or mosaic plot. Parameter `data` has to be specified to compute the needed contingency table. No interactive version is available.

"paracoord" Represents the rules (or itemsets) as a parallel coordinate plot. Currently there is no interactive version available.

"iplots" Experimental interactive plots (package **iplots**) which support selection, highlighting, brushing, etc. Currently plots a scatterplot (support vs. confidence) and several histograms. Interactive manipulations are available.

Value

Several interactive plots return a set of selected rules/itemsets. Other plots might return other data structures. For example, graph-based plots return the graph (invisibly). Engine "htmlwidget" always returns an object of class htmlwidget.

Author(s)

Michael Hahsler and Sudheer Chelluboina. Some visualizations are based on the implementation by Martin Vodenicharov.

References

Bruzzese, D. and Davino, C. (2008), Visual Mining of Association Rules, in Visual Data Mining: Theory, Techniques and Tools for Visual Analytics, Springer-Verlag, pp. 103-122.

Hahsler, M. and Karpienko, R. (2016), Visualizing Association Rules in Hierarchical Groups. *Journal of Business Economics,* 87(3):17-335, May 2016

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).

Hahsler, M. (2017), arulesViz: Visualizing association rules with R. *R Journal,* 9(2):163-175, December 2017.

See Also

`plotly_arules`, `scatterplot3d` in **scatterplot3d**, `plot.igraph` and `tkplot` in **igraph**, `seriate` in **seriation**

Examples

```
data(Groceries)
rules <- apriori(Groceries, parameter=list(support=0.001, confidence=0.8))
rules

## Scatterplot
## -----------
plot(rules)

## Scatterplot with custom colors
library(colorspace) # for sequential_hcl
plot(rules, control = list(col=sequential_hcl(100)))
plot(rules, col=sequential_hcl(100))
plot(rules, col=grey.colors(50, alpha =.8))

## See all control options using verbose
plot(rules, verbose = TRUE)

## Interactive plot (selected rules are returned)
## Not run:
sel <- plot(rules, engine = "interactive")
## End(Not run)

## Create a html widget for interactive visualization
## Not run:
plot(rules, engine = "htmlwidget")
## End(Not run)

## Two-key plot (is a scatterplot with shading = "order")
plot(rules, method = "two-key plot")


## Matrix shading
## --------------

## The following techniques work better with fewer rules
subrules <- subset(rules, lift>5)
subrules

## 2D matrix with shading
plot(subrules, method="matrix")

## 3D matrix
plot(subrules, method="matrix", engine = "3d")

## Matrix with two measures
plot(subrules, method="matrix", shading=c("lift", "confidence"))

## Reorder rules
plot(subrules, method="matrix", control = list(reorder = "none"))
plot(subrules, method="matrix", control = list(reorder = "support/confidence"))
plot(subrules, method="matrix", control = list(reorder = "similarity"))
```

```
## Interactive matrix plot (default interactive and as a html widget)
## Not run:
plot(subrules, method="matrix", engine="interactive")
plot(subrules, method="matrix", engine="htmlwidget")
## End(Not run)

## Grouped matrix plot
## -------------------

plot(rules, method="grouped matrix")
plot(rules, method="grouped matrix",
  col = grey.colors(10),
  gp_labels = gpar(col = "blue", cex=1, fontface="italic"))

## Interactive grouped matrix plot
## Not run:
sel <- plot(rules, method="grouped", engine = "interactive")
## End(Not run)

## Graphs
## ------

## Graphs only work well with very few rules
subrules2 <- sample(subrules, 25)

plot(subrules2, method="graph")

## Custom colors
plot(subrules2, method="graph",
  nodeCol = grey.colors(10), edgeCol = grey(.7), alpha = 1)

## igraph layout generators can be used (see ? igraph::layout_)
plot(subrules2, method="graph", layout=igraph::in_circle())
plot(subrules2, method="graph",
  layout=igraph::with_graphopt(spring.const=5, mass=50))

## Graph rendering using Graphviz
## Not run:
plot(subrules2, method="graph", engine="graphviz")
## End(Not run)

## Default interactive plot (using igraph's tkplot)
## Not run:
plot(subrules2, method="graph", engine = "interactive")
## End(Not run)

## Interactive graph as a html widget (using igraph layout)
## Not run:
plot(subrules2, method="graph", engine="htmlwidget")
plot(subrules2, method="graph", engine="htmlwidget",
  igraphLayout = "layout_in_circle")
```

```
## End(Not run)

## Parallel coordinates plot
## -------------------------

plot(subrules2, method="paracoord")
plot(subrules2, method="paracoord", control = list(reorder=TRUE))

## Doubledecker plot
## -----------------

## Note: only works for a single rule
oneRule <- sample(rules, 1)
inspect(oneRule)
plot(oneRule, method="doubledecker", data = Groceries)

## Itemsets
## --------

itemsets <- eclat(Groceries, parameter = list(support = 0.02, minlen=2))
plot(itemsets)
plot(itemsets, method="graph")
plot(itemsets, method="paracoord", alpha=.5, reorder=TRUE)

## Add more quality measures to use for the scatterplot
## ----------------------------------------------------

quality(itemsets) <- interestMeasure(itemsets, trans=Groceries)
head(quality(itemsets))
plot(itemsets, measure=c("support", "allConfidence"), shading="lift")

## Save HTML widget as web page
## ----------------------------
## Not run:
p <- plot(rules, engine = "html")
htmlwidgets::saveWidget(p, "arules.html", selfcontained = FALSE)
browseURL("arules.html")
## End(Not run)
# Note: selfcontained seems to make the browser slow.
```

`plotly_arules` *Interactive Scatter Plot for Association Rules using plotly*

Description

Plot an interactive scatter plot for association rules using **plotly**. This function is **deprecated**. Use `plot` with parameter `engine = "plotly"` instead.

Usage

```
plotly_arules(x, method = "scatterplot", measure = c("support", "confidence"),
  shading = "lift", max = 1000, ...)
```

Arguments

`x`	an object of class "rules".
`method`	currently the methods "scatterplot", "two-key plot" and "matrix" are supported.
`measure`	measure(s) of interestingness (e.g., "support", "confidence", "lift", "order") used in the visualization as x and y-axis.
`shading`	measure of interestingness used for color shading.
`max`	client side processing in plotly is expensive. We restrict the number of rules to the max best rules (according to the measure used for shading.)
`...`	The following additional arguments can be used: `colors` to specify a color palette, `precision` to specify the precision used for printing quality measures, `jitter` to reduce overplotting in scatterplots (defaults to .1 if overplotting would occur), and `marker` with a list of markter attributes (e.g., size, symbol and opacity).

Value

The plotly object (plotly_hash) which can be saved as a htmlwidget.

Examples

```
## Not run:
library(plotly)
data(Groceries)
rules <- apriori(Groceries, parameter=list(support=0.001, confidence=0.8))
rules

# interactive scatter plot visualization
plotly_arules(rules)
plotly_arules(rules, measure = c("support", "lift"), shading = "confidence")
plotly_arules(rules, method = "two-key plot")

# add jitter, change color and markers and add a title
plotly_arules(rules, jitter = 10,
  marker = list(opacity = .7, size = 10, symbol = 1),
  colors = c("blue", "green"))


# save a plot as a html page
p <- plotly_arules(rules)
htmlwidgets::saveWidget(p, "arules.html", selfcontained = FALSE)
browseURL("arules.html")
# Note: selfcontained seems to make the browser slow.

# interactive matrix visualization
```

```
plotly_arules(rules, method = "matrix")

## End(Not run)
```

ruleExplorer *Explore Association Rules Interactively*

Description

Explore association rules using interactive manipulations and visualization using **shiny**.

Usage

```
ruleExplorer(x, parameter = NULL)
```

Arguments

`x`	a set of rules, a transactions object or a data.frame.
`parameter`	a list with parameters passed on to `apriori`. the list can be used to set the initial support and confidence thresholds. Values are ignored if `x` contains a set of rules.

Author(s)

Tyler Giallanza and Michael Hahsler. Adapted from functions originally created by Andrew Brooks. See https://github.com/brooksandrew/Rsenal for the original code.

See Also

`plot` with `engine = "html"`, `inspectDT`, `apriori`.

Examples

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

# explore pre-mined rules
rules <- apriori(Groceries, parameter=list(support=0.001, confidence=0.8))
rules

ruleExplorer(rules)

# mine and explore rules on the fly
ruleExplorer(Groceries)

## End(Not run)
```

saveAsGraph	*Save rules or itemsets as a graph description*

Description

This function saves a rule sat as a graph description in different formats (e.g., GraphML, dimacs, dot).

Usage

```
saveAsGraph(x, file, type="items", format="graphml")
```

Arguments

x	an object of class "rules" or "itemsets".
file	file name
type	see type in plot with method "graph" (e.g., "itemsets", "items").
format	file format (e.g., "edgelist", "graphml", "dimacs", "gml", "dot"). See `write.graph` in package **igraph**.

Author(s)

Michael Hahsler

See Also

`plot`, `write.graph` in **igraph**

Examples

```
data("Groceries")
rules <- apriori(Groceries, parameter=list(support=0.01, confidence=0.5))

saveAsGraph(rules, "rules.graphml")

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

Index