Package ‘editrules’

June 11, 2015

Maintainer Edwin de Jonge <edwindjonge@gmail.com>
License GPL-3
Title Parsing, Applying, and Manipulating Data Cleaning Rules
LazyData no
Type Package
LazyLoad yes
Author Edwin de Jonge, Mark van der Loo
Description Facilitates reading and manipulating (multivariate) data restrictions
(edit rules) on numerical and categorical data. Rules can be defined with common R syntax
and parsed to an internal (matrix-like format). Rules can be manipulated with
variable elimination and value substitution methods, allowing for feasibility checks
and more. Data can be tested against the rules and erroneous fields can be found based
on Fellegi and Holt's generalized principle. Rules dependencies can be visualized with
using the igraph package.
Version 2.9.0
Depends R (>= 2.12.0), igraph
Imports lpSolveAPI
URL https://github.com/data-cleaning/editrules
BugReports https://github.com/data-cleaning/editrules/issues
Date 2015-06-11
NeedsCompilation no
Repository CRAN
Date/Publication 2015-06-11 11:16:28

R topics documented:
editrules-package ........................................... 3
adjacency ....................................................... 4
as.editmatrix ................................................ 7
as.editset .................................................... 8
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>as.lp.mip</td>
<td>9</td>
</tr>
<tr>
<td>as.matrix.editarray</td>
<td>9</td>
</tr>
<tr>
<td>as.matrix.editmatrix</td>
<td>12</td>
</tr>
<tr>
<td>as.mip</td>
<td>15</td>
</tr>
<tr>
<td>backtracker</td>
<td>16</td>
</tr>
<tr>
<td>blocks</td>
<td>17</td>
</tr>
<tr>
<td>c.editset</td>
<td>19</td>
</tr>
<tr>
<td>checkDatamodel</td>
<td>21</td>
</tr>
<tr>
<td>condition</td>
<td>22</td>
</tr>
<tr>
<td>datamodel</td>
<td>22</td>
</tr>
<tr>
<td>disjunct</td>
<td>23</td>
</tr>
<tr>
<td>echelon</td>
<td>24</td>
</tr>
<tr>
<td>editfile</td>
<td>25</td>
</tr>
<tr>
<td>editnames</td>
<td>25</td>
</tr>
<tr>
<td>editrules.plotting</td>
<td>26</td>
</tr>
<tr>
<td>edits</td>
<td>29</td>
</tr>
<tr>
<td>editType</td>
<td>30</td>
</tr>
<tr>
<td>eliminate</td>
<td>30</td>
</tr>
<tr>
<td>errorLocalizer</td>
<td>33</td>
</tr>
<tr>
<td>errorLocalizer_mip</td>
<td>37</td>
</tr>
<tr>
<td>errorLocation</td>
<td>38</td>
</tr>
<tr>
<td>generateEdits</td>
<td>42</td>
</tr>
<tr>
<td>getA</td>
<td>43</td>
</tr>
<tr>
<td>getAb</td>
<td>44</td>
</tr>
<tr>
<td>getb</td>
<td>45</td>
</tr>
<tr>
<td>getH</td>
<td>46</td>
</tr>
<tr>
<td>getOps</td>
<td>47</td>
</tr>
<tr>
<td>getVars</td>
<td>48</td>
</tr>
<tr>
<td>impliedValues</td>
<td>49</td>
</tr>
<tr>
<td>is.editrules</td>
<td>50</td>
</tr>
<tr>
<td>isFeasible</td>
<td>51</td>
</tr>
<tr>
<td>isNormalized</td>
<td>52</td>
</tr>
<tr>
<td>isObviouslyInfeasible</td>
<td>52</td>
</tr>
<tr>
<td>isObviouslyRedundant</td>
<td>53</td>
</tr>
<tr>
<td>isSubset</td>
<td>54</td>
</tr>
<tr>
<td>localizeErrors</td>
<td>55</td>
</tr>
<tr>
<td>nedits</td>
<td>59</td>
</tr>
<tr>
<td>normalize</td>
<td>59</td>
</tr>
<tr>
<td>reduce</td>
<td>60</td>
</tr>
<tr>
<td>separate</td>
<td>61</td>
</tr>
<tr>
<td>substValue</td>
<td>62</td>
</tr>
<tr>
<td>violatedEdits</td>
<td>64</td>
</tr>
</tbody>
</table>

Index 68
Description

The editrules package aims to provide an environment to conveniently define, read and check recordwise data constraints including

- Linear (in)equality constraints for numerical data,
- Constraints on value combinations of categorical data
- Conditional constraints on numerical and/or mixed data

In literature these constraints, or restrictions are referred to as “edits”. editrules can perform common rule set manipulations like variable elimination and value substitution, and offers error localization functionality based on the (generalized) paradigm of Fellegi and Holt. Under this paradigm, one determines the smallest (weighted) number of variables to adapt such that no (additional or derived) rules are violated. The paradigm is based on the assumption that errors are distributed randomly over the variables and there is no detectable cause of error. It also decouples the detection of corrupt variables from their correction. For some types of error, such as sign flips, typing errors or rounding errors, this assumption does not hold. These errors can be detected and are closely related to their resolution. The reader is referred to the deducorrect package for treating such errors.

I. Define edits

editrules provides several methods for creating edits from a character, expression, data.frame or a text file.

- **editfile** Read conditional numerical, numerical and categorical constraints from textfile
- **editset** Create conditional numerical, numerical and categorical constraints
- **editmatrix** Create a linear constraint matrix for numerical data
- **editarray** Create value combination constraints for categorical data

II. Check and find errors in data

editrules provides several methods for checking data.frames with edits

- **violatedEdits** Find out which record violates which edit.
- **localizeErrors** Localize erroneous fields using Fellegi and Holt’s principle.
- **errorLocalizer** Low-level error localization function using B&B algorithm

Note that you can call plot, summary and print on results of these functions.

IV. Manipulate and check edits

editrules provides several methods for manipulating edits
### substValue
Substitute a value in a set of rules

### eliminate
Derive implied rules by variable elimination

### reduce
Remove unconstraint variables

### isFeasible
Check for contradictions

### duplicated
Find duplicated rules

### blocks
Decompose rules into independent blocks

### disjunct
Decouple conditional edits into disjunct edit sets

### separate
Decompose rules in blocks and decouple conditional edits

### generateEdits
Generate all nonredundant implicit edits (*editarray* only)

## V. Plot and coerce edits

*editrules* provides several methods for plotting and coercion.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>editrules.plotting</em></td>
<td>Plot edit-variable connectivity graph</td>
</tr>
<tr>
<td><em>as.igraph</em></td>
<td>Coerce to edit-variable connectivity igraph object</td>
</tr>
<tr>
<td><em>as.character</em></td>
<td>Coerce edits to character representation</td>
</tr>
<tr>
<td><em>as.data.frame</em></td>
<td>Store character representation in <em>data.frame</em></td>
</tr>
</tbody>
</table>

### adjacency
Derive adjacency matrix from collection of edits

## Description
A set of edits can be represented as a graph where every vertex is an edit. Two vertices are connected if they have at least one variable in *vars* in common.

## Usage

```r
adjacency(E, nodetype = c("all", "rules", "vars"), rules = rownames(E),
          vars = getVars(E), ...)
```

```r
## S3 method for class 'editmatrix'
adjacency(E, nodetype = c("all", "rules", "vars"),
          rules = rownames(E), vars = getVars(E), ...)
```

```r
## S3 method for class 'editarray'
adjacency(E, nodetype = c("all", "rules", "vars"),
          rules = rownames(E), vars = getVars(E), ...)
```

```r
## S3 method for class 'editset'
adjacency(E, nodetype = c("all", "rules", "vars"),
          rules = c(rownames(E$num), rownames(E$mixcat)), vars = getVars(E), ...)
```
### Arguments

- **E** (*editmatrix*, *editarray* or *editset*)
- **nodetype** adjacency between rules, vars or both?
- **rules** selection of edits
- **vars** selection of variables
- ... arguments to be passed to or from other methods
- **x** An object of class *editmatrix*, *editarray* or *editset*
- **weighted** see `graph.adjacency`

### Details

`adjacency` returns the adjacency matrix. The elements of the matrix count the number of variables shared by the edits indicated in the row- and column names. The adjacency matrix can be converted to an igraph object with `graph.adjacency` from the igraph package.

`as.igraph` converts a set of edits to an igraph object directly.

### Value

the adjacency matrix of edits in *E* with respect to the variables in *vars*

### See Also

`plot.editmatrix`, `plot.editarray`, `plot.editset`

### Examples

```r
## Examples with linear (in)equality edits

# load predefined edits from package
data(edits)
edits

# convert to editmatrix
E <- editmatrix(edits)
```
### Not run:
# (Note to reader: the Not run directive only prevents the example commands from
# running when package is built)

# Total edit graph
plot(E)

# Graph with dependent edits
plot(E, nodetype="rules")

# Graph with dependent variables
plot(E, nodetype="vars")

# Total edit graph, but with curved lines (option from igraph package)
plot(E, edge.curved=TRUE)

# graph, plotting just the connections caused by variable 't'
plot(E,vars='t')

### End(Not run)

# here's an example with a broken record.
\[ r \leftarrow c(ct = 100, ch = 30, cp = 70, p=30, t=130 ) \]
violatedEdits(E,r)
errorLocalizer(E,r)$searchBest()$adapt

# we color the violated edits and the variables that have to be adapted

### Not run
set.seed(1) # (for reproducibility)
plot(E,
    adapt=errorLocalizer(E,r)$searchBest()$adapt,
    violated=violatedEdits(E,r))
### End(Not run)

# extract total graph (as igraph object)
as.igraph(E)

# extract graph with edges related to variable 't' and 'ch'
as.igraph(E,vars=c('t','ch'))

# extract total adjacency matrix
adjacency(E)

# extract adjacency matrix related to variables t and 'ch'
adjacency(E,vars=c('t','ch'))

### Examples with categorical edits
# generate an editarray:
E <- editarray(expression(
  age %%in%% c("<15","16-65",">65"),
  employment %%in%% c("unemployed","employed","retired"),
  salary %%in%% c("none","low","medium","high"),
  if (age == "<15") employment="unemployed",
  if (salary != "none") employment != "unemployed",
  if (employment == "unemployed") salary == "none"))

## Not run:
# plot total edit graph
plot(E)

# plot with a different layout
plot(E,layout=layout.circle)

# plot edit graph, just the connections caused by 'salary'
plot(E,vars="salary")

## End(Not run)

# extract edit graph
as.igraph(E)

# extract edit graph, just the connections caused by 'salary'
as.igraph(E,vars="salary")

# extract adjacency matrix
adjacency(E)

# extract adjacency matrix, only caused by 'employment'
adjacency(E,vars="employment")

---

**as.editmatrix**

Coerce a matrix to an edit matrix.

**Description**

*as.editmatrix* interpretes the matrix as an editmatrix. The columns of the matrix are the variables and the rows are the edit rules (contraints).

**Usage**

```r
as.editmatrix(A, b = numeric(nrow(A)), ops = rep("==", nrow(A)), ...)
```
Arguments

- **A** matrix to be transformed into an `editmatrix`.
- **b** Constant, a numeric of length(nrow(x)), defaults to 0
- **ops** Operators, character of length(nrow(x)) with the equality operators, defaults to "=="
- ... further attributes that will be attached to the resulting editmatrix

Details

If only argument `x` is given (the default), the resulting editmatrix is of the form \( Ax = 0 \). This can be influenced by using the parameters `b` and `ops`.

Value

an object of class `editmatrix`.

See Also

`editmatrix`

description

- `x` may be an editset, editmatrix, editarray or character vector

Usage

`as.editset(x, ...)`

Arguments

- **x** object or vector to be coerced to an editset
- ... extra parameters that will be passed to `as.character`, if necessary
**as.lp.mip**

**Coerces a mip object into an lpSolve object**

**Description**

as.lp.mip transforms a mip object into a lpSolveApi object.

**Usage**

```
as.lp.mip(mip)
```

**Arguments**

- `mip`: object of type `mip`.

**See Also**

- `as.mip`, `make.lp`
Usage

```r
## S3 method for class 'editarray'
as.matrix(x, ...)

## S3 method for class 'editarray'
c(...)

editarray(editrules, sep = ":", env = parent.frame())

## S3 method for class 'editarray'
as.character(x, useIf = TRUE, datamodel = TRUE, ...)

## S3 method for class 'editarray'
as.data.frame(x, ...)

## S3 method for class 'editarray'
as.expression(x, ...)

## S3 method for class 'editarray'
summary(object, useBlocks = TRUE, ...)
```

Arguments

- `x` editarray object
- `...` further arguments passed to or from other methods
- `editrules` character or expression vector.
- `sep` textual separator, to be used internally for separating variable from category names.
- `env` environment to evaluate the rhs of `==` or `%in%` in.
- `useIf` logical. Use if( <condition> ) <statement> or !<condition> | <statement> ?
- `datamodel` logical. Include datamodel explicitly?
- `object` an R object
- `useBlocks` logical Summarize each block?

Value

- `as.matrix`: The boolean matrix part of the editarray.
- `editarray`: An object of class editarray
- `as.data.frame`: data.frame with columns 'name', 'edit' and 'description'.

See Also

- `editrules.plotting, violatedEdits, localizeErrors, editfile, editset, editmatrix, getVars, blocks, eliminate, substValue, isFeasible generateEdits, contains, is.editarray, isSubset`
Examples

# Here is the prototypical categorical edit: men cannot be pregnant.
E <- editarray(expression(
  gender %in% c('male', 'female'),
  pregnant %in% c('yes', 'no'),
  if( gender == 'male' ) pregnant == 'no'
)
)
E

# an editarray has a summary method:
summary(E)

# A yes/no variable may also be modeled as a logical:
editarray(expression(
  gender %in% c('male', 'female'),
  pregnant %in% c(TRUE, FALSE),
  if( gender == 'male' ) pregnant == FALSE
)
)

# or, shorter (and using a character vector as input):
editarray(expression(
  gender %in% c('male', 'female'),
  pregnant %in% c(TRUE, FALSE),
  if( gender == 'male' ) !pregnant
)
)

# the \%in\% statement may be used at will
editarray(expression(
  gender %in% c('male', 'female'),
  pregnant %in% c(TRUE, FALSE),
  positionInHousehold %in% c('marriage partner', 'child', 'other'),
  maritalStatus %in% c('unmarried', 'married', 'widowed', 'divorced'),
  if( gender == 'male' ) !pregnant,
  if( maritalStatus %in% c('unmarried',
    'widowed',
    'divorced')
  ) !positionInHousehold %in% c('marriage partner', 'child')
)

# Here is the prototypical categorical edit: men cannot be pregnant.
E <- editarray(expression(
  gender %in% c('male', 'female'),
  pregnant %in% c('yes', 'no'),
  if( gender == 'male' ) pregnant == 'no'
)
)
```r
pregnant %in% c('yes','no'),
if( gender == 'male' ) pregnant == 'no'
)
}

# an editarray has a summary method:
summary(E)

# A yes/no variable may also be modeled as a logical:
editarray(expression(
  gender %in% c('male','female'),
  pregnant %in% c(TRUE, FALSE),
  if( gender == 'male' ) pregnant == FALSE
)
)

# or, shorter (and using a character vector as input):
editarray(expression(
  gender %in% c('male','female'),
  pregnant %in% c(TRUE, FALSE),
  if( gender == 'male' ) !pregnant
)
)

# the %in% statement may be used at will
editarray(expression(
  gender %in% c('male','female'),
  pregnant %in% c(TRUE, FALSE),
  positionInHousehold %in% c('marriage partner', 'child', 'other'),
  maritalStatus %in% c('unmarried','married','widowed','divorced'),
  if( gender == 'male' ) !pregnant,
  if( maritalStatus %in% c('unmarried',
      'widowed',
      'divorced')
  ) !positionInHousehold %in% c('marriage partner','child')
)
)
```

---

**Description**

convert to matrix
Combine editmatrices

An editmatrix is a numerical matrix and a set of comparison operators representing a linear system of (in)equations.

Summary

Usage

```r
## S3 method for class 'editmatrix'
as.matrix(x, ...)

## S3 method for class 'editmatrix'
c(...)

editmatrix(editrules, normalize = TRUE)

## S3 method for class 'editmatrix'
as.data.frame(x, ...)

## S3 method for class 'editmatrix'
as.character(x, ...)

## S3 method for class 'editmatrix'
as.expression(x, ...)

## S3 method for class 'editmatrix'
str(object, ...)

## S3 method for class 'editmatrix'
summary(object, useBlocks = TRUE, ...)
```

Arguments

- `x`: editmatrix object
- `...`: Arguments to pass to or from other methods
- `editrules`: A character or expression vector with (in)equalities written in R syntax. Alternatively, a data.frame with a column named edits, see details.
- `normalize`: logical specifying if all edits should be transformed (see description)
- `object`: an R object
- `useBlocks`: logical Summarize each block?

Details

The function `editmatrix` generates an editmatrix from a character vector, an expression vector or a data.frame with at least the column edit. The function `editfile` reads edits from a free-form textfile, function `as.editmatrix` converts a matrix, a vector of constants and a vector of operators to an editmatrix.
By default, the `editmatrix` is normalized, meaning that all comparison operators are converted to one of `<`, `<=`, or `==`. Users may specify edits using any of the operators `<`, `<=`, `==`, `>=`, `>` (see examples below). However it is highly recommended to let `editmatrix` parse them into normal form as all functions operating on editmatrices expect or convert it to normal form anyway.

**Value**

- **as.matrix**: Augmented matrix of `editmatrix`. (See also `getAb`).
- **editmatrix**: An object of class `editmatrix`.
- **as.data.frame**: a 3-column data.frame with columns 'name' and 'edit'. If the input `editmatrix` has a description attribute a third column is returned.

**Note**

Since version 2.0-0, the behaviour of `as.data.frame.editmatrix` changed to be more symmetrical with `editmatrix.data.frame` and `as.data.frame.editarray`. Use `editrules:::toDataFrame` (unsupported) for the old behaviour.

**See Also**

`editrules.plotting`, `violatedEdits`, `localizeErrors`, `normalize`, `contains`, `is.editmatrix`, `getA`, `getAb`, `getb`, `getOps`, `getVars`, `eliminate`, `substValue`, `isFeasible`

**Examples**

```r
# Using a character vector to define constraints
E <- editmatrix(c("x+3y==2*z", "x==z"))
print(E)

# Using an expression vector to define constraints
E <- editmatrix(expression(x+3*y==2*z, x==z))
print(E)

# an editmatrix also has a summary method:
summary(E)

# select rows from an editmatrix:
E <- editmatrix(c("x+3y==2*z", "x >= z"))
E[getOps(E) == "="]

# Using data.frame to define constraints
E.df <- data.frame(
  name =c("A","B","C"),
  edit = c("x == y",
           "z + w == y + x",
           "z == y + 2*w"),
  description = c("these variables should be equal","","")
)
```
as.mip  

Write an editset into a mip representation

Description

Writes an editset or an object coercable to an editset as a mip problem.

Usage

as.mip(E, x = NULL, weight = NULL, M = 1e+07, epsilon = 0.001, prefix = "delta.", ...)
backtracker

Arguments

- E: an `editset` or an object that is coerciable to an `editset`
- x: a named list/vector with variable values
- weight: reliability weights for values of x
- M: Constant that is used for allowing the values to differ from x
- epsilon: Constant that is used for converting '<' into '<='
- prefix: prefix for dummy variables that are created
- ...: not used

Value

- a `mip` object containing all information for transforming it into an lp/mip problem

Description

`backtracker` creates a binary search program that can be started by calling the `searchNext` function. It walks a binary tree depth first. For all left nodes `choiceLeft` is evaluated, for all right nodes `choiceRight` is evaluated. A solution is found if `isSolution` evaluates to `true`. In that case `searchNext` will return all variables in the search environment in a list. If `isSolution` evaluates to NULL, it will continue to search deeper. If `isSolution` evaluates to `false` it stops at the current node and goes up the next search node.

Usage

```plaintext
backtracker(isSolution, choiceLeft, choiceRight, list = NULL,
            maxdepth = Inf, maxduration = Inf, ...)
```

Arguments

- `isSolution`: expression that should evaluate to `TRUE` when a solution is found.
- `choiceLeft`: expression that will be evaluated for a left node
- `choiceRight`: expression that will be evaluated for a right node
- `list`: list with variables that will be added to the search environment
- `maxdepth`: integer maximum depth of the search tree
- `maxduration`: integer Default maximum search time for `searchNext()` and `searchAll()`
- ...: named variables that will be added to the search environment
blocks

Details

Methods:

$\text{searchNext(..., VERBOSE=FALSE)}$ Search next solution, can be called repeatedly until there is no solution left. Named variables will be added to the search environment, this feature can be used to direct the search in subsequent calls to searchNext. VERBOSE=TRUE will print all intermediate search steps and results. It can be used to debug the expressions in the backtracker.

$\text{searchAll(..., VERBOSE=FALSE)}$ Return all solutions as a list.

$\text{reset()}$ Resets the backtracker to its initial state.

Value

backtracker object, see Methods for a description of the methods

Examples

```r
bt <- backtracker( isSolution= {
  if (y == 0) return(TRUE)
  if (x == 0) return(FALSE)
}
, choiceLeft = ( x <- x - 1; y <- y)
, choiceRight = ( y <- y - 1; x <- x)
# starting values for x and y
, x=2
, y=1
)
bt$\text{searchNext(VERBOSE=TRUE)}
bt$\text{searchNext(VERBOSE=TRUE)}

# next search will return NULL because there is no more solution
bt$\text{searchNext()}
bt$\text{reset()}
```

blocks

Decompose a matrix or edits into independent blocks

Description

blocks returns a list of independent blocks $M_i$ such that $M = M_1 \oplus M_2 \oplus \cdots \oplus M_n$.

blockIndex returns a list of row indices in a logical matrix $D$ designating independent blocks.
Usage
blocks(M)
blockIndex(D)

Arguments
M       matrix, editmatrix, editarray or editset to be decomposed into independent blocks
D       matrix of type logical

Value
list of independent subobjects of M.
list of row indices in D indicating independent blocks. Empty rows (i.e. every column FALSE) are ignored.

Examples
# three separate blocks
E <- editmatrix(expression(
  x1 + x2 == x3,
  x3 + x4 == x5,
  x5 + x6 == x7,
  y1 + y2 == y3,
  z1 + z2 == z3
))
blocks(E)

# four separate blocks
E <- editmatrix(expression(
  x1 + x2 == x3,
  x3 + x4 == x5,
  x8 + x6 == x7,
  y1 + y2 == y3,
  z1 + z2 == z3
))
blocks(E)

# two categorical blocks
E <- editarray(expression(
  x %in% c('a','b','c'),
  y %in% c('d','e'),
  z %in% c('f','g'),
  u %in% c('w','t'),
  if ( x == 'a') y != 'd',
  if ( z == 'f') u != 'w'
))
blocks(E)
Description

An editset combines numerical (linear), categorical and conditional restrictions in a single object. Internally, it consists of two editmatrices and an editarray.

Usage

```r
## S3 method for class 'editset'
c(...)
editset(editrules, env = new.env())

## S3 method for class 'editset'
as.character(x, datamodel = TRUE, useIf = TRUE,
              dummies = FALSE, ...)

## S3 method for class 'editset'
as.data.frame(x, ...)

## S3 method for class 'editset'
summary(object, useBlocks = TRUE, ...)
```

Arguments

- `...`: arguments to be passed to or from other methods
- `editrules`: character vector, expression vector or data.frame (see details) containing edits.
- `env`: environment to parse categorical edits in (normally, users need not specify this)
- `x`: an `editset`
- `datamodel`: include datamodel?
- `useIf`: return vectorized version?
- `dummies`: return datamodel for dummy variables?
- `object`: an R object
- `useBlocks`: logical Summarize each block?

Details

The function `editset` converts a character or expression vector to an editset. Alternatively, a data.frame with a column called `edit` can be supplied. Function `editfile` reads edits from a free-form textfile.
Value

editset: An object of class editset

as.data.frame: a data.frame with columns 'name' and 'edit'.

See Also

editrules.plotting, violatedEdits, localizeErrors, getVars, disjunct, eliminate, substValue, isFeasible, contains, is.editset

Examples

# edits can be read from a vector of expressions
E <- editset(expression(
  if ( x > 0 ) y > 0,
  x + y == z,
  A %in% letters[1:2],
  B %in% letters[2:3],
  if ( A == 'a') B == 'b',
  if ( A == 'b') x >= 0,
  u + v == w,
  if ( u >= 0 ) w >= 0
))
E
summary(E)
as.data.frame(E)
getVars(E)
getVars(E,type='cat')
getVars(E,type='num')

## see also editfile
E <- editfile(system.file('script/edits/mixedits.R',package='editrules'))
E
summary(E)
as.data.frame(E)
getVars(E)
getVars(E,type='cat')
getVars(E,type='num')

# edits can be read from a vector of expressions
E <- editset(expression(
  if ( x > 0 ) y > 0,
  x + y == z,
  A %in% letters[1:2],
  B %in% letters[2:3],
  if ( A == 'a') B == 'b',
  if ( A == 'b') x >= 0,
checkDatamodel

_**Check data against a datamodel**_

**Description**

Categorical variables in `dat` which also occur in `E` are checked against the datamodel for those variables. Numerical variables are checked against edits in `E` that contain only a single variable (e.g. \( x > 0 \)). Values violating such edits as well as empty values are set to adapt.

**Usage**

`checkDatamodel(E, dat, weight = rep(1, ncol(dat)), ...)`

**Arguments**

- `E` 
  an object of class `editset, editarray, or editmatrix`
- `dat` 
  a `data.frame`
- `weight` 
  vector of weights for every variable of `dat` or an array of weight of the same dimensions as `dat`.
- `...` 
  arguments to be passed to or from other methods

**Value**

An object of class `errorLocation`. 

```r
u + v == w,
if ( u >= 0 ) w >= 0
})
E
summary(E)
as.data.frame(E)
getVars(E)
getVars(E,type='cat')
getVars(E,type='num')
```

```r
## see also editfile
E <- editfile(system.file('script/edits/mixedits.R',package='editrules'))
E
summary(E)
as.data.frame(E)
getVars(E)
getVars(E,type='cat')
getVars(E,type='num')
```
See Also

`errorLocation, localizeErrors`.

---

### condition

*Get condition matrix from an editset.*

**Description**

Get condition matrix from an editset.

**Usage**

```r
condition(E)
```

**Arguments**

- `E` an **editset**

**Value**

an **editmatrix**, holding conditions under which the editset is relevant.

**See Also**

`disjunct, separate, editset`

---

### datamodel

*Summarize data model of an editarray in a data.frame*

**Description**

Summarize data model of an editarray in a data.frame

**Usage**

```r
datamodel(E)
```

**Arguments**

- `E` **editarray**

**Value**

data.frame describing the categorical variables and their levels.
disjunct

See Also

checkDatamodel

Examples

e <- editarray(expression(
    age %in% c('under aged', 'adult'),
    positionInHousehold %in% c('marriage partner', 'child', 'other'),
    maritalStatus %in% c('unmarried', 'married', 'widowed', 'divorced'),
    if (maritalStatus %in% c('married', 'widowed', 'divorced')) positionInHousehold != 'child',
    if (age == 'under aged') maritalStatus == 'unmarried'
  )
)
datamodel(E)

disjunct

Decouple a set of conditional edits

Description

An editset is transformed to a list of editsets which do not contain any conditional numeric/categorical edits anymore. Each editset gains an attribute condition, which holds the series of assumptions made to decouple the original edits. This attribute will be printed when not NULL. Warning: this may be slow for large, highly entangled sets of edits.

Usage

disjunct(E, type = c("list", "env"))

Arguments

E Object of class editset
type Return type: list (default) for editlist, env for editenv.

Value

An object of class editlist (editenv), which is nothing more than a list (environment) of editsets with a class attribute. Each element has an attribute 'condition' showing which conditions were assumed to derive the editset.

See Also

separate, condition, blocks
Examples

E <- editset(expression(
  x + y == z,
  if (x > 0) y > 0,
  x >= 0,
  y >= 0,
  z >= 0,
  A %in% letters[1:4],
  B %in% letters[1:4],
  if (A %in% c('a','b')) y > 0,
  if (A == 'c') B %in% letters[1:3]
))

disjunct(E)

---

**echelon**

*Bring an (edit) matrix to reduced row echelon form.*

Description

If E is a matrix, a matrix in reduced row echelon form is returned. If E is an *editmatrix* the equality part of E is transformed to reduced row echelon form. For an *editset*, the numerical part is transformed to reduced row echelon form.

Usage

echelon(E, ...)

## S3 method for class 'editmatrix'
echelon(E, ...)

## S3 method for class 'matrix'
echelon(E, tol = sqrt(.Machine$double.eps), ...)

## S3 method for class 'editset'
echelon(E, ...)

Arguments

- **E** a matrix or editmatrix
- **...** options to pass on to further methods.
- **tol** tolerance that will be used to determine if a coefficient equals zero.
**editfile**

*Read edits edits from free-form textfile*

**Description**

This utility function allows for free editrule definition in a file. One can extract only the numerical (type='num'), only the categorical (type='cat') or all edits (default) in which case an **editset** is returned. The function first parses all assignments in the file, so it is possible to compute or read a list of categories defining a datamodel for example.

**Usage**

```
editfile(file, type = c("all", "num", "cat", "mix"), ...)
```

**Arguments**

- **file**
  name of text file to read in
- **type**
  type of edits to extract. Currently, only 'num' (numerical), 'cat' (categorical) and 'all' are implemented.
- **...**
  extra parameters that are currently ignored

**Value**

- **editset** with all edits if type=all, **editarray** if type='cat', **editmatrix** if type='num', **editset** with conditional edits if type='mix'. If the return value is a list, the elements are named numedits and catedits.

**editnames**

*Names of edits*

**Description**

Retrieve edit names from editset, -array or -matrix

**Usage**

```
editnames(E)
```

**Arguments**

- **E**
  editset, editarray or editmatrix
**Description**

Plots a graph, showing which variables occur in what edits. By default, squares represent edits, circles represent variables and an edge connecting a variable with an edit indicates that the edit contains the variable.

**plot method for editarray**

**plot method for editset**

**Usage**

```r
## S3 method for class 'editmatrix'
plot(x, nodetype = "all", rules = editnames(x),
     vars = getVars(x), violated = logical(nedits(x)),
     adapt = logical(length(getVars(x))), nabbreviate = 5,
     layout = igraph::layout.fruchterman.reingold, edgecolor = "steelblue",
     rulecolor = "khaki1", varcolor = "lightblue1",
     violatedcolor = "sienna1", adaptcolor = "sienna1", ...)

## S3 method for class 'editarray'
plot(x, nodetype = "all", rules = editnames(x),
     vars = getVars(x), violated = logical(nedits(x)),
     adapt = logical(length(getVars(x))), nabbreviate = 5,
     layout = igraph::layout.fruchterman.reingold, edgecolor = "steelblue",
     rulecolor = "khaki1", varcolor = "lightblue1",
     violatedcolor = "sienna1", adaptcolor = "sienna1", ...)

## S3 method for class 'editset'
plot(x, nodetype = "all", rules = editnames(x),
     vars = getVars(x), violated = logical(nedits(x)),
     adapt = logical(length(getVars(x))), nabbreviate = 5,
     layout = igraph::layout.fruchterman.reingold, edgecolor = "steelblue",
     rulecolor = "khaki1", varcolor = "lightblue1",
     violatedcolor = "sienna1", adaptcolor = "sienna1", ...)
```

**Arguments**

- **x** object of class `editmatrix`
- **nodetype** 'rules', 'vars' or 'all'.
- **rules** selection of edits
- **vars** selection of variables
- **violated** A named logical vector of length `nrow(E)`. Ignored when `nodetype='vars`
- **adapt** A named logical vector of length(`getVars(E)`). Ignored when `nodetype='rules'`
editrules.plotting

nabbreviate integer To how many characters should variable and edit names be abbreviated?
layout an igraph layout function. See \texttt{igraph::layout}
edgecolor Color of edges and node frames
rulecolor Color of rule nodes (ignored when nodetype='vars')
varcolor Color of variable nodes (ignored when nodetype='rules')
vviolatedcolor Color of nodes corresponding to violated edits (ignored when nodetype='vars')
adaptcolor Color of nodes corresponding to variables to adapt (ignored when nodetype='rules')
... further arguments to be passed to plot.

Details

Depending on the chosen \texttt{nodetype}, this function can plot three types of graphs based on an edit set.

- If \texttt{nodetype}="all" (default), the full bipartite graph is plotted. Each variable is represented by a square node while each edit is represented by a circular node. An edge is drawn when a variable occurs in an edit.
- If \texttt{nodetype}="vars" the variable graph is drawn. Each node represents a variable, and an edge is drawn between two nodes if the variables occur together in at least one edit. The edge width relates to the number of edits connecting two variables.
- If \texttt{nodetype}="rules" the rule graph is drawn. Each node represents an edit rule and an edge is drawn between two nodes if they share at least one variable. The edge width relates to the number of edits connecting the two edit rules.

The boolean vectors \texttt{violated} and \texttt{adapt} can be used to color violated edits or variables which have to be adapted. The vectors must have named elements, so variables and edit names can be matched.

The function works by coercing an editmatrix to an \texttt{igraph} object, and therefore relies on the plotting capabilities of the \texttt{igraph} package. For more finetuning, use \texttt{as.igraph} and see \texttt{igraph.plotting}.

The default layout generated by the Fruchterman-Reingold algorithm. The resulting layout is one of several optimal layouts, generated randomly (using an attraction-repulsion model between the nodes). To reproduce layouts, use fix a randseed before calling the plot function.

References


See Also

\texttt{as.igraph, adjacency, igraph.plotting}
Examples

```r
## Examples with linear (in)equality edits

# load predefined edits from package
data(edits)
edits

# convert to editmatrix
E <- editmatrix(edits)

## Not run:
# (Note to reader: the Not run directive only prevents the example commands from
# running when package is built)

# Total edit graph
plot(E)

# Graph with dependent edits
plot(E, nodetype="rules")

# Graph with dependent variables
plot(E, nodetype="vars")

# Total edit graph, but with curved lines (option from igraph package)
plot(E, edge.curved=TRUE)

# graph, plotting just the connections caused by variable 't'
plot(E, vars='t')

## End(Not run)

# here's an example with a broken record.
r <- c(ct = 100, ch = 30, cp = 70, p=30,t=130)
violatedEdits(E,r)
errorLocalizer(E,r)$searchBest()$adapt

# we color the violated edits and the variables that have to be adapted

## Not run
set.seed(1) # (for reproducibility)
plot(E,
   adapt=errorLocalizer(E,r)$searchBest()$adapt,
   violated=violatedEdits(E,r))

## End(Not run)

# extract total graph (as igraph object)
as.igraph(E)
```
# extract graph with edges related to variable 't' and 'ch'
as.igraph(E,vars=c('t','ch'))

# extract total adjacency matrix
adjacency(E)

# extract adjacency matrix related to variables t and 'ch'
adjacency(E,vars=c('t','ch'))

## Examples with categorical edits

# generate an editarray:
E <- editarray(expression(  
age %in% c('<15','16-65','>65'),  
employment %in% c('unemployed','employed','retired'),  
salary %in% c('none','low','medium','high'),  
if (age == '<15') employment='unemployed',  
if (salary != 'none') employment != 'unemployed',  
if (employment == 'unemployed') salary == 'none'))

## Not run:
# plot total edit graph
plot(E)

# plot with a different layout
plot(E,layout=layout.circle)

# plot edit graph, just the connections caused by 'salary'
plot(E,vars='salary')

## End(Not run)

# extract edit graph
as.igraph(E)

# extract edit graph, just the connections caused by 'salary'
as.igraph(E,vars='salary')

# extract adjacency matrix
adjacency(E)

# extract adjacency matrix, only caused by 'employment'
adjacency(E,vars='employment')
**Description**

Eliminate a variable from a set of edit rules

Eliminating a variable amounts to deriving all (non-redundant) edits not containing that variable. Geometrically, it can be seen as a projection of the solution space (records obeying all edits) along the eliminated variable’s axis. If the solution space is non-concex (as is the usually case when conditional edits are involved), multiple projections of convex subregions are performed.

For objects of class `editmatrix`, Fourier-Motzkin elimination is used to eliminate a variable from the of linear (in)equality restrictions. An observation of Kohler (1967) is used to reduce the number of implied restrictions. Obvious redundancies of the type $0 < 1$ are removed as well.

For categorical edits in an `editarray`, the elimination method is based on repeated logical reduction on categories. See Van der Loo (2012) for a description.

For an `editset`, $E$ is transformed to an `editlist`. Each element of an `editlist` describes a convex subregion of the total solution space of the `editset`. After this, the elimination method for `editlist` is called.

For an `editlist`, the variable is eliminated from each constituting `editset`.

**Usage**

```
edata(eds)```

**Arguments**

- `E` editset
- `m` if you happen to have `contains(E)` handy, it needs not be recalculated.

**See Also**

contains
eliminate

Usage

    eliminate(E, var, ...)

    # S3 method for class 'editmatrix'
    eliminate(E, var, ...)

    # S3 method for class 'editarray'
    eliminate(E, var, ...)

    # S3 method for class 'editset'
    eliminate(E, var, ...)

    # S3 method for class 'editlist'
    eliminate(E, var, ...)

Arguments

    E     editmatrix or editarray
    var   name of variable to be eliminated
    ...   arguments to be passed to or from other methods

Value

    If E is an editmatrix or editarray, an object of the same class is returned. A returned editmatrix contains an extra history attribute which is used to reduce the number of generated edits in consecutive eliminations (see getH). If E is an editset, an object of class editlist is returned.

References


See Also

    substValue, isObviousInfeasible, isObiouslyRedundant, generateEdits

Examples

    # The following is an example by Williams (1986). Eliminating all variables except z maximizes -4x1 + 5x2 +3x3:
    P <- editmatrix(c(
        "4*x1 - 5*x2 - 3*x3 + z <= 0",
        "-x1 + x2 -x3 <= 2",
        "x1 + x2 + x3 <= 2",
        "x1, x2, x3, z >= 0",
    )
    eliminate(P, "z")
"x1 + x2 + 2*x3 <= 3",
"-x1 <= 0",
"-x2 <= 0",
"-x3 <= 0")
# eliminate 1st variable
(P1 <- eliminate(P, "x1", fancynames=TRUE))
# eliminate 2nd variable. Note that redundant rows have been eliminated
(P2 <- eliminate(P1, "x2", fancynames=TRUE))
# finally, the answer:
(P3 <- eliminate(P2, "x3", fancynames=TRUE))

# check which original edits were used in deriving the new ones
geth(P3)
# check how many variables were eliminated
geth(P3)

# An example with an equality and two inequalities
# The only thing to do is solving for x in e1 and substitute in e3.
(E <- editmatrix(c(
  "2*x + y == 1",
  "y > 0",
  "x > 0"),normalize=TRUE))
eliminate(E,"x", fancynames=TRUE)

# This example has two equalities, and it's solution
# is the origin (x,y)=(0,0)
(E <- editmatrix(c(
  "y <= 1 - x",
  "y >= -1 + x",
  "x == y",
  "y == -2*x") ),normalize=TRUE))
eliminate(E,"x", fancynames=TRUE)

# this example has no solution, the equalities demand (x,y) = (0,2)
# while the inequalities demand y <= 1
(E <- editmatrix(c(
  "y <= 1 - x",
  "y >= -1 + x",
  "y == 2 - x",
  "y == -2 + x") ),normalize=TRUE))
# this happens to result in an obviously unfeasable system:
isObviouslyInfeasible(eliminate(E,"x"))

# for categorical data, elimination amounts to logical derivartions. For example
E <- editarray(expression(
  age %in% c('under aged','adult'),
  positionInHousehold %in% c('marriage partner','child','other'),
  maritalStatus %in% c('unmarried','married','widowed','divorced'),
...
if (maritalStatus %in% c('married', 'widowed', 'divorced'))
    positionInHousehold = 'child',
if (maritalStatus == 'unmarried')
    positionInHousehold = 'marriage partner',
if (age == 'under aged') maritalStatus = 'unmarried'
)
)
E

# by eliminating 'maritalStatus' we can deduce that under aged persons cannot
# be partner in marriage.
eliminate(E, "maritalStatus")

E <- editarray(expression(
    age %in% c('under aged', 'adult'),
    positionInHousehold %in% c('marriage partner', 'child', 'other'),
    maritalStatus %in% c('unmarried', 'married', 'widowed', 'divorced'),
    if (maritalStatus %in% c('married', 'widowed', 'divorced'))
        positionInHousehold != 'child',
    if (maritalStatus == 'unmarried')
        positionInHousehold != 'marriage partner',
    if (age == 'under aged')
        maritalStatus = 'unmarried'
)
)
E

# by eliminating 'maritalStatus' we can deduce that under aged persons cannot
# be partner in marriage.
eliminate(E, "maritalStatus")

---

**errorLocalizer**

Create a backtracker object for error localization

**Description**

Create a backtracker object for error localization

**Usage**

errorLocalizer(E, x, ...)

# S3 method for class 'editset'
errorLocalizer(E, x, ...)
## S3 method for class 'editmatrix'
errorLocalizer(E, x, weight = rep(1, length(x)),
               maxadapt = length(x), maxweight = sum(weight), maxduration = 600,
               tol = sqrt(.Machine$double.eps), ...)

## S3 method for class 'editarray'
errorLocalizer(E, x, weight = rep(1, length(x)),
               maxadapt = length(x), maxweight = sum(weight), maxduration = 600, ...)  

## S3 method for class 'editlist'
errorLocalizer(E, x, weight = rep(1, length(x)),
               maxadapt = length(x), maxweight = sum(weight), maxduration = 600, ...)

### Arguments

- **E**
  - an `editmatrix` or an `editarray`
- **x**
  - a named numerical vector or list (if E is an editmatrix), a named character vector or list (if E is an editarray), or a named list if E is an `editlist` or `editset`. This is the record for which errors will be localized.
- **weight**
  - a `length(x)` positive weight vector. The weights are assumed to be in the same order as the variables in `x`.
- **maxadapt**
  - maximum number of variables to adapt
- **maxweight**
  - maximum weight of solution, if weights are not given, this is equal to the maximum number of variables to adapt.
- **maxduration**
  - maximum time (in seconds), for `$searchNext()`, `$searchAll()` (not for `$searchBest`, use `$searchBest(maxduration=<duration>)` in stead)
- **tol**
  - tolerance passed to link{isObviouslyInfeasible} (used to check for bound conditions).

### Value

- an object of class `backtracker`. Each execution of `$searchNext()` yields a solution in the form of a `list` (see details). Executing `$searchBest()` returns the lowest-weight solution. When multiple solutions with the same weight are found, `$searchBest()` picks one at random.

### Details

Generate a `backtracker` object for error localization in numerical, categorical, or mixed data. This function generates the workhorse program, called by `localizeErrors` with method=`localizer`.

The returned `backtracker` can be used to run a branch-and-bound algorithm which finds the least (weighted) number of variables in `x` that need to be adapted so that all restrictions in `E` can be satisfied. (Generalized principle of Fellegi and Holt (1976)).

The B&B tree is set up so that in in one branche, a variable is assumed correct and its value substituted in `E`, while in the other branche a variable is assumed incorrect and `eliminated` from `E`. See De Waal (2003), chapter 8 or De Waal, Pannekoek and Scholtus (2011) for a concise description of the B&B algorithm.
Every call to `<backtracker>$searchNext()` returns one solution list, consisting of

- `w`: The solution weight.
- `adapt`: logical indicating whether a variable should be adapted (TRUE) or not

Every subsequent call leads either to NULL, in which case either all solutions have been found, or `maxduration` was exceeded. The property `<backtracker>$maxdurationExceeded` indicates if this is the case. Otherwise, a new solution with a weight `w` not higher than the weight of the last found solution is returned.

Alternatively `<backtracker>$searchBest()` will return the best solution found within `maxduration` seconds. If multiple equivalent solutions are found, a random one is returned.

The backtracker is prepared such that missing data in the input record `x` is already set to adapt, and missing variables have been eliminated already.

The backtracker will crash when `E` is an `editarray` and one or more values are not in the data-model specified by `E`. The more user-friendly function `localizeErrors` circumvents this. See also `checkDatamodel`.

**Numerical stability issues**

For records with a large numerical range (eg 1-1E9), the error locations represent solutions that will allow repairing the record to within roundoff errors. We highly recommend that you round near-zero values (for example, everything \( \leq \sqrt{\text{Machine}\$\text{double.eps}} \)) and scale a record with values larger than or equal to 1E9 with a constant factor.

**Note**

This method is potentially very slow for objects of class `editset` that contain many conditional restrictions. Consider using `localizeErrors` with the option `method="mip"` in such cases.

**References**


**See Also**

`errorLocalizer_mip`, `localizeErrors`, `checkDatamodel`, `violatedEdits`,
cp <- errorLocalizer(E, x=c(p=755, c=125, t=200))
# x obviously violates \( E \). With all weights equal, changing any variable will do.
# first solution:
cp$searchNext()
# second solution:
cp$searchNext()
# third solution:
cp$searchNext()
# there are no more solutions since changing more variables would increase the
# weight, so the result of the next statement is NULL:
cp$searchNext()

# Increasing the reliability weight of turnover, yields 2 solutions:

cp <- errorLocalizer(E, x=c(p=755, c=125, t=200), weight=c(1,1,2))
# first solution:
cp$searchNext()
# second solution:
cp$searchNext()
# no more solutions available:
cp$searchNext()

# A case with two restrictions. The second restriction demands that
# \( c/t \geq 0.6 \) (cost should be more than 60% of turnover)
E <- editmatrix(c(
  "p + c == t",
  "c - 0.6*t >= 0""))
cp <- errorLocalizer(E, x=c(p=755, c=125, t=200))
# Now, there's only one solution, but we need two runs to find it (the 1st one
# has higher weight)
cp$searchNext()
cp$searchNext()

# With the searchBest() function, the lowest weight solution is found at once:
errorLocalizer(E, x=c(p=755, c=125, t=200))$searchBest()

# An example with missing data.
E <- editmatrix(c(
  "p + c1 + c2 == t",
  "c1 - 0.3*t >= 0",
  "p > 0",
  "c1 > 0",
  "c2 > 0",
  "t > 0"))
cp <- errorLocalizer(E, x=c(p=755, c1=50, c2=NA, t=200))
# (Note that e2 is violated.)
# There are two solutions. Both demand that \( c2 \) is adapted:
cp$searchNext()
cp$searchNext()

##### Examples with categorical edits


errorLocalizer_mip

# 3 variables, recording age class, position in household, and marital status:
# We define the datamodel and the rules
E <- editarray(expression(
    age %in% c('under aged', 'adult'),
    maritalStatus %in% c('unmarried', 'married', 'widowed', 'divorced'),
    positionInHousehold %in% c('marriage partner', 'child', 'other'),
    if( age == 'under aged' )
        maritalStatus == 'unmarried',
    if( maritalStatus %in% c('married', 'widowed', 'divorced') )
        positionInHousehold %in% c('marriage partner', 'child')
)
E

# Let's define a record with an obvious error:
R <- c(  
    age = 'under aged',  
    maritalStatus = 'married',  
    positionInHousehold = 'child')
# The age class and position in household are consistent, while the marital
# status conflicts. Therefore, changing only the marital status (in stead of
# both age class and position in household) seems reasonable.
El <- errorLocalizer(E, R)
El$searchNext()

errorLocalizer_mip    Localize errors using a MIP approach.

Description

Localize errors using a MIP approach.

Usage

errorLocalizer_mip(E, x, weight = rep(1, length(x)), maxduration = 600L,  
    verbose = "neutral", lpcontrol =getOption("er.lpcontrol"), ...)

__errorLocalizer_mip Localize errors using a MIP approach.____
errorLocation

Arguments

- `E` an editset, editmatrix, or editarray
- `x` named numeric with data
- `weight` numeric with weights
- `maxduration` number of seconds that is spent on finding a solution
- `verbose` verbosity argument that will be passed on to solve lpSolveAPI
- `lpcontrol` named list of arguments that will be passed on to `lp.control`, `maxduration` will override lpSolve’s timeout argument.
  ...
  other arguments that will be passed on to `solve`.

Value

list with solution weight `w`, logical `adapt` stating what to adapt, `x_feasible` and the lp problem (an `lpExtPtr` object)

Details

errorLocalizer_mip uses `E` and `x` to define a mixed integer problem and solves this problem using lpSolveAPI. This function can be much faster than `errorLocalizer` but does not return the degeneracy of a solution. However it does return an bonus: `x_feasible`, a feasible solution.

References

E. De Jonge and Van der Loo, M. (2012) Error localization as a mixed-integer program in editrules (included with the package)


See Also

localizeErrors, errorLocalizer, errorLocation

errorLocation  The errorLocation object

Description

Object storing information on error locations in a dataset.

summary

Usage

```r
## S3 method for class 'errorLocation'
plot(x, topn = min(10, ncol(x$adapt)), ...)

## S3 method for class 'errorLocation'
summary(object, ...)
```
Arguments

x  errorLocation object

`topn`  Number of variables to show in ‘errors per variable plot’. Only the top-n are are shown. By default the top-20 variables with the most errors are shown.

...  other arguments that will be transferred to `barplot`

object  an R object

Details

The `errorLocation` objects consists of the following slots which can be accessed with the dollar operator, just like with lists. Right now the only functions creating such objects are `localizeErrors` and `checkDatamodel`.

- `adapt` a logical array where each row/column shows which record/variable should be adapted.
- `status` A data.frame with the same number of rows as `adapt`. It contains the following columns
  - `weight` weight of the found solution
  - `degeneracy` number of equivalent solutions found
  - `user` user time used to generate solution (as in `sys.time`)
  - `system` system time used to generate solution (as in `sys.time`)
  - `elapsed` elapsed time used to generate solution (as in `sys.time`)
  - `maxDurationExceeded` Was the maximum search time reached?
  - `memFail` Indicates whether a branch was broken off due to memory allocation failure (branch and bound only)
- `method` The error localization method used, can be "mip", "localizer" or "checkDatamodel".
- `call` The R calls to the function generating the object.
- `user` character user who generated the object.
- `timestamp` character timestamp.

It is possible to plot objects of class `errorLocation`. An overview containing three or four graphs will be plotted in a new window. Axes in scatterplots are set to logarithmic if their scales maxima exceed 50.

See Also

`localizeErrors`, `checkDatamodel`

Examples

```r
# an editmatrix and some data:
E <- editmatrix(c(
  "x + y == z",
  "x > 0",
  "y > 0",
  "z > 0"))
```
dat <- data.frame(
  x = c(1,-1,1),
  y = c(-1,1,1),
  z = c(2,0,2))

# localize all errors in the data
err <- localizeErrors(E,dat)

summary(err)

# what has to be adapted:
err$adapt
# weight, number of equivalent solutions, timings,
err$status

## Not run

# Demonstration of verbose processing
# construct 2-block editmatrix
F <- editmatrix(c(
  "x + y == z",
  "x > 0",
  "y > 0",
  "z > 0",
  "w > 10")
)
# Using 'dat' as defined above, generate some extra records
dd <- dat
for (i in 1:5) dd <- rbind(dd,dd)
dd$w <- sample(12,nrow(dd),replace=TRUE)

# localize errors verbosely
(err <- localizeErrors(F,dd,verbose=TRUE))

# printing is cut off, use summary for an overview
summary(err)

# or plot (not very informative in this artificial example)
plot(err)

## End(Not run)

for (d in dir("../pkg/R",full.names=TRUE)) dmp <- source(d)
# Example with different weights for each record
E <- editmatrix('x + y == z')
dat <- data.frame(
  x = c(1,1),
  y = c(1,1),
  z = c(1,1))

# At equal weights, both records have three solutions (degeneracy): adapt x, y
# or z:
localizeErrors(E,dat)$status

# Set different weights per record (lower weight means lower reliability):
w <- matrix(c(
  1,2,2,
  2,2,1),nrow=2,byrow=TRUE)

localizeErrors(E,dat,weight=w)

# an example with categorical variables
E <- editarray(expression(  
age %in% c('under aged','adult'),
  maritalStatus %in% c('unmarried','married','widowed','divorced'),
  positionInHousehold %in% c('marriage partner','child','other'),
  if( age == 'under aged' ) maritalStatus == 'unmarried',
  if( maritalStatus %in% c('married','widowed','divorced'))  
    !positionInHousehold %in% c('marriage partner','child')
  )
)
E

dat <- data.frame(  
age = c('under aged','adult','adult'),
  maritalStatus=c('married','unmarried','widowed'),
  positionInHousehold=c('child','other','marriage partner')
)
dat
localizeErrors(E,dat)

# the last record of dat has 2 degenerate solutions. Running the last command
# a few times demonstrates that one of those solutions is chosen at random.

# Increasing the weight of 'positionInHousehold' for example, makes the best
# solution unique again
localizeErrors(E,dat,weight=c(1,1,2))

# an example with mixed data:
E <- editset(expression(  
x + y == z,
  2*u + 0.5*v == 3*w,
  w >= 0,
  if ( x > 0 ) y > 0,
  x >= 0,
  y >= 0,
  z >= 0,
  A %in% letters[1:4],
  B %in% letters[1:4],
  C %in% c(TRUE,FALSE),
  D %in% letters[5:8],
  if ( A %in% c('a','b') ) y > 0,
generateEdits

Derive all essentially new implicit edits

description
Implement the Field Code Forest (FCF) algorithm of Garfinkel et al (1986) to derive all essentially new implicit edits from an editarray. The FCF is really a single, highly unbalanced tree. This algorithm traverses the tree, pruning many unnecessary branches, uses blocks to divide and conquer, and optimizes traversing order. See Van der Loo (2012) for a description of the algorithms.

usage
generateEdits(E)

arguments
E An editarray

value
A 3-element named list, where element E is an editarray containing all generated edits. nodes contains information on the number of nodes in the tree and vs the number of nodes traversed and duration contains user, system and elapsed time in seconds. The summary method for editarray prints this information.
getA

References
M.P.J. Van der Loo (2012). Variable elimination and edit generation with a flavour of semigroup algebra (submitted)

getA

Returns the coefficient matrix A of linear (in)equalities

Description
Returns the coefficient matrix A of linear (in)equalities

Usage
geta(E)

Arguments

E editmatrix

Value
numeric matrix A

See Alsoeditmatrix

Examples
E <- editmatrix(c( "x+3*y >= 2*z" 
                    , "x > 2"
                    )
print(E)

# get editrules, useful for storing and maintaining the rules external from your script
as.data.frame(E)

# get coefficient matrix of inequalities
geta(E)

# get augmented matrix of linear edit set
getAb(E)

# get constants of inequalities (i.e. c(0, 2))
getb(E)
getAb

Returns augmented matrix representation of edit set.

Description
For a system of linear (in)equations of the form \( Ax \odot b \), \( \odot \in \{ <, \leq, = \} \), the matrix \( A|b \) is called the augmented matrix.

Usage
getAb(E)

Arguments
E editmatrix

Value
numeric matrix \( A|b \)

See Also
editmatrix as.matrix.editmatrix

Examples
E <- editmatrix(c("x+3*y == 2*z","x > 2")
print(E)

# get editrules, useful for storing and maintaining the rules external from your script
getb

Returns the constant part b of a linear (in)equality

Description

Returns the constant part b of a linear (in)equality

Usage

getb(E)

Arguments

E editmatrix

Value

numeric vector b

See Also

editmatrix
Examples

E <- editmatrix(c("x+3*y == 2*z" , "x > 2")
)

print(E)

# get editrules, useful for storing and maintaining the rules external from your script
as.data.frame(E)

# get coefficient matrix of inequalities
geta(E)

# get augmented matrix of linear edit set
getaB(E)

# get constants of inequalities (i.e. c(0, 2))
getb(E)

# get operators of inequalities (i.e. c("=",">"))
getops(E)

# get variables of inequalities (i.e. c("x","y","z"))
getvars(E)

# isNormalized
isNormalized(E)

#normalized E
E <- normalize(E)
E

# is het now normalized?
isNormalized(E)

getH

Returns the derivation history of an edit matrix or array

Description

Function eliminate tracks the history of edits in a logical array H. H has nrow(E) rows and the number of columns is the number of edits in the editmatrix as it was first defined. If H[i,j1], H[i,j2],...,H[i,jn] are TRUE, then E[i] is some (positive, linear) combination of original edits E[j1,]. E[j2,]...E[jn,]

h records the number of variables eliminated from E by eliminate
getOps

Usage
getH(E)
geth(E)

Arguments
E editmatrix

Details
Attributes H and h are used to detect redundant derived edits.

See Also
editmatrix, eliminate
editmatrix, eliminate

getOps

Returns the operator part of a linear (in)equality editmatrix E

Description
Returns the operator part of a linear (in)equality editmatrix E

Usage
getOps(E)

Arguments
E editmatrix

Value
character vector with the (in)equality operators.

See Also
editmatrix
Examples

```r
E <- editmatrix(c("x+3*y == 2*z",
                  "x > 2")
)
print(E)

# get editrules, useful for storing and maintaining the rules external from your script
as.data.frame(E)

# get coefficient matrix of inequalities
geta(E)

# get augmented matrix of linear edit set
getAb(E)

# get constants of inequalities (i.e. c(0, 2))
getb(E)

# get operators of inequalities (i.e. c("==","reater than"))
getOps(E)

# get variables of inequalities (i.e. c("x","y","z"))
getVars(E)

# isNormalized
isNormalized(E)

#normalized E
E <- normalize(E)
E

# is het now normalized?
isNormalized(E)
```

---

**getVars**

*get names of variables in a set of edits*

**Description**

get names of variables in a set of edits

**Usage**

```r
getVars(E, ...)
```

## S3 method for class 'editset'
### impliedValues

Retrieve values strictly implied by rules

```r
getVars(E, type = c("all", "num", "cat", "mix", "dummy"),
...) # S3 method for class 'NULL'
getVars(E, ...)
```

**Arguments**

- `E` — `editset`, `editmatrix`, or `editarray`

- `...` — Arguments to be passed to or from other methods

- `type` — (editset- or list only) select which variables to return. `all` means all (except dummies), `num` means all numericals, `cat` means all categoricals, `mix` means those numericals appearing in a logical constraint and `dummy` means dummy variables connecting the logical with numerical constraints.

**Value**

character vector with the names of the variables.

**See Also**

`getA`, `getb`, `getAb`, `getOps`

**Examples**

```r
E <- editmatrix(c("x+3*y == 2*z",
                  "x > 2")
getVars(E)

E <- editarray(expression(
    gender %in% c('male', 'female'),
    pregnant %in% c(TRUE, FALSE),
    if( gender == 'male' ) pregnant == FALSE
  )
getVars(E)
```
Description

Retrieve values strictly implied by rules

Detects cases where two inequalities imply an equality, e.g. \(x \leq 0\) and \(x \geq 0\) implies \(x = 0\). Also detects straight equalities, e.g. \(x == 0\) implies \(x = 0\). Such cases arise frequently when manipulating edits by value substitution or variable elimination. The function recursively detects equalities and combined inequalities that imply fixed values, substitutes those fixed values and looks for new implied values until no new values are found.

Usage

\[
\text{impliedValues}(E, \ldots)
\]

## S3 method for class 'editmatrix'

\[
\text{impliedValues}(E, \text{tol} = \text{sqrt(.Machine$double.eps)}, \ldots)
\]

Arguments

- \(E\) editmatrix
- \(\ldots\) Currently unused
- \(\text{tol}\) Maximum deviation for two values to be considered equal.

Value

Numeric vector, whose names are variable names and values are unique values implied by the rules.

See Also

\[\text{reduce, substValue, eliminate}\]

is.editrules

Check object class

Description

Check object class

Usage

\[
is.editset(x)
\]

\[
is.editmatrix(x)
\]

\[
is.editarray(x)
\]

Arguments

- \(x\) object to be checked
Description

When variables are eliminated one by one from a set of edits, eventually either no edits are left or an obvious contradiction is encountered. In the case no records can obey all edits in the set which is therefore infeasible.

Usage

isFeasible(E, warn = FALSE)

Arguments

E an editmatrix, editarray or editset

warn logical: should a warning be emitted when system is infeasible?

Value

TRUE or FALSE

Note

This function can potentially take a long time to complete, especially when many connected (conditional) edits are present. Consider using blocks to check feasibility of indendent blocks.

See Also

isObviouslyInfeasible, isObviouslyRedundant
**isNormalized**  
*Check if an editmatrix is normalized*

**Description**  
Check if an editmatrix is normalized

**Usage**  
isNormalized(E)

**Arguments**
- **E** editmatrix

**Value**
TRUE when all comparison operators of E are in \{<,<=,==\}

**See Also**
- editmatrix

---

**isObviouslyInfeasible**  
*Check for obvious contradictions in a set of edits*

**Description**  
Obvious contradictions are edits of the form 1 < 0, or categorical edits defining that a record fails for any value combination If this function evaluates to TRUE, the set of edits is guaranteed infeasible. If it evaluates to FALSE this does not guarantee feasibility. See isFeasible for a complete test.

**Usage**
- isObviouslyInfeasible(E, ...)
  
  ## S3 method for class 'editmatrix'
  isObviouslyInfeasible(E, tol = sqrt(.Machine$double.eps), ...)

  ## S3 method for class 'editarray'
  isObviouslyInfeasible(E, ...)

  ## S3 method for class 'editset'
  isObviouslyInfeasible(E, ...)
isObviouslyRedundant

## S3 method for class 'editlist'
isObviousInfeasible(E, ...)

## S3 method for class 'editenv'
isObviousInfeasible(E, ...)

### Arguments

- **E**
  - An `editset`, `editmatrix`, `editarray`, `editlist` or `editenv`.
- **...**
  - Arguments to be passed to or from other methods.
- **tol**
  - Tolerance for checking against zero.

### Value

A logical for objects of class `editset`, `editarray` or `editmatrix`. A logical vector in the case of an `editlist` or `editset`.

### See Also

- `isObviouslyRedundant`, `isFeasible`
- `eliminate editmatrix`

---

**isObviouslyRedundant**  
*Find obvious redundancies in set of edits*

### Description

Detect simple redundancies such as duplicates or edits of the form $0 < 1$ or $0 == 0$. For categorical edits, simple redundancies are edits that define an empty subregion of the space of all possible records (no record can ever be contained in such a region).

### Usage

```r
isObviouslyRedundant(E, duplicates = TRUE, ...)
```

## S3 method for class 'editmatrix'
isObviouslyRedundant(E, duplicates = TRUE, ...)

## S3 method for class 'editarray'
isObviouslyRedundant(E, duplicates = TRUE, ...)

## S3 method for class 'editset'
isObviouslyRedundant(E, duplicates = rep(TRUE, 2), ...)

## S3 method for class 'editlist'
isObviouslyRedundant(E, duplicates = rep(TRUE, 2), ...)

```
isSubset

Check which edits are dominated by other ones.

Description

An edit defines a subregion of the space of all possible value combinations of a record. Records in this region are interpreted as invalid. An edit rule which defines a region equal to or contained in the region defined by another edit is redundant. (In data editing literature, this is often referred to as a domination relation.)

Usage

isSubset(E)

Arguments

E editarray

Value

logical vector indicating if an edit is a subset of at least one other edit.
Localize errors on records in a `data.frame`.

**Description**

For each record in a `data.frame`, the least (weighted) number of fields is determined which can be adapted or imputed so that no edit in `E` is violated. Anymore.

**Usage**

```r
localizeErrors(E, dat, verbose = FALSE, weight = rep(1, ncol(dat)),
    maxduration = 600, method = c("bb", "mip", "localizer"),
    useBlocks = TRUE, retrieve = c("best", "first"), ...)
```

**Arguments**

- `E` an object of class `editset`, `editmatrix` or `editarray`
- `dat` a `data.frame` with variables in `E`
- `verbose` print progress to screen?
- `weight` Vector of positive weights for every variable in `dat`, or an `array` or `data.frame` of weights with the same dimensions as `dat`.
- `maxduration` maximum time for `DsearchBest()` to find the best solution for a single record.
- `method` should `errorlocalizer` ("bb") or mix integer programming ("mip") be used?
- `useBlocks` DEPRECATED. Process error localization seperately for independent blocks in `E` (always TRUE)?
- `retrieve` Return the first found solution or the best solution? ("bb" method only).
- `...` Further options to be passed to `errorLocalizer` or `errorLocalizer_mip`. Specifically, when `method='mip'`, the parameter `lpcontrol` is a list of options passed to lpSolveAPI.

**Details**

For performance purposes, the edits are split in independent blocks which are processed separately. Also, a quick vectorized check with `checkDatamodel` is performed first to exclude variables violating their one-dimensional bounds from further calculations.

By default, all weights are set equal to one (each variable is considered equally reliable). If a vector of weights is passed, the weights are assumed to be in the same order as the columns of `dat`. By passing an array of weights (of same dimensions as `dat`) separate weights can be specified for each record.

In general, the solution to an error localization problem need not be unique, especially when no weights are defined. In such cases, `localizeErrors` chooses a solution randomly. See `errorLocalizer` for more control options.

Error localization can be performed by the Branch and Bound method of De Waal (2003) (option `method="localizer"`, the default) or by rewriting the problem as a mixed-integer programming
(MIP) problem (method="mip") which is parsed to the lp.solve library. The former case uses errorLocalizer and is very reliable in terms of numerical stability, but may be slower in some cases (see note below). The MIP approach is much faster, but requires that upper and lower bounds are set on each numerical variable. Sensible bounds are derived automatically (see the vignette on error localization as MIP), but could cause instabilities in very rare cases.

Value

an object of class errorLocation

Note

As of version 2.8.1 method 'bb' is not available for conditional numeric (e.g: if (x>0) y>0) or conditional edits of mixed type (e.g. if (A=='a') x>0).

References

E. De Jonge and Van der Loo, M. (2012) Error localization as a mixed-integer program in editrules (included with the package)

See Also

errorLocalizer

Examples

# an editmatrix and some data:
E <- editmatrix(c(
   "x + y == z",
   "x > 0",
   "y > 0",
   "z > 0"))

dat <- data.frame(
   x = c(1,-1,1),
   y = c(-1,1,1),
   z = c(2,0,2))

# localize all errors in the data
err <- localizeErrors(E,dat)

summary(err)

# what has to be adapted:
err$adapt
# weight, number of equivalent solutions, timings,
err$status
## Not run

### Demonstration of verbose processing

```r
# construct 2-block editmatrix
F <- editmatrix(c(
  "x + y == z",
  "x > 0",
  "y > 0",
  "z > 0",
  "w > 10"))
```

```r
# Using 'dat' as defined above, generate some extra records
dd <- dat
for (i in 1:5) dd <- rbind(dd, dd)
dd$w <- sample(12, nrow(dd), replace=TRUE)
```

```r
# localize errors verbosely
(err <- localizeErrors(F, dd, verbose=TRUE))
```

```r
# printing is cut off, use summary for an overview
summary(err)
```

```r
# or plot (not very informative in this artificial example)
plot(err)
```

### End(Not run)

```r
for (d in dir("./pkg/R", full.names=TRUE)) dmp <- source(d)
```

### Example with different weights for each record

```r
E <- editmatrix("x + y == z")
dat <- data.frame(
  x = c(1,1),
  y = c(1,1),
  z = c(1,1))
```

```r
# At equal weights, both records have three solutions (degeneracy): adapt x, y
# or z:
localizeErrors(E, dat)$status
```

```r
# Set different weights per record (lower weight means lower reliability):
weight <- matrix(c(1,2,2,
  2,2,1), nrow=2, byrow=TRUE)
```

```r
localizeErrors(E, dat, weight=weight)
```

### an example with categorical variables

```r
E <- editarray(expression(
  age %in% c('under aged', 'adult'),
  maritalStatus %in% c('unmarried', 'married', 'widowed', 'divorced'),
  positionInHousehold %in% c('marriage partner', 'child', 'other'),
)
if( age == 'under aged') maritalStatus == 'unmarried',
  if( maritalStatus %in% c('married','widowed','divorced'))
    positionInHousehold %in% c('marriage partner','child')
  )
} E

#

dat <- data.frame(
  age = c('under aged','adult','adult'),
  maritalStatus=c('married','unmarried','widowed'),
  positionInHousehold=c('child','other','marriage partner')
)

dat localizeErrors(E,dat)
# the last record of dat has 2 degenerate solutions. Running the last command
# a few times demonstrates that one of those solutions is chosen at random.

# Increasing the weight of 'positionInHousehold' for example, makes the best
# solution unique again
localizeErrors(E,dat,weight=c(1,1,2))

# an example with mixed data:

E <- editset(expression(
  x + y == z,
  2*u + 0.5*v == 3*w,
  w >= 0,
  if ( x > 0 ) y > 0,
  x >= 0,
  y >= 0,
  z >= 0,
  A %in% letters[1:4],
  B %in% letters[1:4],
  C %in% c(TRUE,FALSE),
  D %in% letters[5:8],
  if ( A %in% c('a','b') ) y > 0,
  if ( A == 'c' ) B %in% letters[1:3],
  if ( !C == TRUE) D %in% c('e','f')
))

set.seed(1)
dat <- data.frame(
  x = sample(-1:8),
  y = sample(-1:8),
  z = sample(10),
  u = sample(-1:8),
  v = sample(-1:8),
  w = sample(10),
  A = sample(letters[1:4],10,replace=TRUE),
  B = sample(letters[1:4],10,replace=TRUE),
  C = sample(c(TRUE,FALSE),10,replace=TRUE),
  D = sample(c('e','f'),10,replace=TRUE),
  E = sample(c(TRUE,FALSE),10,replace=TRUE)
)
nedits

Number of edits Count the number of edits in a collection of edits.

Description

Number of edits Count the number of edits in a collection of edits.

Usage

nedits(E)

Arguments

E editset, editarray or editmatrix

normalize

Normalizes an editmatrix

Description

An set of linear edits of the form \( a \cdot x \odot b \) with is called normalized when all \( \odot \in \{=, \leq, <\} \)

Usage

normalize(E)

Arguments

E editmatrix

Value

If \( E \) was normalized, the original editmatrix is returned, otherwise a new normalized editmatrix will be returned
reduce

Remove redundant variables and edits.

Description

Remove variables which are not contained in any edit and remove edits which are obviously redundant.
Usage

reduce(E, ...)

## S3 method for class 'editmatrix'
reduce(E, tol = sqrt(.Machine$double.eps), ...)

## S3 method for class 'editarray'
reduce(E, ...)

## S3 method for class 'editset'
reduce(E, ...)

Arguments

E editmatrix or editarray
... arguments to pass to other methods
tol elements of E with absolute value < tol are considered 0.

See Also

contains, eliminate, substValue

Description

The input edits are separated into disjunct blocks, and simplified to editmatrix or editarray where possible. Remaining editsets are separated into disjunct editlists.

Usage

separate(E)

Arguments

E An editset

Value

A list where each element is either an editmatrix, an editarray or an object of class editlist which cannot be simplified further.

References

See Also
blocks.disjunct.condition

Examples

E <- editset(expression(
  x + y == z,
  2*x + 0.5*y == 3*w,
  w >= 0,
  if ( x > 0 ) y > 0,
  x >= 0,
  y >= 0,
  z >= 0,
  A %in% letters[1:4],
  B %in% letters[1:4],
  C %in% c(TRUE,FALSE),
  D %in% letters[5:8],
  if ( A %in% c('a','b') ) y > 0,
  if ( A == 'c' ) B %in% letters[1:3],
  if ( !C == TRUE) D %in% c('e','f')
))

(L <- separate(E))

sapply(L, class)

-----------

substValue

Replace a variable by a value in a set of edits.

Description
Replace a variable by a value in a set of edits.

Usage

substValue(E, var, value, ...)

## S3 method for class 'editmatrix'
substValue(E, var, value, reduce = FALSE,
removeredundant = TRUE, ...)
The `substValue` function is designed to replace variables with values in different objects like `editset`, `editmatrix`, `editarray`, `editlist`, and `editenv`. Here's a detailed look at how it works:

### Arguments

- **E**
  - `editset, editmatrix, editarray, editlist` or `editenv`
- **var**
  - Character with name(s) of variable(s) to substitute
- **value**
  - Vector with value(s) of variable(s)
- **reduce**
  - Logical; should the result be simplified? For `editmatrix` this has the same effect as calling the function `reduce`. For `editarray`, the datamodel of the substituted variable is reduced to a single value, and the variable itself is not removed.
- **removeredundant**
  - Logical. Should empty rows be removed?
- **simplify**
  - Simplify editset by moving logical edits containing a single numerical statement to the pure numerical part? (This is mostly for internal purposes and overwriting the default should normally not be necessary for package users).

### Value

- E, with variables replaced by values

### Note

At the moment, objects of class `editenv` are converted to `list` prior to processing (so no performance is gained there) and reconverted afterwards.

### References

Value substitution is extensively described in the package vignettes.

### See Also

- `eliminate`
Examples

E <- editmatrix(expression(
  x + y == z,
  2*y < 10,
  3*x + 1.5*u < 7,
  z >= 0
)
)

# single value
substValue(E, 'z', 10)
# multiple values
substValue(E,c('x','y'),c(1,3))
# remove substituted variable from edits
substValue(E, 'z', 10, reduce=TRUE)
# do not remove redundant row:
substValue(E, 'z', 10, removedundant=FALSE)

# example with an editset
E <- editset(expression(
  x + y == z,
  x >= 0,
  y >= 0,
  A %in% c('a1','a2'),
  B %in% c('b1','b2'),
  if ( x > 0 ) y > 0,
  if ( y > 0 ) x > 0,
  if ( A == 'a' ) B == 'b',
  if ( A == 'b' ) y > 3
)
)

# substitute pure numerical variable
substValue(E, 'z', 10)
# substitute pure categorical variable
substValue(E, 'A', 'a1')
# substitute variable appearing in logical constraints
substValue(E, 'x', 3)
Description

Determine which record violates which edits. Returns NA when edits cannot be checked because of missing values in the data.

- For rules of the form \( Ax = b \) \( |Ax - b| \leq \text{tol} \) is returned.
- For rules of the form \( Ax < b \), \( Ax - b < \text{tol} \) is returned.
- For rules of the form \( Ax \leq b \), \( Ax - b \leq \text{tol} \) is returned.

For numerical records, the default tolerance is 0. When working with doubles, the square root of machine accuracy is a reasonable alternative (\( \sqrt{\text{.Machine}\$\text{double}\$.\text{eps}} \)). The editmatrix is normalized before checks are performed.

Plot summary statistics on violatedEdits

as.data.frame violatedEdits

Usage

\[
\text{violatedEdits}(E, \text{dat}, ...) \\
\text{"S3 method for class 'character'"} \\
\text{violatedEdits}(E, \text{dat}, \text{name = NULL, ...}) \\
\text{"S3 method for class 'editmatrix'"} \\
\text{violatedEdits}(E, \text{dat}, \text{tol = 0, ...}) \\
\text{"S3 method for class 'editarray'"} \\
\text{violatedEdits}(E, \text{dat}, \text{datamodel = TRUE, ...}) \\
\text{"S3 method for class 'editset'"} \\
\text{violatedEdits}(E, \text{dat}, \text{datamodel = TRUE, ...}) \\
\text{"S3 method for class 'violatedEdits'"} \\
\text{plot(x, toppn = min(10, ncol(x)), ...)} \\
\text{"S3 method for class 'violatedEdits'"} \\
\text{summary(object, E = NULL, minfreq = 1, ...)} \\
\text{"S3 method for class 'violatedEdits'"} \\
\text{as.data.frame(x, ...)}
\]

Arguments

- **E** character vector with constraints, editset, editmatrix or editarray.
- **dat** data.frame with data that should be checked, if a named vector is supplied it will converted internally to a data.frame
- **...** further arguments that can be used by methods implementing this generic function
- **name** name of edits
tol 

tolerance to check rules against.

datamodel 

Also check against datamodel?

x 

violatedEdits object.

topn 

Top n edits to be plotted.

object 

violatedEdits object

minfreq 

minimum freq for edit to be printed

Value

An object of class violatedEdits, which is a logical nrow(dat)Xn edits(E) matrix with an extra class attribute for overloading purposes.

Note

When summarizing an object of class violatedEdits, every empty value is counted as one edit violation when counting violations per record.

See Also

checkDatamodel

Examples

# Using character vector to define contraints
E <- editmatrix(c("x+3*y==2*z",
                   "x==z")
               )

dat <- data.frame(x = c(0,2,1),
                   y = c(0,0,1),
                   z = c(0,1,1))
print(dat)

ve <- violatedEdits(E,dat)

print(ve)
summary(ve, E)
plot(ve)

# An example with categorical data:

E <- editarray(expression(
    gender %in% c('male','female'),
    pregnant %in% c(TRUE, FALSE),
    if( gender == 'male' ) !pregnant)
)
print(E)
dat <- data.frame(
    gender=c('male','male','female','cylon'),
    pregnant=c(TRUE,FALSE,TRUE,TRUE)
)

print(dat)
# Standard, the datamodel is checked as well,
violatedEdits(E,dat)

# but we may turn this of
violatedEdits(E,dat,datamodel=FALSE)
Index

*Topic data
  edits, 30

adjacency, 4, 27
as.character.editarray
  (as.matrix.editarray), 9
as.character.editmatrix
  (as.matrix.editmatrix), 12
as.character.editset(\texttt{c.editset}), 19
as.data.frame.editarray
  (as.matrix.editarray), 9
as.data.frame.editmatrix
  (as.matrix.editmatrix), 12
as.data.frame.editset(\texttt{c.editset}), 19
as.data.frame.violatedEdits
  (violatedEdits), 64
as.editmatrix, 7, 13
as.editset, 8
as.expression.editarray
  (as.matrix.editarray), 9
as.expression.editmatrix
  (as.matrix.editmatrix), 12
as.igraph, 4, 27
as.igraph.editarray(\texttt{adjacency}), 4
as.igraph.editmatrix(\texttt{adjacency}), 4
as.igraph.editset(\texttt{adjacency}), 4
as.lp.mip, 9
as.matrix.editarray, 9
as.matrix.editmatrix, 12, 44
as.mip, 9, 15

backtracker, 16, 34
blockIndex(\texttt{blocks}), 17
blocks, 4, 10, 17, 23, 42, 51, 55, 62
c.editarray(\texttt{as.matrix.editarray}), 9
c.editmatrix(\texttt{as.matrix.editmatrix}), 12
c.editset, 19
character, 65
checkDatamodel, 21, 23, 35, 39, 55, 66

choicepoint (\texttt{backtracker}), 16
condition, 22, 23, 62
contains, 10, 14, 20, 30, 61
data.frame, 9
datamodel, 22
disjunct, 4, 20, 22, 23, 61, 62
duplicated, 4
echelon, 24
editarray, 3–5, 19, 21, 22, 25, 30, 31, 34, 35, 38, 42, 49, 51, 53–55, 59, 61, 63, 65
editarray (\texttt{as.matrix.editarray}), 9
ditenv, 53, 54, 63
ditfile, 3, 9, 10, 13, 19, 25
ditlist, 30, 31, 34, 53, 54, 61, 63
ditlists, 61
ditmatricontents, 19
editmatrix, 3, 5, 8, 10, 18, 21, 22, 24–26, 30, 31, 34, 38, 43–47, 49, 51–55, 59–61, 63, 65
editmatrix (\texttt{as.matrix.editmatrix}), 12
ditnames, 25
ditrules-package, 3
ditrules.plotting, 4, 10, 14, 20, 26
dits, 29
ditset, 3, 5, 10, 19, 21–25, 30, 31, 34, 35, 38, 49, 51, 53–55, 59, 61, 63, 65
ditset(\texttt{c.editset}), 19
ditsets, 61
ditsetType, 30
eliminate, 4, 10, 14, 20, 25, 30, 46, 47, 50, 53, 61, 63
eliminated, 34, 51
derrorLocalizer, 3, 33, 38, 55, 56
derrorLocalizer.mip, 35, 37, 55
derrorLocation, 21, 22, 38, 38, 56
generateEdits, 4, 10, 31, 42
getA, 14, 43, 49
INDEX

getAb, 14, 44, 49
getb, 14, 45, 49
geth, 31, 46
geth (getH), 46
getVars, 14, 47, 49

impliedValues, 49
is.editarray, 10
is.editarray (is.editrules), 50
is.editmatrix, 14
is.editmatrix (is.editrules), 50
is.editrules, 50
is.editset, 20
is.editset (is.editrules), 50
isFeasible, 4, 10, 14, 20, 51, 52, 53
isNormalized, 52
isObviouslyInfeasible, 31, 51, 52, 54
isObviouslyRedundant, 31, 51, 53, 53
isSubset, 10, 54, 54

localizeErrors, 3, 10, 14, 20, 22, 34, 35, 38, 39, 55
lp.control, 38

make.lp, 9

nedits, 59
normalize, 14, 59
normalized, 65

plot.editarray, 5
plot.editarray (editrules.plotting), 26
plot.editmatrix, 5
plot.editmatrix (editrules.plotting), 26
plot.editset, 5
plot.editset (editrules.plotting), 26
plot.errorLocation (errorLocation), 38
plot.violatedEdits (violatedEdits), 64

reduce, 4, 50, 60, 63

separate, 4, 22, 23, 61
str.editmatrix (as.matrix.editmatrix), 12
substValue, 4, 10, 14, 20, 25, 31, 50, 61, 62
summary, 42
summary.editarray
(as.matrix.editarray), 9
summary.editmatrix
(as.matrix.editmatrix), 12
summary.editset (c.editset), 19
summary.errorLocation (errorLocation), 38
summary.violatedEdits (violatedEdits), 64

violatedEdits, 3, 10, 14, 20, 35, 64