Package ‘errorlocate’

May 3, 2021

Type Package
Title Locate Errors with Validation Rules
Version 0.9.9
Description Errors in data can be located and removed using validation rules from package ‘validate’.
License GPL-3
URL https://github.com/data-cleaning/errorlocate
BugReports https://github.com/data-cleaning/errorlocate/issues
Depends validate
Imports lpSolveAPI, methods, parallel
Suggests testthat (>= 2.1.0), covr, knitr, rmarkdown
RoxygenNote 7.1.1
Encoding UTF-8
VignetteBuilder knitr
NeedsCompilation no
Author Edwin de Jonge [aut, cre] (<https://orcid.org/0000-0002-6580-4718>),
Mark van der Loo [aut]
Maintainer Edwin de Jonge <edwindjonge@gmail.com>
Repository CRAN
Date/Publication 2021-05-03 13:50:02 UTC

R topics documented:

  errorlocate-package ................................................................. 2
  add_noise ............................................................................. 3
Description

Find errors in data given a set of validation rules. The errorlocate helps to identify obvious errors in raw datasets.

Details

It works in tandem with the package validate(). With validate you formulate data validation rules to which the data must comply. For example:

"age cannot be negative": age >= 0

While validate can identify if a record is valid or not, it does not identify which of the variables are responsible for the invalidation. This may seem a simple task, but is actually quite tricky: a set of validation rules form a web of dependent variables: changing the value of an invalid record to repair for rule 1, may invalidate the record for rule 2.

Errorlocate provides a small framework for record based error detection and implements the Felligi Holt algorithm. This algorithm assumes there is no other information available then the values of a record and a set of validation rules. The algorithm minimizes the (weighted) number of values that need to be adjusted to remove the invalidation.

The errorlocate package translates the validation and error localization problem into a mixed integer problem and uses a mip solver to find a solution.

Author(s)

Maintainer: Edwin de Jonge <edwindjonge@gmail.com> (ORCID)

Authors:

• Mark van der Loo <mark.vanderloo@gmail.com>
add_noise

References

Van der Loo, M., de Jonge, E, Data Cleaning With Applications in R
version 5.5.2.0-5. http://CRAN.R-project.org/package=lpSolveAPI

See Also

Useful links:
• https://github.com/data-cleaning/errorlocate
• Report bugs at https://github.com/data-cleaning/errorlocate/issues

add_noise  Add (a small amount of) noise

Description

Utility function to add some small positive noise to weights. This is mainly done to randomly
choose between solutions of equal weight. Without adding noise to weights lp solvers may return
an identical solution over and over while there are multiple solutions of equal weight. The generated
noise is positive to prevent that weights will be zero or negative.

Usage

add_noise(x, max_delta = NULL, ...)

Arguments

x numeric vector or matrix. When x is a matrix, the function will be applied to
each row of the matrix.
max_delta when supplied noise will be drawn from [0,max_delta] otherwise see details
... currently not used

Details

When no max_delta is supplied, add_noise will use the minimum difference larger than zero di-
vided by the length(x).

Value

numeric vector/matrix with noise applied.
ErrorLocalizer-class  Base class for class locate errors based on rules and data

Description

ErrorLocalizer can be used as a base class to implement a new error localization algorithm. The derived class must implement two methods: initialize, which is called before any error localization is done and locate which operates upon data. The extra parameter ... can used to supply algorithmic specific parameters.

errorlocation-class  Error location object

Description

Errorlocation contains the result of a error detection. Errors can record based or variable based.

- A record based error is restricted within one observation. errorlocate() using the Felligi Holt algorithm assumes errors are record based.
- A variable based error is a flaw in uni- or multivariate distribution. To correct this error multiple observations or the aggregated number should be adjusted.

Details

Current implementation assumes that errors are record based. The error locations can be retrieved using the method values() and are a matrix of rows and columns, with the same dimensions are the data.frame that was checked. For errors that are purely column based, or dataset based, errorlocations will return a matrix with all rows or cells set to TRUE. The values() return NA for missing values.

Fields

- $errors: matrix indicating which values are erroneous (TRUE), missing (NA) or valid (FALSE)
- $weight: The total weight per record. A weight of 0 means no errors were detected.
- $status: The status of the mip solver for this record.
- $duration: The number of seconds for processing each record.

See Also

Other error finding: errors_removed(), locate_errors(), replace_errors()
errors_removed  Get location of removed errors from a `cleaned` data set

Description

errors_removed retrieves the errors detected by replace_errors()

Usage

errors_removed(x, ...)

Arguments

x data.frame that was checked for errors
...
not used

Value

errorlocation-class() object

See Also

Other error finding: errorlocation-class, locate_errors(), replace_errors()

Examples

rules <- validator( profit + cost == turnover
  , cost - 0.6*turnover >= 0
  , cost>= 0
  , turnover >= 0
)
data <- data.frame(profit=755, cost=125, turnover=200)
data_no_error <- replace_errors(data, rules)

# faulty data was replaced with NA
data_no_error

errors_removed(data_no_error)

# a bit more control, you can supply the result of locate_errors
# to replace_errors, which is a good thing, otherwise replace_errors will call
# locate_errors internally.
error_locations <- locate_errors(data, rules)
replace_errors(data, error_locations)
FHLocalizer-class
Feligi-Holt Errorlocalizer

Description
Implementation of the Feligi-Holt algorithm using the ErrorLocalizer base class. Given a set of validation rules and a dataset the Feligi-Holt algorithm finds for each record the smallest (weighted) combination of variables that are erroneous (if any).

Note
Most users do not need this class and can use locate_errors(). errorlocalizer implements feligi holt using a MIP-solver. For problems in which coefficients of the validation rules or the data are too different, you should consider scaling the data.

inspect_mip
inspect the mip problem formulation for one record

Description
Utility function to inspect the mip problem for a record.

Usage
inspect_mip(data, x, weight, ...)

Arguments
- data: data to be checked
- x: validation rules or errorlocalizer object to be used for finding possible errors.
- weight: numeric optional weight vector to be used in the error localization.
- ...: optional parameters that are passed to lpSolveAPI::lp.control() (see details)

Details
It may sometimes be handy to find out what is happening exactly with a record. See the example section for finding out what to do with inspect_mip.

See Also
Other Mixed Integer Problem: MipRules-class
Examples

```r
rules <- validator(x > 1)
data <- list(x = 0)
weight <- c(x = 1)

mip <- inspect_mip(data, rules)
print(mip)

# inspect the lp problem (prior to solving it with lpSolveAPI)
lp <- mip$to_lp()
print(lp)

# for large problems write the lp problem to disk for inspection
# lpSolveAPI::write.lp(lp, "my_problem.lp")

# solve the mip system / find a solution
res <- mip$execute()
names(res)

# lpSolveAPI status of finding a solution
res$s

# lp problem after solving (often simplified version of first lp)
res$lp

# records that are deemed "faulty"
res$errors

# values of variables used in the mip formulation. Also contains a valid solution
# for "faulty" variables
res$values

# see the derived mip rules and objective function, used in the construction of
# lp problem
mip$mip_rules()
mip$objective
```

---

**is_categorical**

**Check if rules are categorical**

Description

Check if rules are categorical

Usage

```r
is_categorical(x, ...)
```
Arguments

- `x`: validator or expression object
- `...`: not used

Details

```r
#' @note errorlocate supports linear, categorical and conditional rules to be used in finding errors. Other rule types are ignored during error finding.
```

Value

Logical indicating which rules are purely categorical/logical

See Also

Other rule type: `is_conditional()`, `is_linear()`

Examples

```r
v <- validator( A %in% c("a1", "a2")
                , B %in% c("b1", "b2")
                , if (A == "a1") B == "b1"
                , y > x
                )
is_categorical(v)
```

---

`is_conditional`  
**Check if rules are conditional rules**

Description

Check if rules are conditional rules

Usage

```
is_conditional(rules, ...)
```

Arguments

- `rules`: validator object containing validation rules
- `...`: not used

Value

Logical indicating which rules are conditional
is_linear

Note

erorlocate supports linear, categorical and conditional rules to be used in finding errors. Other rule types are ignored during error finding.

See Also

Other rule type: is_categorical(), is_linear()

Examples

```r
v <- validator( A %in% c("a1", "a2")
   , B %in% c("b1", "b2")
   , if (A == "a1") x > 1 # conditional
   , if (y > 0) x >= 0 # conditional
   , if (A == "a1") B == "b1" # categorical
   )

is_conditional(v)
```
locate_errors

Locate errors in data

Description

Locate erroneous fields in rows of data using validation rules or a specific errorlocalizer object. This method returns found errors, according to the specified method \( x \). Use method \texttt{replace_errors()} to automatically remove these errors.

Usage

\[
\text{locate}\_\text{errors}(\text{data}, \ x, \ \ldots, \ \text{cl} = \text{NULL}, \ \text{Ncpus} = \text{getOption("Ncpus", 1)}, \ \text{timeout} = 60)
\]

## S4 method for signature 'data.frame,validator'
\[
\text{locate}\_\text{errors}(\text{data}, \ x, \ \text{weight} = \text{NULL}, \ \text{ref} = \text{NULL}, \ \ldots, \ \text{cl} = \text{NULL}, \ \text{Ncpus} = \text{getOption("Ncpus", 1)}, \ \text{timeout} = 60)
\]

## S4 method for signature 'data.frame,ErrorLocalizer'
\[
\text{locate}\_\text{errors}(\text{data}, \ x, \ \text{weight} = \text{NULL}, \ \text{ref} = \text{NULL}, \ \ldots, \ \text{cl} = \text{NULL}, \ \text{Ncpus} = \text{getOption("Ncpus", 1)}, \ \text{timeout} = 60)
\]

Arguments

data data to be checked
locate_errors

x validation rules or errorlocalizer object to be used for finding possible errors.
...
optional parameters that are passed to lpSolveAPI::lp.control() (see details)
cl optional parallel / cluster.
Ncpus number of nodes to use. See details
timeout maximum number of seconds that the localizer should use per record.
weight numeric optional weight vector to be used in the error localization.
ref data.frame optional reference data to be used in the rules checking

Details

locate_errors uses lpSolveAPI to formulate and solve a mixed integer problem. This solver has many options: lpSolveAPI::lp.control.options. Noteworthy options to be used are:

- timeout: restricts the time the solver spends on a record (seconds)
- break.at.value: set this to minimum weight + 1 to improve speed.
- presolve: default for errorlocate is "rows". Set to "none" when you have solutions where all variables are deemed wrong.

locate_errors can be run on multiple cores using R package parallel.

- The easiest way to use the parallel option is to set Ncpus to the number of desired cores, @seealso parallel::detectCores()
- Alternatively one can create a cluster object (parallel::makeCluster()) and use cl to pass the cluster object.
- Or set cl to an integer which results in parallel::mclapply(), which only works on non-windows.

Value

errorlocation-class() object describing the errors found.

See Also

Other error finding: errorlocation-class, errors_removed(), replace_errors()

Examples

rules <- validator( profit + cost == turnover
   , cost >= 0.6 * turnover # cost should be at least 60% of turnover
   , turnover >= 0 # can not be negative.
)
data <- data.frame(profit=755, cost=125, turnover=200)
le <- locate_errors(data, rules)

print(le)
summary(le)
v_categorical <- validator( A %in% c("a1", "a2")
    , B %in% c("b1", "b2")
    , if (A == "a1") B == "b1"
)

data <- data.frame(A = c("a1", "a2"), B = c("b2", "b2"))
locate_errors(data, v_categorical)$errors

v_logical <- validator( A %in% c(TRUE, FALSE)
    , B %in% c(TRUE, FALSE)
    , if (A == TRUE) B == TRUE
)

data <- data.frame(A = TRUE, B = FALSE)
locate_errors(data, v_logical, weight=c(2,1))$errors

# try a conditional rule
v <- validator( married %in% c(TRUE, FALSE), if (married==TRUE) age >= 17 )
data <- data.frame( married = TRUE, age = 16)
locate_errors(data, v, weight=c(married=1, age=2))$errors

MipRules-class  Create a mip object from a validator object

Description

Create a mip object from validator() object. This is a utility class that translates a validator object into a mixed integer problem that can be solved. Most users should use locate_errors() which will handle all translation and execution automatically. This class is provided so users can implement or derive an alternative solution.

Methods

The MipRules class contains the following methods:

- $execute() calls the mip solver to execute the rules.
- $to_lp(): transforms the object into a lp_solve object
- $is_infeasible Checks if the current system of mixed integer rules is feasible.
- $set_values: set values and weights for variables (determines the objective function).

See Also

Other Mixed Integer Problem: inspect_mip()
Examples

```r
rules <- validator(x > 1)
mr <- miprules(rules)
mr$to_lp()
mr$set_values(c(x=0), weights=c(x=1))
mr$execute()
```

---

**replace_errors**

Replace erroneous fields with NA or a suggested value

**Description**

Find erroneous fields using `locate_errors()` and replace these fields automatically with NA or a suggestion that is provided by the error detection algorithm.

**Usage**

```r
replace_errors(
  data,
  x,
  ref = NULL,
  ...,
  cl = NULL,
  Ncpus =getOption("Ncpus", 1),
  value = c("NA", "suggestion")
)
```

## S4 method for signature 'data.frame,validator'

```r
replace_errors(
  data,
  x,
  ref = NULL,
  ...,
  cl = NULL,
  Ncpus =getOption("Ncpus", 1),
  value = c("NA", "suggestion")
)
```

## S4 method for signature 'data.frame,ErrorLocalizer'

```r
replace_errors(
  data,
  x,
  ref = NULL,
  ...,
  cl = NULL,
  Ncpus =getOption("Ncpus", 1),
  value = c("NA", "suggestion")
)
replace_errors

## S4 method for signature 'data.frame,errorlocation'
replace_errors(
  data,
  x,
  ref = NULL,
  ..., cl = NULL,
  Ncpus = 1,
  value = c("NA", "suggestion")
)

**Arguments**

- **data**: data to be checked
- **x**: validator() or errorlocation object. If an errorlocation is already available (through locate_errors()) this is more efficient.
- **ref**: optional reference data set
- **...**: these parameters are handed over to locate_errors()
- **cl**: optional cluster for parallel execution (see details)
- **Ncpus**: number of nodes to use. (see details)
- **value**: NA

**Details**

Note that you can also use the result of locate_errors() with replace_errors. When the procedure takes a long time and locate_errors was called previously this is the preferred way, because otherwise locate_errors will be executed again. The errors that were removed from the data.frame can be retrieved with the function errors_removed(). For more control over error localization see locate_errors().

replace_errors has the same parallelization options as locate_errors() (see there).

**Value**

data with erroneous values removed.

**Note**

In general it is better to replace the erroneous fields with NA and apply a proper imputation method. Suggested values from the error localization method may introduce an undesired bias.

**See Also**

errorlocation-class()

Other error finding: errorlocation-class.errors_removed(), locate_errors()
Examples

```r
rules <- validator( profit + cost == turnover,
                    cost - 0.6*turnover >= 0,
                    cost>= 0,
                    turnover >= 0)

data <- data.frame(profit=755, cost=125, turnover=200)

data_no_error <- replace_errors(data, rules)

# faulty data was replaced with NA

data_no_error

errors_removed(data_no_error)

# a bit more control, you can supply the result of locate_errors
# to replace_errors, which is a good thing, otherwise replace_errors will call
# locate_errors internally.

error_locations <- locate_errors(data, rules)

replace_errors(data, error_locations)
```

---

**translate_mip_lp**

*translate linear rules into an lp problem*

**Description**

translate linear rules into an lp problem

**Usage**

```r
translate_mip_lp(rules, objective = NULL, eps = 0.001, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rules</td>
<td>mip rules</td>
</tr>
<tr>
<td>objective</td>
<td>function</td>
</tr>
<tr>
<td>eps</td>
<td>accuracy for equality/inequality</td>
</tr>
<tr>
<td>...</td>
<td>additional <code>lp.control()</code> parameters that are set for the mip problem</td>
</tr>
</tbody>
</table>
Index

* Mixed Integer Problem
  inspect_mip, 6
  MipRules-class, 12

* error finding
  errorlocation-class, 4
  errors_removed, 5
  locate_errors, 10
  replace_errors, 13

* rule type
  is_categorical, 7
  is_conditional, 8
  is_linear, 9

add_noise, 3

create_errorlocation
  (errorlocation-class), 4

ErrorLocalizer-class, 4
errorlocate (errorlocate-package), 2
errorlocate(), 4
errorlocate-package, 2
errorlocation-class, 4
errors_removed, 4, 5, 11, 14
errors_removed(), 14

fh_localizer (FHLocalizer-class), 6
FHLocalizer-class, 6

inspect_mip, 6, 12
is_categorical, 7, 9
is_conditional, 8, 8, 9
is_linear, 8, 9, 9

locate_errors, 4, 5, 10, 14
locate_errors(), 6, 12–14
locate_errors.data.frame,ErrorLocalizer-method
  (locate_errors), 10
locate_errors.data.frame,validator-method
  (locate_errors), 10
lp.control(), 15

lpSolveAPI::lp.control(), 6, 11
lpSolveAPI::lp.control.options, 11

miprules (MipRules-class), 12
MipRules-class, 12

parallel::detectCores(), 11
parallel::makeCluster(), 11
parallel::mclapply(), 11

replace_errors, 4, 5, 11, 13
replace_errors(), 5, 10
replace_errors.data.frame,ErrorLocalizer-method
  (replace_errors), 13
replace_errors.data.frame,errorlocation-method
  (replace_errors), 13
replace_errors.data.frame,validator-method
  (replace_errors), 13

status, 4

translate_mip_lp, 15

validate(), 2
validator(), 9, 12, 14
values(), 4