Package 'errorlocate'

October 13, 2022

Type Package

Title Locate Errors with Validation Rules

Version 1.1

Description Errors in data can be located and removed using validation rules from package 'validate'. See also Van der Loo and De Jonge (2018) <doi:10.1002/9781118897126>, chapter 7.

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 7.2.0

Encoding UTF-8

Collate 'MipRules.R' 'addnoise.R' 'categorical.R' 'conditional.R' 'dnf.R' 'errorlocalizer.R' 'errorlocate-package.r' 'errorlocation.R' 'expand_weights.R' 'expr_manip.R' 'inspect_mip.R' 'linear.R' 'locate-errors.R' 'log_extract.R' 'mip_lpsolve.R' 'mip_rule.R' 'replace-errors.R' 'soft-rule.R' 'utils.R' 'values.R'

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 2022-06-29 11:20:04 UTC

R topics documented:

`errorlocate-package` *Find errors in data given a set of validation rules.*

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>

References

T. De Waal (2003) Processing of Erroneous and Unsafe Data. PhD thesis, University of Rotterdam.

Van der Loo, M., de Jonge, E, Data Cleaning With Applications in R

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

lp_solve and Kjell Konis. (2011). lpSolveAPI: R Interface for lp_solve version 5.5.2.0. R package 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 divided 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

- `$errors`: `matrix` indicating which values are erronuous (`TRUE`), missing (`NA`) or valid (`FALSE`)
- `$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()`, `expand_weights()`, `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, expand_weights(), 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)
```

`expand_weights` *Create a weight matrix*

Description

Expands a weight specification into a weight matrix to be used by `locate_errors` and `replace_errors`. Weights allow for "guiding" the errorlocalization process, so that less reliable values/variables with less weight are selected first. See details on the specification.

Usage

```
expand_weights(dat, weight = NULL, as.data.frame = FALSE, ...)
```

Arguments

`dat`	`data.frame` the data to be checked
`weight`	weight specification, see details.
`as.data.frame`	if `TRUE` a `data.frame` will be returned.
`...`	unused

Details

If weight fine tuning is needed, a possible scenario is to generate a weight `data.frame` using `expand_weights` and adjust it before executing `locate_errors()` or `replace_errors()`. The following specifications for `weight` are supported:

- `NULL`: generates a weight matrix with 1's
- a named `numeric`, unmentioned columns will have weight 1
- a unnamed `numeric` with a length equal to `ncol(dat)`
- a `data.frame` with same number of rows as `dat`
- a `matrix` with same number of rows as `dat`
- `Inf`, `NA` weights will be interpreted as that those variables must not be changed and are fixated. `Inf` weights perform much better than setting a weight to a large number.

Value

`matrix` or `data.frame` of same dimensions as `dat`

See Also

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

Examples

```
dat <- read.csv(text=
"age,country
  49,     NL
  23,     DE
", strip.white=TRUE)

weight <- c(age = 2, country = 1)
expand_weights(dat, weight)

weight <- c(2, 1)
expand_weights(dat, weight, as.data.frame = TRUE)

# works too
weight <- c(country=5)
expand_weights(dat, weight)

# specify a per row weight for country
weight <- data.frame(country=c(1,5))
expand_weights(dat, weight)

# country should not be changed!
weight <- c(country = Inf)
expand_weights(dat, weight)
```

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*

Description

Utility function to inspect the mip problem for a record. `inspect_mip` can be used as a "drop-in" replacement for `locate_errors()`, but works on the first 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 specification to be used in the error localization (see `expand_weights()`).
`...`	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 `vignette("inspect_mip")` for more details.

See Also

Other Mixed Integer Problem: `MipRules-class`

Examples

```
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

```
is_categorical(x, ...)
```

Arguments

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

Details

#' @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

```
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

Note

`errorlocate` 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

```
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)
```

is_linear *Check which rules are linear rules.*

Description

Check which rules are linear rules.

Usage

```
is_linear(x, ...)
```

Arguments

x	validator() object containing data validation rules
...	not used

Value

logical indicating which rules are (purely) linear.

Note

errorlocate 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_conditional()

locate_errors *Find errors in data*

Description

Find out which fields in a data.frame are "faulty" using validation rules This method returns found errors, according to the specified method x. Use method replace_errors(), to automatically remove these errors. ‘

Usage

```
locate_errors(
  data,
  x,
  ...,
  cl = NULL,
  Ncpus = getOption("Ncpus", 1),
  timeout = 60
)

## S4 method for signature 'data.frame,validator'
locate_errors(
  data,
  x,
  weight = NULL,
  ref = NULL,
  ...,
  cl = NULL,
  Ncpus = getOption("Ncpus", 1),
  timeout = 60
)

## S4 method for signature 'data.frame,ErrorLocalizer'
locate_errors(
  data,
  x,
  weight = NULL,
  ref = NULL,
  ...,
  cl = NULL,
  Ncpus = getOption("Ncpus", 1),
  timeout = 60
)
```

Arguments

data	data to be checked
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 specification to be used in the error localization (see expand_weights()).
ref	data.frame optional reference data to be used in the rules checking

Details

Use an `Inf` `weight` specification to fixate variables that can not be changed. See `expand_weights()` for more details.

`locate_errors` uses lpSolveAPI to formulate and solves a mixed integer problem. For details see the vignettes. 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()`, `expand_weights()`, `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( branch %in% c("government", "industry")
                          , tax %in% c("none", "VAT")
                          , if (tax == "VAT") branch == "industry"
)

data <- read.csv(text=
```

```
"   branch, tax
government, VAT
industry  , VAT
", strip.white = TRUE)
locate_errors(data, v_categorical)$errors

v_logical <- validator( citizen %in% c(TRUE, FALSE)
                      , voted %in% c(TRUE, FALSE)
                      ,  if (voted == TRUE) citizen == TRUE
                      )

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

# try a condinational 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


# different weights per row
data <- read.csv(text=
"married, age
    TRUE,  16
    TRUE,  14
", strip.white = TRUE)

weight <- read.csv(text=
"married, age
       1,   2
       2,   1
", strip.white = TRUE)

locate_errors(data, v, weight = weight)$errors

# fixate / exclude a variable from error localiziation
# using an Inf weight
weight <- c(age = Inf)
locate_errors(data, v, weight = weight)$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 validor 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

```
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

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

## S4 method for signature 'data.frame,validator'
replace_errors(
  data,
  x,
  ref = NULL,
  ...,
  cl = NULL,
```

```
  Ncpus = getOption("Ncpus", 1),
  value = c("NA", "suggestion")
)

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

## 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()`, `expand_weights()`, `locate_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)
```

---

translate_mip_lp *translate linear rules into an lp problem*

Description

translate linear rules into an lp problem

Usage

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

Arguments

`rules`	mip rules
`objective`	function
`eps`	accuracy for equality/inequality
`...`	additional `lp.control()` parameters that are set for the mip problem

Index