Package ‘migrbc’

May 15, 2020

Type Package

Title Production Rules Based Classification of Migration

Version 2.0.9

URL https://github.com/statisticsnz/migrbc,
https://statisticsnz.github.io/migrbc

BugReports https://github.com/statisticsnz/migrbc/issues

Language en-US

Author Leshi Chen [aut, cre],
Pubudu Senanayake [aut],
Del Robinson [aut],
Statistics New Zealand [cph]

Description Provides mechanisms for classifying border crossings using a rules-based methodology. The goal of performing this type of classification is to identify any potential long-term migrants. A long-term migration is defined as a border crossing involving a change in residence status. A border crossing counts as a long-term migration to/from a country if it entails a change from non-residence to residence / residence to non-residence. The rules-based classification that used to determine a long-term migration is defined by a threshold duration and a test duration, alternatively named window size. Under a 12/16 rule, for instance, the threshold duration is 12 months and the test duration (window size) is 16 months. With a 9/12 rule, the threshold duration is 9 months and the test duration (window size) is 12 months. For more information about the methodology applied, please visit Stats NZ (2020) <https://www.stats.govt.nz/methods/defining-migrants-using-travel-histories-and-the-1216-month-rule>.

License MIT + file LICENSE

LinkingTo Rcpp

Encoding UTF-8

LazyData true

NeedsCompilation yes

Depends R (>= 3.5)

Imports Rcpp (>= 1.0), lubridate (>= 1.7), stringr (>= 1.4), dplyr (>= 0.8), methods, parallel, futile.logger

Suggests magrittr, tools, knitr, rmarkdown, testthat

RoxygenNote 6.1.1

Collate 'RcppExports.R' 'check_functions.R' 'plot_mig_hist.R'
'pre_process.R' 'resolve_data.R' 'resolve_data_with_error.R'
'run_rbc.R' 'utility_functions.R' 'migrbc.R'

VignetteBuilder knitr

Maintainer Leshi Chen <leshi.chen@stats.govt.nz>

Repository CRAN

Date/Publication 2020-05-15 15:05:51 UTC

R topics documented:

`check_and_tidy_date` *Validate general dates*

Description

A function to check the date variable whether is the right date. This is an internal function.

Usage

```
check_and_tidy_date(date, date_name)
```

Arguments

`date`	A date object in string format such as '2018-01-01'.
`date_name`	The name of the date variable.

Value

A verified date object in string format

`check_and_tidy_date_crossing` *Validate dates on border crossing.*

Description

A function to check the date variable whether is the right date. This is an internal function.

Usage

```
check_and_tidy_date_crossing(date_crossing)
```

Arguments

`date_crossing`	The border crossing date.

Value

The border crossing date that has been verified and tidied up.

`check_and_tidy_date_first_last` *Validate dates in sequence*

Description

A function to check the date variable whether is the right date. This is an internal function.

Usage

```
check_and_tidy_date_first_last(date, date_crossing, name)
```

Arguments

`date`	The last date to compare with.
`date_crossing`	The border crossing date.
`name`	A name of the checking variable.

Value

The date value that has been verified and reformatted correctly.

`check_data_columns` *Validate the data columns of crossing information*

Description

A function to check the data variable whether contains the right columns of crossing information.

Usage

```
check_data_columns(data)
```

Arguments

`data`	The journey data that should contain columns in the set of 'journeyId', 'personId', 'date_crossing', 'is_arrival', 'journey_sequence', and 'journeyId_prev'.

Value

A NULL value if there is no issue raised.

`check_ini_res_data_columns` *Validate the data columns of the initial residence status data*

Description

A function to check the data variable whether contains the right columns of crossing information.

Usage

```
check_ini_res_data_columns(data)
```

Arguments

`data`	The journey data that should contain columns in the set of 'personId', 'res_status_initial', and 'date_finalised'.

Value

A NULL value if there is no issue raised.

`check_integer` *Validate an integer value*

Description

A function to check the variable whether is the right integer type. This is an internal function.

Usage

```
check_integer(name = NULL, value = NULL)
```

Arguments

`name`	The name of the variable.
`value`	The validating variable.

Value

A NULL value if there is no issue raised.

`check_is_logic` *Validate a logical value*

Description

A function to check the variable whether is the right logic type. This is an internal function.

Usage

```
check_is_logic(check_value)
```

Arguments

`check_value`	Boolean value to present In/Out the country.

Value

A NULL value if there is no issue raised.

`check_object_size` *Validate the size of a object*

Description

A function to check the size of a data variable whether is in the right range.

Usage

```
check_object_size(object, max_ram = 2, target_unit = "Gb")
```

Arguments

`object`	An object that is required to check.
`max_ram`	The maximum size of the target object.
`target_unit`	The target unit that is constrained. The value is one of c('bytes', 'Kb', 'Mb', 'Gb', 'Tb', 'Pb').

Value

A NULL value if there is no issue raised.

`check_positive_number` *Validate a positive numeric value*

Description

A function to check the variable whether is positive number. This is an internal function.

Usage

```
check_positive_number(number, name)
```

Arguments

`number`	The checking value.
`name`	The name of the variable.

Value

A NULL value if there is no issue raised.

`check_work_spaces` *Validate the size of data (work space)*

Description

A function to check the size of a data variable whether is in the right range.

Usage

```
check_work_spaces(pre_processed_data, max_ram = 2, target_unit = "Gb")
```

Arguments

`pre_processed_data`	Data that processed by the function `pre_process`.
`max_ram`	A value of the maximum size of the list of CrossingWorkSpace instance.
`target_unit`	The target unit, i.e., ’Gb’, ’Tb’ and ’Pb’. The default value is ’Gb’.

Value

A NULL value if there is no issue raised.

`get_object_size` *Get Object Size*

Description

A function to get the size of an object

Usage

```
get_object_size(object)
```

Arguments

`object`	The target object.

Value

A named list object that contains information on the size of an object and the size unit.

Examples

```
res <- get_object_size(TRUE)
res$size
res$unit
```

`get_random_dates` *Get Random Dates*

Description

An internal function to create test data

This function is used to generate random dates for `setup_random_test_data`

Usage

```
get_random_dates(start_date, num_of_dates = 1000, min = 0, max = 100,
  seed = NULL)
```

Arguments

`start_date`	The start crossing date.
`num_of_dates`	The number of journeys for each person.
`min`	The minimum duration between journeys.
`max`	The maximum duration between journeys.
`seed`	A random seed to generate random dates.

Value

A list of boarder crossing dates

`initialize_logger` *Initialize Futile Logger*

Description

This function is used to initialize the futile.logger so that the user can be notified with the current status of running RBC.

Usage

```
initialize_logger(log_level = 6, log_path = NULL)
```

Arguments

`log_level`	a parameter representing a threshold, which affects the visibility of a given logger. If the log level is at or higher in priority than the logger threshold, a message will print. Otherwise the command will silently return. The value of the log_level is a number between 1 and 9. 9 or futile.logger::TRACE will show all messages in details.
`log_path`	A path for the output log files generated by the logger. If NULL, all messages will be displayed in the calling environment.

Value

it runs on side effects but also return a simple message.

Examples

```
## futile.logger::FATAL: 1
## futile.logger::ERROR: 2
## futile.logger::WARN:  4
## futile.logger::INFO:  6
## futile.logger::DEBUG: 8
## futile.logger::TRACE: 9

## to suppresse log messages to the console
migrbc::initialize_logger(log_level = 1)

## to display all messages to the console
migrbc::initialize_logger(log_level = 9)
```

internal_process *Internal function*

Description

Internal function

Usage

```
internal_process(subgroup, window_size, threshold_year)
```

Arguments

subgroup	A subgroup of the pre-processed data groups, generated by `migrbc::pre_process`.
window_size	The maximum length of the scanning period. Can be an integer giving the number of days, the result of a call to function `difftime`, or an object of class `Duration`.
threshold_year	The length of the yearly test period. Can be an integer giving the number of days, the result of a call to function `difftime`, or an object of class `Duration`.

Value

A data frame object of classified / labelled journeys.

migrbc *A package for classifying border crossings using a rules-based methodology.*

Description

The migrbc package provides three categories of important functions: `run_rbc`, `pre_process`, `plot_mig_hist`, `resolve_data`, `initialize_logger`, and `resolve_data_with_error`. In among of the five functions, `run_rbc` is the main entry function of the package. Three functions: `initialize_logger`, `pre_process` and `plot_mig_hit` are utility functions. The rest functions `resolve_data` and `resolve_data_with_error` are the key functions to do the rules based classification.

`initialize_logger` function

This function is used to initialize the futile.logger so that the user can be notified with the current status of running RBC.

`run_rbc` function

The run_rbc function attempt to determine long-term migration statuses, and pre-crossing and post-crossing residence statuses, for all crossings where these statuses are not known.

`pre_process` function

This function provides a mechanism to divide large data into small chunks.

`plot_mig_hist` function

Given a sequence of border crossings for a person, draw a diagram describing that person's migration history

`resolve_data` function

This function is the key function to do the rules based classification.

`resolve_data_with_error` function

This function is used to produce error result.

`plot_mig_hist` *Plot a migration history.*

Description

Given a sequence of border crossings for a person, draw a diagram describing that person's migration history.

Note that, unlike elsewhere in package `migrbc`, the `date_crossing` and `is_arrival` arguments for `plot_mig_hist` refer to a single individual.

If values for `date_first` and `date_last` are not supplied, then defaults are calculated, based on the length of the travel history.

Usage

```
plot_mig_hist(date_crossing, is_arrival, days_to_next_crossing,
  res_status_before_str = NULL, res_status_after_str = NULL,
  date_first = NULL, date_last = NULL, show_dates = TRUE,
  show_days = TRUE, cex = 1, lwd = 1)
```

Arguments

`date_crossing`	A vector of dates.
`is_arrival`	A logical vector, the same length as `date_crossing` specifying whether each border crossing is an arrival.
`days_to_next_crossing`	A number vector, the same length as `date_crossing` specifying the days span between two journeys.
`res_status_before_str`	Character or numeric vector, the same length as `date_crossing`, showing residence status before each crossing. Optional.

`res_status_after_str`	Character or numeric vector, the same length as `date_crossing`, showing residence status after each crossing.
`date_first`	The start date for the travel history. Optional.
`date_last`	The end date for the travel history. Optional.
`show_dates`	Logical. Whether to display the dates of each border crossing.
`show_days`	Logical. Whether to display the length, in days, of each spell in or out of the country.
`cex`	'Character expansion factor'. A number. Larger values lead to larger text. Defaults to 1.
`lwd`	Line width. A number. Larger values lead to thicker lines. Defaults to 1.

Value

Returns `NULL`, but as a side effect draws a graph (using R's traditional graphics system).

Examples

```
## to suppresse log messages to the console
migrbc::initialize_logger(log_level = 1)

plot_test <- function(mig_hist) {
  plot_mig_hist(date_crossing = as.character(mig_hist$date_crossing),
                is_arrival = mig_hist$is_arrival,
                days_to_next_crossing = mig_hist$days_to_next_crossing,
                show_date = FALSE,
                cex = 0.8)
}
number_of_people = 1
person_data <- migrbc::setup_random_test_data(number_of_people,
                                              initial_date = '2001-01-01',
                                              numJourneys = 3,
                                              min = 0,
                                             max = 100)
cross_spaces <- migrbc::pre_process(person_data, n_groups = 1)
## run in non-parallel
post_data <- migrbc::run_rbc(cross_spaces,
                       window_size = 487,
                       threshold_year = 365,
                       parallel=FALSE)
old_par <- par(mfrow = c(1, 1))
plot_test(post_data$journeys)
par(old_par)
```

`pre_process` *A function to convert a large data into a number of sub groups*

Description

This function provides a mechanism to divide large data into small chunks.

Usage

```
pre_process(data, init_res_status_data = NULL, n_groups = 1)
```

Arguments

`data`	A dataframe object.
`init_res_status_data`	The raw data of the initial residence status in the format of data frame.
`n_groups`	The number of groups required to be returned.

Value

A list object contains reformatted raw data.

Examples

```
## to suppresse log messages to the console
migrbc::initialize_logger(log_level = 1)

number_of_people = 10
person_data <- migrbc::setup_random_test_data(number_of_people,
                                              initial_date = '2001-01-01',
                                              numJourneys = 5,
                                              min = 0,
                                              max = 10)
crossings <- migrbc::pre_process(person_data, n_groups = 10)
crossings
```

`rcpp_resolve` *Processing RBC for a person.*

Description

This function is used to resolve one person's journeys, i.e., classifying a person and marking it whether or not to be a long term migrant based on the person's journeys. This function is used internally inside the package and shouldn't be exposed to the outside caller.

Usage

```
rcpp_resolve(person_data, int_res_status, initial_date_finalised, tw, tm)
```

Arguments

`person_data`	A list object of a person's journeys.
`int_res_status`	The initial residence status of the target person
`initial_date_finalised`	The final resolved date of the initial residence status.
`tw`	Windows Size, by default, it is 487 days.
`tm`	Threshold of Year, by default, it is 365 days.

Value

A list of classified / labelled journeys.

`resolve_data` *Process RBC*

Description

This function is the key function to do the rules based classification.

Usage

```
resolve_data(grouped_data, window_size = 487, threshold_year = 365,
  parallel = FALSE, n_core = 2, include_error_columns = FALSE,
  mc.cleanup = FALSE)
```

Arguments

`grouped_data`	A list of data frame objects.
`window_size`	The maximum length of the scanning period. Can be an integer giving the number of days, the result of a call to function `difftime`, or an object of class `Duration`.
`threshold_year`	The length of the yearly test period. Can be an integer giving the number of days, the result of a call to function `difftime`, or an object of class `Duration`.
`parallel`	Optional, if it is TRUE, run on parallel.
`n_core`	if `parallel` set to TRUE, this will specify the number of computer cores required.
`include_error_columns`	Optional, if it is TRUE, the returned result of `error_data` will contain two extra columns `error_code` and `error_message`.

`mc.cleanup`	if set to TRUE then all children that have been forked by this function will be killed (by sending SIGTERM) before this function returns. Under normal circumstances mclapply waits for the children to deliver results, so this option usually has only effect when mclapply is interrupted. If set to FALSE then child processes are collected, but not forcefully terminated. As a special case this argument can be set to the number of the signal that should be used to kill the children instead of SIGTERM.

Value

A list type of object that contains a classified journey dataframe object and a error dataframe object.

Examples

```
## to suppresse log messages to the console
migrbc::initialize_logger(log_level = 1)

number_of_people = 10
person_data <- migrbc::setup_random_test_data(number_of_people,
                                 initial_date = '2001-01-01',
                                 numJourneys = 5,
                                 min = 0,
                                 max = 10)
crossings <- migrbc::pre_process(person_data, n_groups = 10)
crossings
cross_spaces <- migrbc::resolve_data(crossings)
cross_spaces
```

`resolve_data_with_error` *Produce Error Result*

Description

This function is used to produce error result.

Usage

```
resolve_data_with_error(data_with_error, initial_res_status_data,
  error_message = "", include_error_columns = FALSE,
  window_size = 487)
```

Arguments

`data_with_error`	The personal crossing data for RBC process with error.
`initial_res_status_data`	the initial residence status data.

`error_message`	The error message.
`include_error_columns`	Optional, if it is TRUE, the returned result of `error_data` will contain two extra columns `error_code` and `error_message`.
`window_size`	The maximum length of the scanning period. Can be an integer giving the number of days, the result of a call to function `difftime`, or an object of class `Duration`.

Value

A dataframe type of object contains journeys with error.

Examples

```
## to suppresse log messages to the console
migrbc::initialize_logger(log_level = 1)

j1 <-       c(journeyId = 1,
              personId = 1,
              is_arrival = 1,
              date_crossing = '2017-01-01',
              journey_sequence = 1,
              journeyId_prev = NA)

j2 <-       c(journeyId = 2,
              personId = 1,
              is_arrival = 1,
              date_crossing = '2018-01-06',
              journey_sequence = 2,
              journeyId_prev = 1)

j3 <-       c(journeyId = 3,
              personId = 1,
              is_arrival = 1,
              date_crossing = '2018-01-16',
              journey_sequence = 3,
              journeyId_prev = 2)


j4 <-       c(journeyId = 4,
              personId = 2,
              is_arrival = 0,
              date_crossing = '2017-01-01',
              journey_sequence = 1,
              journeyId_prev = NA)

j5 <-       c(journeyId = 5,
              personId = 2,
              is_arrival = 0,
              date_crossing = '2018-01-06',
              journey_sequence = 2,
              journeyId_prev = 4)

j6 <-       c(journeyId = 6,
              personId = 2,
              is_arrival = 0,
              date_crossing = '2018-01-16',
              journey_sequence = 3,
              journeyId_prev = 5)

person_data <- as.data.frame(rbind(j1, j2, j3, j4, j5, j6),
                             stringsAsFactors = FALSE)
i1 <- c(personId = 1,
        res_status_initial = 1,
        date_finalised = '2017-01-01')
ini_data <- as.data.frame(t(i1), stringsAsFactors = FALSE)

person_data$journeyId <- as.numeric(person_data$journeyId)
person_data$personId <- as.numeric(person_data$personId)
person_data$is_arrival <- as.numeric(person_data$is_arrival)
person_data$journey_sequence <-
  as.numeric(person_data$journey_sequence)
person_data$journeyId_prev <-
  as.numeric(person_data$journeyId_prev)

ini_data$personId <- as.numeric(ini_data$personId)
ini_data$res_status_initial <-
  as.numeric(ini_data$res_status_initial)
ini_data$date_finalised <-
  as.character(ini_data$date_finalised)

res <- migrbc::resolve_data_with_error(person_data,
                                        initial_res_status_data = ini_data,
                                        error_message = 'custom error',
                                        include_error_columns = TRUE)
head(res)
```

---

run_rbc *Run RBC*

Description

A function that attempts to determine long-term migration statuses, and pre-crossing and post-crossing residence statuses, for all border crossings where these statuses are not known.

Usage

```
run_rbc(crossing_data, init_res_status_data = NULL, window_size = 487,
  threshold_year = 365, parallel = FALSE, n_core = 2, max_ram = 2,
  include_error_columns = FALSE, mc.cleanup = FALSE)
```

Arguments

`crossing_data`	A pre-processed group data contain journeys, movements and other raw crossing data. The data should contain columns in the set of 'journeyId', 'personId', 'date_crossing', 'is_arrival', 'journey_sequence', and 'journeyId_prev'.
`init_res_status_data`	Optional, the raw data of the initial residence status in the format of data frame. The journey data should contain columns in the set of 'personId', 'res_status_initial', and 'date_finalised' if applied. The initial data is a supplementary to the `crossing_data` that provides the initial residence status of the target people who made the border crossing (journey).
`window_size`	The maximum length of the scanning period. Can be an integer giving the number of days, the result of a call to function `difftime`, or an object of class `Duration`.
`threshold_year`	The length of the yearly test period. It can be an integer giving the number of days, the result of a call to function `difftime`, or an object of class `Duration`.
`parallel`	Logical. Whether to use parallel processing, to speed up the calculation of migration statuses. Defaults to `TRUE`.
`n_core`	The number of cores to use, if `parallel` is `TRUE`. Defaults to `2`. Higher values will typically result in faster calculations on computers with more than two cores.
`max_ram`	Optional, it is used to limit the RAM that can be used by this function. The default value is 2 Gb.
`include_error_columns`	Optional, if it is TRUE, the returned result of `error_data` will contain two extra columns `error_code` and `error_message`.
`mc.cleanup`	Optional, if set to TRUE then all children that have been forked by this function will be killed (by sending SIGTERM) before this function returns. Under normal circumstances mclapply waits for the children to deliver results, so this option usually has only effect when mclapply is interrupted. If set to FALSE then child processes are collected, but not forcefully terminated. As a special case this argument can be set to the number of the signal that should be used to kill the children instead of SIGTERM.

Value

A list type of object that contains two items: one is a data frame object that contains classified journeys and the other contains journeys that have been marked as error. Both items contain the same table structure in the set of 'journeyId', 'journeyId_prev', 'personId', 'date_crossing', 'is_arrival', 'journey_sequence','days_to_next_crossing', 'res_status_before', 'res_status_after', 'is_long_term_mig', 'date_finalised_res_before', 'date_finalised_res_after' and 'date_finalised_LTM'. The Boolean value (0, and 1) in the column 'is_long_term_mig' is the key classified result that tells us which journey derived the person to be a long term migrant.

Examples

```
## generate test data 100 people and each person has
## 10 journeys

## to suppresse log messages on the screen
migrbc::initialize_logger(log_level = 1)

number_of_people <- 100
person_data <- migrbc::setup_random_test_data(
    number_of_people,
    initial_date = '2001-01-01',
    numJourneys = 10,
    min = 0,
    max = 100)
head(person_data)

cross_spaces <- migrbc::pre_process(person_data)

## run in non-parallel
res <- migrbc::run_rbc(cross_spaces,
                       window_size = 487,
                       threshold_year = 365,
                       parallel=FALSE)

## run in parallel with n_core = 2
cross_spaces <- migrbc::pre_process(person_data, n_groups = 2)
res <- migrbc::run_rbc(cross_spaces,
                       window_size = 487,
                       threshold_year = 365,
                       parallel=TRUE,
                       n_core = 2)

head(res$journeys)
head(res$error_data)
```

run_rbc_process_core *Processing RBC for a list of person.*

Description

This function is used to resolve a list of person's journeys, i.e., classifying a list of people and marking it whether or not to be a long term migrant based on the person's journeys. This function is used internally inside the package and shouldn't be exposed to the outside caller.

Usage

```
run_rbc_process_core(cross_data, ini_status_data, tw, ty)
```

Arguments

`cross_data`	The personal crossing data for RBC process
`ini_status_data`	the initial residence status data
`tw`	Windows Size, by default, it is 487 days.
`ty`	Threshold of Year, by default, it is 365 days.

Value

A data frame object of classified / labelled journeys

`run_rbc_process_with_error` *Processing RBC for a list of person.*

Description

This function is used to resolve a list of person's journeys with error, This function is used internally inside the package and shouldn't be exposed to the outside caller.

Usage

```
run_rbc_process_with_error(cross_data, ini_status_data, error_message, tw)
```

Arguments

`cross_data`	The personal crossing data for RBC process
`ini_status_data`	the initial residence status data
`error_message`	The error message.
`tw`	Windows Size, by default, it is 487 days.

Value

A data frame object of classified / labelled journeys

`segment_coord_horiz` *Internal function*

Description

Internal function

Usage

```
segment_coord_horiz(date_crossing, is_arrival, date_first, date_last)
```

Arguments

`date_crossing`	date of border crossing.
`is_arrival`	A Boolean value.
`date_first`	The first date occurred.
`date_last`	The last date occurred.

Value

A list object that contains values for coordinates of horizon.

`segment_coord_vert` *Internal function*

Description

Internal function

Usage

```
segment_coord_vert(date_crossing, is_arrival)
```

Arguments

`date_crossing`	date of border crossing.
`is_arrival`	A Boolean value.

Value

A list object that contains values for coordinates of vertical.

setup_random_test_data

Setup Random Test Data

Description

A function to generate test data for RBC for toy examples.

Usage

```
setup_random_test_data(num_people = 10, initial_date = "2001-01-01",
  numJourneys = 5, min = 0, max = 10)
```

Arguments

num_people	The number of person instances.
initial_date	The start crossing date.
numJourneys	The number of journeys for each person.
min	The minimum duration between journeys.
max	The maximum duration between journeys.

Value

A data frame object

Examples

```
res <- setup_random_test_data(10,
                              initial_date = '2001-01-01',
                              numJourneys = 5,
                              min = 0,
                              max = 10)
head(res)
```

Index