Package 'stopdetection'

April 18, 2023

Title Stop Detection in Timestamped Trajectory Data using Spatiotemporal Clustering

Version 0.1.2

Description Trajectory data formed by human or animal movement is often marked by periods of movement interspersed with periods of standing still. It is often of interest to researchers to separate geolocation trajectories of latitude/longitude points by clustering consecutive locations to produce a model of this behavior. This package implements the Stay Point detection algorithm originally described in Ye (2009) <doi:10.1109/MDM.2009.11> that uses time and distance thresholds to characterize spatial regions as 'stops'. This package also implements the concept of merging described in Montoliu (2013) <doi:10.1007/s11042-011-0982-z> as stay point region estimation, which allows for clustering of temporally adjacent stops for which distance between the midpoints is less than the provided threshold. GPS-like data from various sources can be used, but the temporal thresholds must be considered with respect to the sampling interval, and the spatial thresholds must be considered with respect to the measurement error.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.1

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports data.table, geodist, lubridate, stats

Depends R (>= 2.10)

LazyData true

VignetteBuilder knitr

URL https://github.com/daniellemccool/stopdetection

BugReports https://github.com/daniellemccool/stopdetection/issues

NeedsCompilation no

Author McCool Danielle [aut, cre] (<https://orcid.org/0000-0002-7055-7539>)

Maintainer McCool Danielle <d.m.mccool@uu.nl>

Repository CRAN

Date/Publication 2023-04-18 11:00:02 UTC

R topics documented:

`loc_data_2019` *Timestamped Location Data*

Description

Real data from November 2019 extracted from Google Location History files captured with an Android smartphone. Contains two weeks of human movement behavior of a single person occurring in the Netherlands. Modes include biking, walking, bus and train.

Usage

```
loc_data_2019
```

Format

`loc_data_2019`**:**

A data frame with 21,911 rows and 3 columns:

latitude unprojected latitude coordinate using WGS84 ellipsoid

longitude unprojected longitude coordinate using WGS84 ellipsoid

timestamp POSIXct timestamp with date and time using fractional seconds ...

Source

Personal recorded location history

mergingCycle *Merging Cycle*

Description

Runs the stop and merging cycle until no changes are seen or until the max number of merges are met.

Usage

```
mergingCycle(
  res,
  max_merges = Inf,
  thetaD = 200,
  small_track_action = "merge",
  ...
)
```

Arguments

res	Results data.table from stopFinder containing columns timestamp, longitude, latitude and state
max_merges	integer of maximum number of merges to perform
thetaD	how many meters away may stops be and still be merged
small_track_action	one of "merge" or "exclude" for short tracks
...	additional optional arguments passed to moveMerger including max_locs, max_time and max_dist

Value

Modifies res data.table by reference

Examples

```
# Load data
library(data.table)
data(loc_data_2019); setDT(loc_data_2019)
# Find initial set of stops
stopFinder(loc_data_2019, thetaD = 200, thetaT = 300)
# This selection contains two short tracks to eliminate and two stops to merge
example <- copy(loc_data_2019[state_id %between% c(1, 11)])
events_pre_merge <- returnStateEvents(example)
# Perform the merging
mergingCycle(example, thetaD = 200, small_track_action = "exclude", max_locs = Inf)
events_post_merge <- returnStateEvents(example)
# From 11 states to 8 states
```

```
events_pre_merge[, .(state_id, state, meanlat, meanlon, n_locations)]
events_post_merge[, .(state_id, state, meanlat, meanlon, n_locations)]
```

`moveMerger` *Move Merger*

Description

Handles move/track events that do not meet specific thresholds to be considered. This is based on the researcher-decided total number of allowable locations that the discarded track can consist of, as well as a maximum total time length that may elapse. Tracks can be merged into the preceding stop or excluded. Future versions of this should consider assigning to the closest stop for `small_track_action = merge`.

Usage

```
moveMerger(
  events,
  small_track_action = "merge",
  max_locs = 1,
  max_time = 600,
  max_dist = 100
)
```

Arguments

`events`	data.table of events from `returnStateEvents`
`small_track_action`	One of "merge" or "exclude" for specifying the method of handling mergeable tracks
`max_locs`	Maximum number of locations for a track to be mergeable. Set to Inf to not consider.
`max_time`	Maximum time elapsed (seconds) for a track to be mergeable. Set to Inf to not consider.
`max_dist`	Maximum distance (meters) traveled while on track to be mergeable. Set to Inf to not consider.

Value

Modifies events data.table by reference

`radiusOfGyrationDT` *Radius of Gyration*

Description

Calculates the time-weighted radius of Gyration provided a data.table containing latitude, longitude and a timestamp. This is the root-mean-square time-weighted average of all locations. Weighting by time is provided to adjust for unequal frequency of data collection.

Usage

```
radiusOfGyrationDT(lat_col, lon_col, timestamp, dist_measure = "geodesic")
```

Arguments

`lat_col`	Time-ordered vector of latitudes
`lon_col`	Time-ordered vector of longitudes
`timestamp`	Timestamps associated with the latitude/longitude pairs
`dist_measure`	Passed through to geodist::geodist_vec, One of "haversine" "vincenty", "geodesic", or "cheap" specifying desired method of geodesic distance calculation.

Details

Time-weighted RoG is defined as

$$\sqrt{\frac{\sum_i w_j \times dist([\overline{lon}, \overline{lat}], [lon_j, lat_j])}{\sum_i w_j}}$$

Where

$$\overline{lon} = \frac{\sum_j w_j lon_j}{\sum_j w_j} \quad \text{and} \quad \overline{lat} = \frac{\sum_j w_j lat_j}{\sum_j w_j}$$

And the weighting element w_j represents half the time interval during which a location was recorded

$$w_j = \frac{t_{j+1} - t_{j-1}}{2}$$

Value

Time-weighted radius of gyration

Examples

```
# Inside a data.table
dt <- data.table::data.table(
  lat = c(1, 1, 1, 1, 1),
  lon = c(1, 1.5, 4, 1.5, 2),
  timestamp = c(100, 200, 300, 600, 900)
```

```
)
dt[, radiusOfGyrationDT(lat, lon, timestamp)]
# As vectors
radiusOfGyrationDT(
  c(1, 1, 1, 1, 1),
  c(1, 1.5, 4, 1.5, 2),
  c(100, 200, 300, 600, 900)
  )
```

`returnStateEvents` *Return State Events*

Description

Given a data.table updated with stop and move events from `stopFinder`, returns data aggregated to the event level.

Usage

```
returnStateEvents(dt)
```

Arguments

`dt`	data.table updated with stop and move events from `stopFinder`

Value

data.table with one line per stop/move event, annotated with columns state_id, state, begin_time, end_time and n_locations. Move events contain information on the raw_travel_dist and a move_id. Stop events have values for columns meanlat and meanlon, which are respectively the mean latitude and longitude of locations occurring during the stop.

Examples

```
library(data.table)
data(loc_data_2019); setDT(loc_data_2019)
stopFinder(loc_data_2019, thetaD = 200, thetaT = 300)
returnStateEvents(loc_data_2019)
```

stopFinder *Find an initial set of stops given timestamped locations*

Description

stopFinder modifies by reference a data.table of trajectories, which are clustered spatiotemporally based on a user-provided distance radius parameter and time parameter. Points are evaluated sequentially to determine whether they meet the criteria for being a stop (at least thetaT time spent within thetaD distance of the initiating location). Points must therefore have a timestamp, longitude and latitude column.

Usage

```
stopFinder(traj, thetaD, thetaT)
```

Arguments

traj	An ordered data.table with columns named timestamp, longitude and latitude
thetaD	The distance parameter, represents a radius in meters for establishing how much area a stop can encompass.
thetaT	The time parameter, representing the length of time that must be spent within the stop area before being considered a stop.

Details

This function has been optimized for simulation studies where it will be called repeatedly. Because of this, all error-handling is done prior to this step. If calling this function directly, the user must ensure that the data are ordered based on the timestamp, and that the columns names are correct.

Value

traj is modified by reference to include a column stop_initiation_idx, which is NA for locations not belonging to a stop, and equal to the row number initiating the stop it belongs to otherwise.

Examples

```
# Set up data
library(data.table)
dt <- data.table(entity_id = rep(1, 27),
timestamp = c(1, 2, 4, 10, 14, 18, 20, 21, 24, 25, 28, 29, 45, 80, 100,
              120, 200, 270, 300, 340, 380, 450, 455, 460, 470, 475,
              490),
longitude = c(5.1299311, 5.129979, 5.129597, 5.130028, 5.130555, 5.131083,
              5.132101, 5.132704, 5.133326, 5.133904, 5.134746, 5.135613,
              5.135613, 5.135613, 5.135613, 5.135613, 5.135613, 5.135613,
              5.135613, 5.135613, 5.135613, 5.135613, 5.134746, 5.133904,
              5.133326, 5.132704, 5.132101),
latitude = c(52.092839, 52.092827, 52.092571, 52.092292, 52.092076, 52.091821,
```

```
              52.091420, 52.091219, 52.091343, 52.091651, 52.092138, 52.092698,
              52.092698, 52.092698, 52.092698, 52.092698, 52.092698, 52.092698,
              52.092698, 52.092698, 52.092698, 52.092138, 52.091651, 52.091343,
              52.091219, 52.091420, 52.091821))
stopFinder(dt, thetaD = 50, thetaT = 400)[]
plot(dt$longitude, dt$latitude, type = "b", lwd = dt$timedif, pch = 20,
 main = "Stay point detection from timestamped trajectory",
 sub = "Point size is elapsed time, points in red form a stop")
points(x = dt$longitude[dt$state == "stopped"],
 y = dt$latitude[dt$state == "stopped"],
 col = "red", lwd = dt$timedif[dt$state == "stopped"], pch = 20)
```

`stopMerger` *Stop Merger*

Description

Given the events data.table containing the spatiotemporally clustered stop/ move states, merges stops separated by less than `thetaD` meters. Modifies events by reference.

Usage

```
stopMerger(events, thetaD)
```

Arguments

`events`	data.table of events from `returnStateEvents`
`thetaD`	maximum distance for merging subsequent stops

Value

modifies events data.table by reference, changing new_stop_id and new_state

Index