Package 'spatialrisk'

September 1, 2026

Type Package

Title Spatial Concentration and Radius-Based Risk Calculations

Version 0.8.2

Author Martin Haringa [aut, cre]

Maintainer Martin Haringa <mtharinga@gmail.com>

BugReports https://github.com/mharinga/spatialrisk/issues

Description Provides computational building blocks for fixed-radius spatial aggregation, weighted circle-placement problems, hotspot detection, and polygon-based spatial summaries. The package focuses on efficient determination of the sum of observations within a given radius, identifying areas of high local concentration, and aggregating point data to polygon geometries. These methods are useful for applications such as insurance, urban analytics, environmental exposure analysis, and other spatial point pattern workflows. The fixed-radius circle placement problem is discussed by Chazelle and Lee (1986) <doi:10.1007/BF02238188>, and related maximum covering problems are described by Church (1974) <doi:10.1007/BF01942293>.

License GPL (>= 2)

URL https://github.com/mharinga/spatialrisk, https://mharinga.github.io/spatialrisk/

LazyData true

LinkingTo Rcpp, RcppProgress

Depends R (>= 4.1.0)

Imports data.table, dplyr, fs, lifecycle, Rcpp, RcppProgress, rlang, sf, terra, units

Encoding UTF-8

Suggests classInt, colourvalues, GenSA, geohashTools, ggplot2, knitr, leafem, leafgl, leaflet, mapview, mgcv, rmarkdown, testthat, tmap, vroom

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-09-01 12:40:02 UTC

Contents

`choropleth` *Create a choropleth map of polygon-level values*

Description

Creates a choropleth map from an 'sf' object containing polygon-level reporting values, for example one produced by `summarise_points_by_polygon()`. Polygons are shaded according to values in a specified column, with clustering based on the Fisher–Jenks algorithm.

Usage

```
choropleth(
  data,
  value = "output",
  id = NULL,
  mode = c("plot", "view"),
```

```
  n = 7,
  legend_title = "Value",
  palette = "viridis",
  id_name = NULL,
  ...
)
```

Arguments

data	An object of class sf.
value	A string giving the name of the column used to shade the polygons.
id	Optional string giving the name of the column containing polygon IDs used for tooltips.
mode	A string indicating whether to create a static map ("plot", default) or an interactive map ("view").
n	Integer; number of clusters. Default is 7.
legend_title	A string giving the legend title.
palette	A palette name or vector of colors. See tmaptools::palette_explorer() for available palettes. Prefix the name with "-" to reverse the order. Default is "viridis".
id_name	Deprecated. Use id instead.
...	Additional arguments passed to tmap::tm_polygons().

Details

The function uses the Fisher–Jenks algorithm (style = "fisher") to classify values into n groups.

Value

A tmap object (static or interactive, depending on mode).

Author(s)

Martin Haringa

Examples

```
test <- summarise_points_by_polygon(nl_provincie, insurance, "amount")
choropleth(test, value = "amount_sum")
choropleth(test, value = "amount_sum", id = "areaname", mode = "view")
```

`concentration_hotspot` *Find fixed-radius concentration hotspots*

Description

Finds fixed-radius concentration hotspots in weighted point-level data. This is a computational building block for weighted circle-placement problems: given point locations and a fixed radius, find a centre whose surrounding circle contains a large aggregated value. In insurance applications, the weights may represent insured values or another exposure measure. This function is a wrapper around the decomposed workflow `prepare_spatialrisk`, `select_candidates`, and `optimize_hotspot`.

Usage

```
concentration_hotspot(
  data,
  value,
  n_hotspots = 1,
  radius = 200,
  cell_size = 100,
  grid_spacing = 1,
  max_refinement_points = 1500,
  lon = "lon",
  lat = "lat",
  crs_metric = 3035,
  progress = TRUE,
  method = c("continuous", "observed", "grid"),
  top_n = lifecycle::deprecated(),
  grid_precision = lifecycle::deprecated()
)
```

Arguments

`data`	A data.frame containing point-level exposures. Must include columns for longitude and latitude in EPSG:4326 and the value of interest. The coordinates are projected internally to ‘crs_metric‘.
`value`	A string giving the name of the numeric column in `data` to aggregate within each radius.
`n_hotspots`	Positive integer greater or equal to 1. Number of sequential non-overlapping hotspots to return. Default is `1`.
`radius`	Numeric. Radius of the circle in meters. This is typically the application-specific radius of interest. Default is `200`.
`cell_size`	Numeric. Size of the initial screening cells in meters. This is used by `method = "continuous"` and `method = "grid"`. Smaller values give a finer initial search but increase computation time. `method = "observed"` searches observed point locations and does not use this value as a search-grid resolution. Default is `100`.

`grid_spacing` Numeric. Spacing between candidate grid centres in the units of `crs_metric`; for the default metric CRS these units are meters. This is used by `method = "grid"` and by `method = "continuous"` only when the local subset is larger than `max_refinement_points` and the method falls back to grid refinement. It is not used by `method = "observed"`. Smaller values evaluate more candidate centres and increase computation time. Default is `1`.

`max_refinement_points` Positive integer. Maximum number of local points used for pair-intersection refinement. If the local subset contains more points, `method = "continuous"` automatically falls back to the grid refinement used by `method = "grid"`. Default is `1500`.

`lon` A string giving the longitude column in `data`. Default is `"lon"`.

`lat` A string giving the latitude column in `data`. Default is `"lat"`.

`crs_metric` Numeric. EPSG code for a projected CRS with meter units, used for distances, buffers, raster cells, and pair-intersection calculations. The default `3035` is ETRS89 / LAEA Europe and is a suitable default for Europe-wide applications. For other regions, choose a metric CRS appropriate to the study area, for example a local UTM zone, `5070` for the conterminous United States, or `3577` for Australia. For Asian portfolios there is no single universal choice; use a national projected CRS or the relevant UTM zone. Default is `3035`.

`progress` Logical. Whether to print progress messages for the main hotspot search steps. This is useful for larger portfolios and for `n_hotspots > 1`. Default is `TRUE`.

`method` Hotspot search strategy. `"continuous"` is the default and searches for a centre that may lie between observed points. `"observed"` searches only observed point locations as candidate centres. `"grid"` uses the original grid-refinement workflow.

`top_n` Deprecated. Use `n_hotspots` instead.

`grid_precision` Deprecated. Use `grid_spacing` instead.

Details

The default `method = "continuous"` first uses terra rasterisation and focal sums to identify candidate areas above an automatically estimated lower bound. It then refines all retained areas using observed local points and the circle centres implied by local point pairs. Local refinement subsets are retrieved from the raster cells that can affect each candidate area, using a conservative margin based on `radius` and `cell_size`. If more than `max_refinement_points` local points are involved, it falls back to the grid refinement used by `method = "grid"`. In that fallback case, `grid_spacing` controls the local refinement grid; otherwise the pair-intersection step does not use `grid_spacing`. The focal window includes the radius plus a raster-cell diagonal. For non-negative values this makes the focal sum an upper bound for exact centres in that cell. A second screening bound sums only active points within the radius of the centre-cell rectangle, including numerical boundary tolerance. Feasible trial centres in the most promising surviving cells can improve the lower bound before further pruning. With the default automatic lower bound, exact evaluation is consequently restricted to observed or pair-intersection centres whose own raster cell passed screening; the two centres generated by a point pair are tested separately. For a single hotspot, Rcpp processes these centres as a streaming angular sweep and maintains their exact totals over the complete active portfolio.

It therefore avoids materialising all centres or running a separate radius query for each one. A retained centre is never scored only against the points used to generate it. This centre-level pruning is disabled for a user-supplied threshold or negative values. Under non-negative weights, the default automatic threshold, complete pair-intersection refinement, and exact scoring with no grid fallback, this screening is optimality-preserving for the single-disk problem: a cell containing a strictly improving centre cannot be removed. The `"observed"` method is fast and deterministic, but can miss a larger hotspot when the optimal centre lies between observed points. The `"grid"` method uses a grid-based search with local refinement; smaller `grid_spacing` values generally increase search resolution and computation time. Use `prepare_spatialrisk`, `select_candidates`, and `optimize_hotspot` when these steps need to be run or inspected separately. The high-level function always uses the screened continuous search.

The underlying continuous hotspot problem can be viewed as a fixed-radius weighted circle placement problem. For point observations with non-negative values in a projected metric coordinate system, candidate centres formed by observed point locations and by intersections of radius-'r' circles around pairs of observations are sufficient to characterise the single-disk optimum. The practical `method = "continuous"` implementation retains that optimum under the screening conditions stated above. This guarantee does not apply to grid fallback, a user-supplied threshold, negative values, or joint optimisation of multiple circles.

Calling `optimize_hotspot()` directly on an object returned by `prepare_spatialrisk()` instead performs the complete geometric candidate search over the full active portfolio. Calling it after `select_candidates()` restricts candidate generation to the screened search state. In both routes, every retained centre is scored against the complete active portfolio. The direct full route does not use grid fallback and is principally intended for small validation or benchmark problems.

For `n_hotspots > 1`, hotspots are selected greedily: after each hotspot is found, the covered observations are removed before the next hotspot is computed. The resulting sequence is not necessarily globally optimal as a joint multi-circle problem.

Value

An object of class `hotspot`. The main components are `hotspots`, containing the selected centre coordinates and summed values, and `contributing_points`, containing the points inside the selected hotspot radii. The summed value column is named from `value`; for example, `value = "amount"` creates an `amount_sum` column. In `contributing_points`, `data_row` gives the row number of the contributing point in the original input data.

Author(s)

Martin Haringa

References

Chazelle, B. M. and Lee, D. T. (1986). On a circle placement problem. Computing, 36(1–2), 1–16. doi:10.1007/BF02238188.

Examples

```
portfolio <- Groningen[1:200, c("lon", "lat", "amount")]

hotspot <- concentration_hotspot(
```

```
  portfolio,
  value = "amount",
  radius = 200,
  n_hotspots = 2,
  cell_size = 100,
  progress = FALSE
)

hotspot$hotspots
head(hotspot$contributing_points)

observed_hotspot <- concentration_hotspot(
  portfolio,
  value = "amount",
  radius = 200,
  method = "observed",
  progress = FALSE
)

rbind(
  continuous = hotspot$hotspots[1, ],
  observed = observed_hotspot$hotspots
)
```

`convert_crs_df` *Convert Coordinate Reference System (CRS)*

Description

Convert Coordinate Reference System (CRS) of a data.frame from one CRS to another.

Usage

```
convert_crs_df(
  df,
  crs_from = 3035,
  crs_to = 4326,
  lon_from = "x",
  lat_from = "y",
  lon_to = "lon",
  lat_to = "lat"
)
```

Arguments

`df`	data.frame to be converted.
`crs_from`	CRS code of the original coordinate system (default: 3035).
`crs_to`	CRS code of the target coordinate system (default: 4326).

`lon_from`	column name of longitude values in `df` (default: "x").
`lat_from`	column name of latitude values in `df` (default: "y").
`lon_to`	column name for longitude values in the converted data frame (default: "lon").
`lat_to`	column name for latitude values in the converted data frame (default: "lat").

Value

data.frame with converted coordinates

Author(s)

Martin Haringa

`Groningen` *Example addresses in Groningen*

Description

A sample of addresses in Groningen with point coordinates in EPSG:4326 and an example numeric amount column.

Usage

```
Groningen
```

Format

A data frame with 25,000 rows and 9 variables:

street Street name.

number House number.

letter House letter.

suffix House number suffix.

postal_code Postal code.

city City name.

lon Longitude.

lat Latitude.

amount Example numeric amount.

Details

The `amount` column is an example value used in package examples and tests.

Source

BAG, the Dutch registry for addresses and buildings (Basisregistratie Adressen en Gebouwen), adapted for package examples.

haversine *Haversine great-circle distance*

Description

Calculates the shortest distance between two points on the Earth's surface using the Haversine formula, also known as the great-circle distance or "as the crow flies".

Usage

```
haversine(lat_from, lon_from, lat_to, lon_to, r = 6378137)
```

Arguments

lat_from	Numeric. Latitude(s) of the starting point(s) in decimal degrees (EPSG:4326).
lon_from	Numeric. Longitude(s) of the starting point(s) in decimal degrees (EPSG:4326).
lat_to	Numeric. Latitude(s) of the destination point(s) in decimal degrees (EPSG:4326).
lon_to	Numeric. Longitude(s) of the destination point(s) in decimal degrees (EPSG:4326).
r	Numeric. Radius of the Earth in meters (default = 6378137).

Details

This function is vectorized: if multiple coordinates are supplied, it returns one distance for each corresponding pair of points.

Value

A numeric vector with distances in the same unit as ‘r‘ (default: meters).

References

Sinnott, R.W, 1984. Virtues of the Haversine. Sky and Telescope 68(2): 159.

Examples

```
haversine(53.24007, 6.520386, 53.24054, 6.520386)

lat_from <- c(53.24, 52.37)
lon_from <- c(6.52, 4.90)
lat_to   <- c(48.85, 51.92)
lon_to   <- c(2.35, 4.48)
haversine(lat_from, lon_from, lat_to, lon_to)
```

insurance *Example insurance portfolio*

Description

A sample insurance portfolio with postal codes, insured amounts, population at 4-digit postcode level, and point coordinates in EPSG:4326.

Usage

```
insurance
```

Format

A data frame with 29,990 rows and 5 variables:

postcode 6-digit postal code.

population_pc4 Population for the corresponding 4-digit postcode area.

amount Insured amount.

lon Longitude of the corresponding 6-digit postal code.

lat Latitude of the corresponding 6-digit postal code.

Details

This dataset is intended for examples and tests of spatial concentration workflows.

Author(s)

Martin Haringa

knmi_historic_data *Retrieve historic weather data for the Netherlands*

Description

This function retrieves historic hourly weather data collected by official KNMI weather stations. See knmi_stations for the station metadata included in this package.

Usage

```
knmi_historic_data(
  startyear,
  endyear,
  stations = NULL,
  progress = interactive()
)
```

Arguments

startyear, endyear	Start and end year for the historic weather data. Both must be single whole years. KNMI hourly data is available from 1951.
stations	Optional station IDs to download. The default, NULL, downloads all stations in knmi_stations.
progress	Should a progress bar be shown? Defaults to interactive().

Format

The returned data frame contains the following columns:

- station = ID of measurement station;
- date = Date;
- FH = Hourly mean wind speed (in 0.1 m/s);
- FX = Maximum wind gust (in 0.1 m/s) during the hourly division;
- DR = Precipitation duration (in 0.1 hour) during the hourly division;
- RH = Hourly precipitation amount (in 0.1 mm) (-1 for <0.05 mm);
- city = City where the measurement station is located;
- lon = Longitude of station (EPSG:4326);
- lat = Latitude of station (EPSG:4326).

Details

The data is downloaded from KNMI when the function is called. This requires an internet connection and may take some time when many stations or years are requested.

Value

Data frame containing weather data and metadata for weather station locations.

Author(s)

Martin Haringa

Examples

```
## Not run:
knmi_historic_data(2015, 2019, stations = c(260, 280))

## End(Not run)
```

knmi_stations *KNMI weather stations*

Description

A data frame containing station IDs and location metadata for official KNMI weather stations in the Netherlands.

Usage

```
knmi_stations
```

Format

A data frame with 50 rows and 7 variables:

station Station ID.

city City where the station is located.

lon Longitude of the station in EPSG:4326.

lat Latitude of the station in EPSG:4326.

altitude Altitude of the station in meters.

X Projected X coordinate of the station in EPSG:32631.

Y Projected Y coordinate of the station in EPSG:32631.

Author(s)

Martin Haringa

Source

Royal Netherlands Meteorological Institute (KNMI), adapted for package examples.

map_points *Create interactive point map*

Description

Creates an interactive map for a data.frame containing point coordinates, optionally colored by a selected variable.

Usage

```
map_points(
  data,
  value = NULL,
  lon = "lon",
  lat = "lat",
  crs = 4326,
  at = NULL,
  layer_name = NULL,
  ...
)
```

Arguments

data	A data.frame containing columns for longitude and latitude.
value	A string giving the name of the column in data to be visualized. If NULL, points are mapped without a color variable.
lon	A string with the name of the column containing longitude values. Default is "lon".
lat	A string with the name of the column containing latitude values. Default is "lat".
crs	Integer; EPSG code for the coordinate reference system. Default is 4326.
at	Optional numeric vector; breakpoints used for visualization.
layer_name	Optional layer name passed to mapview.
...	Additional arguments passed to mapview::mapview().

Value

An interactive mapview object.

Examples

```
## Not run:
map_points(Groningen, value = "amount")

## End(Not run)
```

---

nl_corop *COROP regions in the Netherlands*

Description

An sf object with COROP region geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_corop
```

Format

A simple feature object with 40 rows and 5 variables:

corop_nr COROP number.

areaname COROP region name.

geometry COROP region geometry.

lon Longitude of the COROP centroid.

lat Latitude of the COROP centroid.

Details

COROP regions are regional areas used for analytical purposes by, among others, Statistics Netherlands.

Author(s)

Martin Haringa

Source

Statistics Netherlands (CBS), adapted for package examples.

---

nl_gemeente *Municipalities in the Netherlands*

Description

An `sf` object with municipal geometries for the Netherlands in 2021. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_gemeente
```

Format

A simple feature object with 380 rows and 6 variables:

id Municipality identifier.

code Municipality code.

areaname Municipality name.

lon Longitude of the municipality centroid.

lat Latitude of the municipality centroid.

geometry Municipality geometry.

Author(s)

Martin Haringa

Source

Statistics Netherlands (CBS), adapted for package examples.

---

nl_postcode2 *Two-digit postcode regions in the Netherlands*

Description

An `sf` object with 2-digit postcode region geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_postcode2
```

Format

A simple feature object with 90 rows and 4 variables:

areaname 2-digit postcode area.

geometry Postcode region geometry.

lon Longitude of the 2-digit postcode centroid.

lat Latitude of the 2-digit postcode centroid.

Details

Postal codes in the Netherlands are alphanumeric and consist of four digits followed by two uppercase letters. This object aggregates those codes to their first two digits.

Author(s)

Martin Haringa

Source

Adapted from Dutch postcode boundary data for package examples.

nl_postcode3 *Three-digit postcode regions in the Netherlands*

Description

An `sf` object with 3-digit postcode region geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_postcode3
```

Format

A simple feature object with 799 rows and 4 variables:

areaname 3-digit postcode area.

geometry Postcode region geometry.

lon Longitude of the 3-digit postcode centroid.

lat Latitude of the 3-digit postcode centroid.

Details

Postal codes in the Netherlands are alphanumeric and consist of four digits followed by two uppercase letters. This object aggregates those codes to their first three digits.

Author(s)

Martin Haringa

Source

Adapted from Dutch postcode boundary data for package examples.

nl_postcode4 *Four-digit postcode regions in the Netherlands*

Description

An `sf` object with 4-digit postcode region geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_postcode4
```

Format

A simple feature object with 4053 rows and 7 variables:

pc4 4-digit postcode.

areaname Name of corresponding 4-digit postcode area.

city City name.

biggest_20cities Whether the area is in one of the twenty largest cities in the Netherlands.

geometry Postcode region geometry.

lon Longitude of the 4-digit postcode centroid.

lat Latitude of the 4-digit postcode centroid.

Details

Postal codes in the Netherlands are alphanumeric and consist of four digits followed by two uppercase letters. This object aggregates those codes to their first four digits.

Author(s)

Martin Haringa

Source

Adapted from Dutch postcode boundary data for package examples.

`nl_provincie` *Provinces in the Netherlands*

Description

An `sf` object with province geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_provincie
```

Format

A simple feature object with 12 rows and 4 variables:

areaname Province name.

geometry Province geometry.

lon Longitude of the province centroid.

lat Latitude of the province centroid.

Author(s)

Martin Haringa

Source

Statistics Netherlands (CBS), adapted for package examples.

`plot.hotspot` *Plot concentration hotspot results*

Description

Visualise objects returned by `concentration_hotspot()`. The default plot shows hotspot centres, fixed-radius buffers, and the contributing points. For terra-based results, diagnostic raster layers can also be plotted.

Usage

```
## S3 method for class 'hotspot'
plot(
  x,
  type = c("concentration", "focal", "rasterized", "updated_focal"),
  color1 = NULL,
  max.rad = 20,
  ...
)
```

Arguments

`x`	An object of class `hotspot`.
`type`	Plot type. `"concentration"` shows hotspot buffers and contributing points and works for all hotspot search methods. `"focal"`, `"rasterized"`, and `"updated_focal"` are diagnostic terra layers and are only available for terra-based results.
`color1`	Optional colour or colours for hotspot buffers and points. If `NULL`, colours are chosen with `grDevices::hcl.colors()`.
`max.rad`	Maximum point radius passed to `mapview::mapview()`. Default is `20`.
`...`	Additional arguments passed to `mapview::mapview()` for the contributing point layer when `type = "concentration"`, or to the raster mapview call for diagnostic raster layers.

Details

The observed-points hotspot method does not create terra raster or focal objects. For observed-points results, use `type = "concentration"`.

Value

A `mapview` object.

`points_within_radius` *Find points within radius around one or more centre coordinates*

Description

This function selects rows from a data frame whose longitude/latitude coordinates fall within a given radius (in meters) from one or more specified centre points. It also calculates the distance of each point to the centre.

Usage

```
points_within_radius(
  data,
  lon_center,
  lat_center,
  lon = "lon",
  lat = "lat",
  radius = 200,
  sort = TRUE
)
```

Arguments

`data`	A data frame containing at least longitude and latitude columns.
`lon_center`	Numeric scalar or vector, longitude(s) of the circle centre(s).
`lat_center`	Numeric scalar or vector, latitude(s) of the circle centre(s).
`lon`	A string with the name of the longitude column in ‘data‘.
`lat`	A string with the name of the latitude column in ‘data‘.
`radius`	Numeric, circle radius in meters. Default is 200.
`sort`	Logical, if ‘TRUE‘ results are sorted by distance within each centre.

Value

A data frame subset of ‘data‘ with an extra column ‘distance_m‘ and if multiple centres are provided, also a column ‘center_index‘.

prepare_spatialrisk *Prepare fixed-radius concentration hotspot analysis*

Description

'prepare_spatialrisk()', 'select_candidates()', and 'optimize_hotspot()' expose the main steps used by concentration_hotspot. They are useful when the intermediate search state needs to be inspected or when the same prepared portfolio is used in more than one hotspot search strategy.

Usage

```
prepare_spatialrisk(
  data,
  value,
  radius = 200,
  cell_size = 100,
  lon = "lon",
  lat = "lat",
  crs_metric = 3035
)

select_candidates(
  x,
  grid_spacing = 1,
  max_refinement_points = 1500,
  method = c("continuous", "observed", "grid"),
  threshold = NULL,
  progress = TRUE,
  grid_precision = lifecycle::deprecated()
)

optimize_hotspot(
  x,
  n_hotspots = 1,
  progress = TRUE,
  top_n = lifecycle::deprecated()
)

## S3 method for class 'spatialrisk_hotspot_workflow'
plot(x, type = c("auto", "raster", "candidates"), ...)
```

Arguments

data	A data.frame containing point-level exposures. Must include longitude and latitude in EPSG:4326 and the value of interest. Coordinates are projected internally to 'crs_metric'.
value	A string giving the numeric column in 'data' to aggregate within each radius.

radius	Numeric. Radius of the circle in meters.
cell_size	Numeric. Size of the raster cells used for the initial screening raster.
lon	A string giving the longitude column in ‘data‘.
lat	A string giving the latitude column in ‘data‘.
crs_metric	Numeric. EPSG code for a projected CRS with meter units. The default ‘3035‘ is ETRS89 / LAEA Europe.
x	A prepared spatial-risk workflow object returned by ‘prepare_spatialrisk()‘ or ‘select_candidates()‘.
grid_spacing	Numeric. Spacing between candidate grid centres in the units of ‘crs_metric‘; for the default metric CRS these units are meters. Used for grid-based refinement.
max_refinement_points	Positive integer. Maximum number of local points used for pair-intersection refinement before falling back to grid refinement in a search state produced by ‘select_candidates()‘. Direct optimisation of a prepared object never falls back to a grid; values above this limit produce a computational-cost warning instead.
method	Hotspot search strategy. ‘"continuous"‘ is the default and searches for centres that may lie between observed points. ‘"observed"‘ searches only observed point locations. ‘"grid"‘ uses the grid-refinement workflow.
threshold	Optional numeric lower bound for candidate focal cells. If ‘NULL‘, the lower bound is estimated using the same preliminary refinement step as ‘concentration_hotspot()‘.
progress	Logical. Whether to print progress messages.
grid_precision	Deprecated. Use ‘grid_spacing‘ instead.
n_hotspots	Positive integer. Number of non-overlapping hotspots to return.
top_n	Deprecated. Use ‘n_hotspots‘ instead.
type	Plot type. ‘"auto"‘ shows the prepared raster before candidate selection and selected focal candidate cells afterwards.
...	Additional arguments passed to ‘mapview::mapview()‘.

Details

The three-step interface decomposes the hotspot workflow without replacing ‘concentration_hotspot()‘. The wrapper remains the simplest public function for normal use, while the decomposed functions make the intermediate candidate selection visible.

‘optimize_hotspot()‘ optimises over the candidate search state supplied to it. Called directly on the output of ‘prepare_spatialrisk()‘, it performs a full geometric search: candidate centres are the active observed locations and the valid radius-circle intersections generated by every active point pair no farther than twice the radius apart. This route does not use raster screening or grid fallback and is intended mainly for small portfolios, diagnostics, validation, and methodological benchmarks. Pair generation can be computationally expensive.

In ‘select_candidates()‘, ‘threshold = NULL‘ estimates a lower bound by taking the highest focal raster cells, refining those cells on a small local grid, evaluating those centres in the projected metric coordinates, and using the best feasible value as the candidate-cell threshold. The focal cells first

survive when their moving-window sum is at least this lower bound. For continuous search with non-negative values and an automatic threshold, a second bound sums only active points whose minimum distance to the closed centre-cell rectangle is within the radius, allowing for the scoring tolerance. It does not count the entire value of an extra raster cell. The five highest surviving point bounds also seed feasible trial centres; only their actual portfolio sums can raise the lower bound. Cells with a point bound strictly below that lower bound are removed. The focal window includes the requested radius plus a raster-cell diagonal. With non-negative values, its sum is therefore an upper bound for every exact centre located in that focal cell. Under the default automatic threshold, observed and pair-intersection centres are mapped back to the raster and exact evaluation is restricted to centres that lie in a selected cell. For a point pair, its two possible circle centres are screened separately. For a single hotspot, a streaming Rcpp angular sweep maintains the complete active-portfolio total at the retained pair-intersection events; this avoids materialising and separately querying every centre. With complete pair-intersection refinement and exact scoring, and provided no grid fallback occurs, this removes no cell that can contain a strictly improving centre and therefore preserves a global optimum of the single-disk problem. The guarantee does not apply to a user-supplied threshold, negative values, or grid fallback.

Candidate screening never defines the evaluation portfolio. Every retained centre is scored against all records in the complete active portfolio, so a point outside the candidate-generation subset still contributes when it lies within the radius. Point-to-raster-cell membership is stored during preparation and reused to retrieve nearby records; candidate regions within one hotspot iteration share the same active-portfolio evaluation state. When ‘optimize_hotspot(n_hotspots > 1)‘ or ‘concentration_hotspot(n_hotspots > 1)‘ is used, the points in the selected hotspot are removed and the candidate-selection logic is run again for the next hotspot. Therefore the number of candidate cells shown by ‘select_candidates()‘ for the first iteration does not limit the number of hotspots returned by ‘n_hotspots‘.

Value

‘prepare_spatialrisk()‘ and ‘select_candidates()‘ return an object of class ‘spatialrisk_hotspot_workflow‘. ‘optimize_hotspot()‘ returns the same ‘hotspot‘ object structure as concentration_hotspot.

Author(s)

Martin Haringa

Examples

```
portfolio <- Groningen[1:200, c("lon", "lat", "amount")]

model <- prepare_spatialrisk(portfolio, value = "amount", radius = 200,
                             cell_size = 100)

# Full geometric reference search
full <- optimize_hotspot(model, progress = FALSE)

# Screened continuous search
screened <- model |>
  select_candidates(progress = FALSE) |>
  optimize_hotspot(progress = FALSE)
```

```
full$hotspots
screened$hotspots
```

`radius_sum` *Sum values within a radius around target coordinates*

Description

Calculates the sum of all observations from a reference data set that fall within a given radius (in meters) of each target point.

Usage

```
radius_sum(
  targets,
  reference,
  value,
  lon_targets = "lon",
  lat_targets = "lat",
  lon_reference = "lon",
  lat_reference = "lat",
  radius = 200,
  progress = TRUE,
  result_col = "radius_sum"
)
```

Arguments

`targets`	A data.frame of target points for which sums are calculated. Must include at least columns for longitude and latitude.
`reference`	A data.frame containing reference points. Must include at least columns for longitude, latitude, and the value of interest to summarize.
`value`	A string giving the name of the column in ‘reference‘ to be summed.
`lon_targets`	A string with the name of the longitude column in ‘targets‘. Default is ‘"lon"‘.
`lat_targets`	A string with the name of the latitude column in ‘targets‘. Default is ‘"lat"‘.
`lon_reference`	A string with the name of the longitude column in ‘reference‘. Default is ‘"lon"‘.
`lat_reference`	A string with the name of the latitude column in ‘reference‘. Default is ‘"lat"‘.
`radius`	Numeric. Radius of the circle in meters. Must be positive (default: 200).
`progress`	Logical. Whether to display a progress bar. Default is ‘TRUE‘.
`result_col`	A string giving the name of the output column. Default is ‘"radius_sum"‘.

Details

This function uses a C++ backend for efficient distance calculations (Haversine formula).

Value

A data.frame equal to ‘targets‘ with an additional numeric column named by ‘result_col‘ containing the summed values from ‘reference‘.

Author(s)

Martin Haringa

Examples

```
targets <- data.frame(location = c("p1", "p2"),
                      lon = c(6.561561, 6.561398),
                      lat = c(53.21369, 53.21326))

reference <- data.frame(lon = c(6.5614, 6.5620, 6.5630),
                        lat = c(53.2132, 53.2140, 53.2150),
                        amount = c(10, 20, 15))

radius_sum(targets, reference, value = "amount", radius = 100,
           progress = FALSE)
```

---

summarise_points_by_polygon

Summarise point exposures by reporting polygon

Description

Spatially joins point data to polygon geometries and summarises a numeric point exposure or value for each reporting area.

Usage

```
summarise_points_by_polygon(
  polygons,
  points,
  value,
  fun = sum,
  lon = "lon",
  lat = "lat",
  crs = 4326,
  output_col = NULL,
  na.rm = TRUE,
  outside = c("message", "warning", "ignore"),
  repair_geometry = TRUE
)
```

Arguments

polygons	An object of class sf containing polygon geometries.
points	A data.frame containing point coordinates and the value to summarise.
value	A string giving the name of the numeric column in points to summarise.
fun	A summary function, such as sum, mean, or length. Default is sum.
lon	A string with the name of the longitude column in points. Default is "lon".
lat	A string with the name of the latitude column in points. Default is "lat".
crs	Coordinate reference system of the point coordinates. Default is 4326.
output_col	Optional string giving the name of the output column. If NULL, the name is created from value and fun, for example "amount_sum" or "amount_mean".
na.rm	Logical. Whether to remove missing values when fun supports an na.rm argument. Default is TRUE.
outside	What to do when points fall outside all polygons: "message" (default), "warning", or "ignore".
repair_geometry	Logical. Whether to try sf::st_buffer(x, 0) if transforming polygon geometries fails. Default is TRUE.

Value

An sf object equal to polygons with an additional summary column.

Examples

```
summarise_points_by_polygon(
  polygons = nl_postcode2,
  points = insurance,
  value = "amount",
  fun = sum
)
```

Index