Package 'sdsfun'

September 30, 2024

Title Spatial Data Science Complementary Features

Version 0.3.0

Description Wrapping and supplementing commonly used functions in the R ecosystem related to spatial data science, while serving as a basis for other packages maintained by Wenbo Lv.

License GPL-3

Encoding UTF-8

URL https://stscl.github.io/sdsfun/, https://github.com/stscl/sdsfun

BugReports https://github.com/stscl/sdsfun/issues

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports dplyr, geosphere, magrittr, purrr, sf, spdep, stats, tibble

Suggests ggplot2, Rcpp, testthat (>= 3.0.0)

Config/testthat/edition 3

LinkingTo Rcpp

NeedsCompilation yes

Author Wenbo Lv [aut, cre, cph] (<https://orcid.org/0009-0002-6003-3800>)

Maintainer Wenbo Lv <lyu.geosocial@gmail.com>

Repository CRAN

Date/Publication 2024-09-30 04:50:02 UTC

Contents

dummy_tbl 2
dummy_vector 3
fuzzyoverlay 3
inverse_distance_swm 4
normalize_vector 5
sf_distance_matrix 5

sf_geometry_name 6
sf_geometry_type 6
sf_utm_proj_wgs84 7
sf_voronoi_diagram 8
spdep_contiguity_swm 8
spdep_distance_swm 10
spdep_nb 11
spdep_skater 12
standardize_vector 13

Index **14**

`dummy_tbl` *transforming a category tibble into the corresponding dummy variable tibble*

Description

transforming a category tibble into the corresponding dummy variable tibble

Usage

```
dummy_tbl(tbl)
```

Arguments

`tbl`	A `tibble` or `data.frame`.

Value

A `tibble`

Examples

```
a = tibble::tibble(x = 1:3,y = 4:6)
dummy_tbl(a)
```

dummy_vector *transforming a categorical variable into dummy variables*

Description

transforming a categorical variable into dummy variables

Usage

```
dummy_vector(x)
```

Arguments

x	An integer vector or can be converted into an integer vector.

Value

A matrix.

Examples

```
dummy_vector(c(1,1,3,2,4,6))
```

fuzzyoverlay *spatial fuzzy overlay*

Description

spatial fuzzy overlay

Usage

```
fuzzyoverlay(formula, data, method = "and")
```

Arguments

formula	A formula of spatial fuzzy overlay.
data	A data.frame or tibble of discretized data.
method	(optional) Overlay methods. When method is and, use min to do fuzzy overlay; and when method is or,use max to do fuzzy overlay. Default is and.

Value

A numeric vector.

Note

Independent variables in the `data` provided to `fuzzyoverlay()` must be discretized variables, and dependent variable are continuous variable.

Examples

```
sim = tibble::tibble(y = stats::runif(7,0,10),
                     x1 = c(1,rep(2,3),rep(3,3)),
                     x2 = c(rep(1,2),rep(2,2),rep(3,3)))
fo1 = fuzzyoverlay(y~x1+x2,data = sim, method = 'and')
fo1
fo2 = fuzzyoverlay(y~x1+x2,data = sim, method = 'or')
fo2
```

`inverse_distance_swm` *construct inverse distance weight*

Description

Function for constructing inverse distance weight.

Usage

```
inverse_distance_swm(sfj, power = 1, bandwidth = NULL)
```

Arguments

`sfj`	Vector object that can be converted to `sf` by `sf::st_as_sf()`.
`power`	(optional) Default is 1. Set to 2 for gravity weights.
`bandwidth`	(optional) When the distance is bigger than bandwidth, the corresponding part of the weight matrix is set to 0. Default is `NULL`, which means not use the bandwidth.

Details

The inverse distance weight formula is $w_{ij} = 1/d_{ij}^{\alpha}$

Value

A inverse distance weight matrices with class of `matrix`.

Examples

```
library(sf)
pts = read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
wt = inverse_distance_swm(pts)
wt[1:5,1:5]
```

normalize_vector *normalization*

Description

normalization

Usage

```
normalize_vector(x, to_left = 0, to_right = 1)
```

Arguments

x	A continuous numeric vector.
to_left	(optional) Specified minimum. Default is 0.
to_right	(optional) Specified maximum. Default is 1.

Value

A continuous vector which has normalized.

Examples

```
normalize_vector(c(-5,1,5,0.01,0.99))
```

sf_distance_matrix *generates distance matrix*

Description

Generates distance matrix for sf object

Usage

```
sf_distance_matrix(sfj)
```

Arguments

sfj	An sf object or can be converted to sf by sf::st_as_sf().

Value

A matrix.

Examples

```
library(sf)
pts = read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
pts_distm = sf_distance_matrix(pts)
pts_distm[1:5,1:5]
```

`sf_geometry_name` *sf object geometry column name*

Description

Get the geometry column name of an sf object

Usage

```
sf_geometry_name(sfj)
```

Arguments

`sfj`	An `sf` object.

Value

A character.

Examples

```
library(sf)
gzma = read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
sf_geometry_name(gzma)
```

`sf_geometry_type` *sf object geometry type*

Description

Get the geometry type of an sf object

Usage

```
sf_geometry_type(sfj)
```

Arguments

`sfj`	An `sf` object.

Value

A lowercase character vector

Examples

```
library(sf)
gzma = read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
sf_geometry_type(gzma)
```

`sf_utm_proj_wgs84` *generates wgs84 utm projection epsg coding character*

Description

Generates a utm projection epsg coding character corresponding to an `sfj` object under the WGS84 spatial reference.

Usage

```
sf_utm_proj_wgs84(sfj)
```

Arguments

`sfj`	An `sf` object or can be converted to `sf` by `sf::st_as_sf()`.

Details

For more details, please refer to https://zhuanlan.zhihu.com/p/670055831.

Value

A character.

Examples

```
library(sf)
gzma = read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
sf_utm_proj_wgs84(gzma)
```

sf_voronoi_diagram *generates voronoi diagram*

Description

Generates Voronoi diagram (Thiessen polygons) for sf object

Usage

```
sf_voronoi_diagram(sfj)
```

Arguments

sfj	An sf object.

Value

An sf object of polygon geometry type or can be converted to this by sf::st_as_sf().

Note

Only sf objects of (multi-)point type are supported to generate voronoi diagram and the returned result includes only the geometry column.

Examples

```
library(sf)
pts = read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))
pts_v = sf_voronoi_diagram(pts)

library(ggplot2)
ggplot() +
  geom_sf(data = pts_v, color = 'red',
          fill = 'transparent') +
  geom_sf(data = pts, color = 'blue', size = 1.25) +
  theme_void()
```

spdep_contiguity_swm *constructs spatial weight matrices based on contiguity*

Description

Constructs spatial weight matrices based on contiguity via spdep package.

Usage

```
spdep_contiguity_swm(
  sfj,
  queen = TRUE,
  k = NULL,
  order = 1L,
  cumulate = TRUE,
  style = "W",
  zero.policy = TRUE
)
```

Arguments

sfj	An sf object or can be converted to sf by sf::st_as_sf().
queen	(optional) if TRUE, using queen contiguity, otherwise rook contiguity. Default is TRUE.
k	(optional) The number of nearest neighbours. Ignore this parameter when not using distance based neighbours to construct spatial weight matrices.
order	(optional) The order of the adjacency object. Default is 1.
cumulate	(optional) Whether to accumulate adjacency objects. Default is TRUE.
style	(optional) style can take values W, B, C, and S. More to see spdep::nb2mat(). Default is W.
zero.policy	(optional) if FALSE stop with error for any empty neighbour sets, if TRUE permit the weights list to be formed with zero-length weights vectors. Default is TRUE.

Value

A matrix

Note

When k is set to a positive value, using K-Nearest Neighbor Weights.

Examples

```
library(sf)
gzma = read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))

wt1 = spdep_contiguity_swm(gzma, k = 6, style = 'B')
wt2 = spdep_contiguity_swm(gzma, queen = TRUE, style = 'B')
wt3 = spdep_contiguity_swm(gzma, queen = FALSE, order = 2, style = 'B')
```

spdep_distance_swm *constructs spatial weight matrices based on distance*

Description

Constructs spatial weight matrices based on distance via spdep package.

Usage

```
spdep_distance_swm(
  sfj,
  kernel = NULL,
  k = NULL,
  bandwidth = NULL,
  power = 1,
  style = "W",
  zero.policy = TRUE
)
```

Arguments

sfj	An sf object or can be converted to sf by sf::st_as_sf().
kernel	(optional) The kernel function, can be one of uniform, triangular,quadratic(epanechnikov),quartic and gaussian. Default is NULL.
k	(optional) The number of nearest neighbours. Default is NULL. Only useful when kernel is provided.
bandwidth	(optional) The bandwidth, default is NULL. When the spatial reference of sf object is the geographical coordinate system, the unit of bandwidth is km. The unit used in the projection coordinate system are consistent with those used in the sf object coordinate system.
power	(optional) Default is 1. Useful when kernel is not provided.
style	(optional) style can take values W, B, C, and S. More to see spdep::nb2mat(). Default is W. For spatial weights based on distance functions, a style of B means using the original value of the calculated distance function.
zero.policy	(optional) if FALSE stop with error for any empty neighbour sets, if TRUE permit the weights list to be formed with zero-length weights vectors. Default is TRUE.

Details

five different kernel weight functions:

- uniform: $K_{(z)} = 1/2$,for $|z| < 1$
- triangular $K_{(z)} = 1 - |z|$,for $|z| < 1$
- quadratic (epanechnikov) $K_{(z)} = \frac{3}{4}\left(1 - z^2\right)$,for $|z| < 1$

- quartic $K_{(z)} = \frac{15}{16}\left(1 - z^2\right)^2$,for $|z| < 1$
- gaussian $K_{(z)} = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$

For the equation above, $z = d_{ij}/h_i$ where h_i is the bandwidth

Value

A matrix

Note

When `kernel` is setting, using distance weight based on kernel function, Otherwise the inverse distance weight will be used.

Examples

```
library(sf)
pts = read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))

wt1 = spdep_distance_swm(pts, style = 'B')
wt2 = spdep_distance_swm(pts, kernel = 'gaussian')
wt3 = spdep_distance_swm(pts, k = 3, kernel = 'gaussian')
wt4 = spdep_distance_swm(pts, k = 3, kernel = 'gaussian', bandwidth = 10000)
```

---

spdep_nb *construct neighbours list*

Description

construct neighbours list

Usage

```
spdep_nb(sfj, queen = TRUE, k = NULL, order = 1L, cumulate = TRUE)
```

Arguments

`sfj`	An `sf` object or can be converted to `sf` by `sf::st_as_sf()`.
`queen`	(optional) if `TRUE`, using queen contiguity, otherwise rook contiguity. Default is `TRUE`.
`k`	(optional) The number of nearest neighbours. Ignore this parameter when not using distance based neighbours.
`order`	(optional) The order of the adjacency object. Default is `1`.
`cumulate`	(optional) Whether to accumulate adjacency objects. Default is `TRUE`.

Value

A neighbours list with class nb

Note

When k is set to a positive value, using K-Nearest Neighbor

Examples

```
library(sf)
pts = read_sf(system.file('extdata/pts.gpkg',package = 'sdsfun'))

nb1 = spdep_nb(pts, k = 6)
nb2 = spdep_nb(pts, queen = TRUE)
nb3 = spdep_nb(pts, queen = FALSE, order = 2)
```

---

spdep_skater *spatial c(k)luster analysis by tree edge removal*

Description

SKATER forms clusters by spatially partitioning data that has similar values for features of interest.

Usage

```
spdep_skater(sfj, k = 6, nb = NULL, ini = 5, ...)
```

Arguments

sfj	An sf object of observation data. Please ensure that the attribute columns are included in the SKATER analysis.
k	(optional) The number of clusters. Default is 6.
nb	(optional) A neighbours list with class nb. If the input nb is NULL, it will be constructed automatically using spdep_nb().
ini	(optional) The initial node in the minimal spanning tree. Defaul is 5.
...	(optional) Other parameters passed to spdep::skater().

Value

A numeric vector of clusters.

Examples

```
library(sf)
gzma = read_sf(system.file('extdata/gzma.gpkg',package = 'sdsfun'))
gzma_c = spdep_skater(gzma,8)
gzma$group = gzma_c
plot(gzma["group"])
```

`standardize_vector` *standardization*

Description

To calculate the Z-score using variance normalization, the formula is as follows:
$Z = \frac{(x - mean(x))}{sd(x)}$

Usage

```
standardize_vector(x)
```

Arguments

x	A numeric vector

Value

A standardized numeric vector

Examples

```
standardize_vector(1:10)
```

Index

dummy_tbl, 2
dummy_vector, 3

fuzzyoverlay, 3

inverse_distance_swm, 4

normalize_vector, 5

sf_distance_matrix, 5
sf_geometry_name, 6
sf_geometry_type, 6
sf_utm_proj_wgs84, 7
sf_voronoi_diagram, 8
spdep_contiguity_swm, 8
spdep_distance_swm, 10
spdep_nb, 11
spdep_skater, 12
standardize_vector, 13