Package ‘nngeo’

May 12, 2019

Type Package

Title k-Nearest Neighbor Join for Spatial Data

Version 0.2.8

Description K-nearest neighbor search for projected and non-projected 'sf' spatial layers. Nearest neighbor search uses (1) C code from GeographicLib for lon-lat point layers, (2) function nn2() from package 'RANN' for projected point layers, or (3) function st_distance() from package 'sf' for line or polygon layers.

Imports RANN, units, Rcpp, sp, methods, lwgeom

Depends R (>= 3.2.3), sf (>= 0.6-0)

License MIT + file LICENSE

LazyData TRUE

RoxygenNote 6.1.1

Suggests DBI, RPostgreSQL, knitr, rmarkdown, testthat

VignetteBuilder knitr

URL https://github.com/michaeldorman/nngeo/

BugReports https://github.com/michaeldorman/nngeo/issues/

Encoding UTF-8

NeedsCompilation yes

Author Michael Dorman [aut, cre],
Johnathan Rush [ctb],
Ian Hough [ctb],
Charles F.F Karney [ctb, cph] (Author of included C code from 'GeographicLib' for geodesic distance)

Maintainer Michael Dorman <dorman@post.bgu.ac.il>

Repository CRAN

Date/Publication 2019-05-12 17:20:03 UTC

R topics documented:

`cities` *Point layer of the three largest cities in Israel*

Description

A `sf` POINT layer of the three largest cities in Israel: Jerusalem, Tel-Aviv and Haifa.

Usage

```
cities
```

Format

A `sf` POINT layer with 3 features and 1 attribute:

name Town name

`st_connect` *Create lines between features of two layers*

Description

Create lines between features of two layers

Usage

```
st_connect(x, y, ids = NULL, dist, progress = TRUE, ...)
```

Arguments

x	Object of class sf or sfc
y	Object of class sf or sfc
ids	A sparse list representation of features to connect such as returned by function st_nn. If NULL the function automatically calculates ids using st_nn
dist	Sampling distance interval for generating outline points to choose from. Required when at least one of x or y is a line or polygon layer. Should be given in CRS units for projected layers, or in meters for layers in lon/lat
progress	Display progress bar? (default TRUE)
...	Other arguments passed to st_nn when calculating ids, such as k and maxdist

Value

Object of class sfc with geometry type LINESTRING

Examples

```
# Nearest 'city' per 'town'
l = st_connect(towns, cities)
plot(st_geometry(towns), col = "darkgrey")
plot(st_geometry(l), add = TRUE)
plot(st_geometry(cities), col = "red", add = TRUE)

# Ten nearest 'towns' per 'city'
l = st_connect(cities, towns, k = 10)
plot(st_geometry(towns), col = "darkgrey")
plot(st_geometry(l), add = TRUE)
plot(st_geometry(cities), col = "red", add = TRUE)

## Not run:

# Nearest 'city' per 'town', search radius of 30 km
cities = st_transform(cities, 32636)
towns = st_transform(towns, 32636)
l = st_connect(cities, towns, k = nrow(towns), maxdist = 30000)
plot(st_geometry(towns), col = "darkgrey")
plot(st_geometry(l), add = TRUE)
plot(st_buffer(st_geometry(cities), units::set_units(30, km)), border = "red", add = TRUE)

# The 20-nearest towns for each water body
water = st_transform(water, 32636)
l = st_connect(water[-1, ], towns, k = 20, dist = 100)
plot(st_geometry(water[-1, ]), col = "lightblue", border = NA)
plot(st_geometry(towns), col = "darkgrey", add = TRUE)
plot(st_geometry(l), col = "red", add = TRUE)


# The 2-nearest water bodies for each town
l = st_connect(towns, water[-1, ], k = 2, dist = 100)
plot(st_geometry(water[-1, ]), col = "lightblue", border = NA)
```

```
plot(st_geometry(towns), col = "darkgrey", add = TRUE)
plot(st_geometry(l), col = "red", add = TRUE)


## End(Not run)
```

---

st_ellipse *Calculate ellipse polygon*

Description

The function calculates ellipse polygons, given centroid locations and sizing on the x and y axes.

Usage

```
st_ellipse(pnt, ex, ey, res = 30)
```

Arguments

pnt	Object of class `sf` or `sfc` (type `"POINT"`) representing centroid locations
ex	Size along x-axis, in CRS units
ey	Size along y-axis, in CRS units
res	Number of points the ellipse polygon consists of (default `30`)

Value

Object of class `sfc` (type `"POLYGON"`) containing ellipse polygons

References

Based on StackOverflow answer by user 'fdetsch' -

https://stackoverflow.com/questions/35841685/add-an-ellipse-on-raster-plot-in-r

Examples

```
# Sample data
dat = data.frame(
  x = c(1, 1, -1, 3, 3),
  y = c(0, -3, 2, -2, 0),
  ex = c(0.5, 2, 2, 0.3, 0.6),
  ey = c(0.5, 0.2, 1, 1, 0.3),
  stringsAsFactors = FALSE
)
dat = st_as_sf(dat, coords = c("x", "y"))
dat

# Plot 1
plot(dat %>% st_geometry, graticule = TRUE, axes = TRUE, main = "Input")
```

```
text(dat %>% st_coordinates, as.character(1:nrow(dat)), pos = 2)

# Calculate ellipses
el = st_ellipse(pnt = dat, ex = dat$ex, ey = dat$ey)

# Plot 2
plot(el, graticule = TRUE, axes = TRUE, main = "Output")
plot(dat %>% st_geometry, pch = 3, add = TRUE)
text(dat %>% st_coordinates, as.character(1:nrow(dat)), pos = 2)
```

`st_nn` *Nearest Neighbor Search for Simple Features*

Description

The function returns the indices of layer `y` which are nearest neighbors of each feature of layer `x`. The number of nearest neighbors `k` and the search radius `maxdist` can be modified.

The function has three modes of operation -

- lon-lat points - Calculation using C code from `GeographicLib`, similar to `sf::st_distance`
- projected points - Calculation using `RANN::nn2`, a fast search method based on the ANN C++ library
- lines or polygons, either lon-lat or projected - Calculation based on `sf::st_distance`

Usage

```
st_nn(x, y, sparse = TRUE, k = 1, maxdist = Inf,
  returnDist = FALSE, progress = TRUE)
```

Arguments

`x`	Object of class `sf` or `sfc`
`y`	Object of class `sf` or `sfc`
`sparse`	logical; should a sparse index list be returned (TRUE) or a dense logical matrix? See below.
`k`	The maximum number of nearest neighbors to compute. Default is `1`, meaning that only a single point (nearest neighbor) is returned
`maxdist`	Search radius (in meters). Points farther than search radius are not considered. Default is `Inf` meaning that search is unconstrained
`returnDist`	logical; whether to return a matrix with the distances between detected neighbors
`progress`	Display progress bar? (default ‘TRUE‘)

Value

If sparse=FALSE, returned object is a logical matrix with element [i,j] being TRUE when y[j,] is a neighbor of x[i]; if sparse=TRUE (the default), a sparse list representation of the same matrix is returned, with list element i a numeric vector with the indices j of neighboring features from y for the feature x[i,], or integer(0) in case there are no neighbors. If returnDists=TRUE the function returns a list, with the first element as specified above, and the second element the matrix of distances (in meters) between each pair of detected neighbors.

References

C. F. F. Karney, GeographicLib, Version 1.49 (2017-mm-dd), https://geographiclib.sourceforge.io/1.49

Examples

```
data(cities)
data(towns)

cities = st_transform(cities, 32636)
towns = st_transform(towns, 32636)

# Nearest town
st_nn(cities, towns)

# Using 'sfc' objects
st_nn(st_geometry(cities), st_geometry(towns))
st_nn(cities, st_geometry(towns))
st_nn(st_geometry(cities), towns)

# With distances
st_nn(cities, towns, returnDist = TRUE)

## Not run:

# Distance limit
st_nn(cities, towns, maxdist = 7200)
st_nn(cities, towns, k = 3, maxdist = 12000)
st_nn(cities, towns, k = 3, maxdist = 12000, returnDist = TRUE)

# 3 nearest towns
st_nn(cities, towns, k = 3)

# Spatial join
st_join(cities, towns, st_nn, k = 1)
st_join(cities, towns, st_nn, k = 2)
st_join(cities, towns, st_nn, k = 1, maxdist = 7200)
st_join(towns, cities, st_nn, k = 1)

# Polygons to polygons
st_nn(water, water, k = 4)
```

```
# Large example - Geo points
n = 1000
x = data.frame(
  lon = (runif(n) * 2 - 1) * 70,
  lat = (runif(n) * 2 - 1) * 70
)
x = st_as_sf(x, coords = c("lon", "lat"), crs = 4326)
start = Sys.time()
result = st_nn(x, x, k = 3)
end = Sys.time()
end - start

# Large example - Proj points
n = 1000
x = data.frame(
  lon = (runif(n) * 2 - 1) * 70,
  lat = (runif(n) * 2 - 1) * 70
)
x = st_as_sf(x, coords = c("lon", "lat"), crs = 4326)
x = st_transform(x, 32630)
start = Sys.time()
result = st_nn(x, x, k = 3)
end = Sys.time()
end - start

# Large example - Polygons
n = 1000
x = data.frame(
  lon = (runif(n) * 2 - 1) * 70,
  lat = (runif(n) * 2 - 1) * 70
)
x = st_as_sf(x, coords = c("lon", "lat"), crs = 4326)
x = st_transform(x, 32630)
x = st_buffer(x, 1000)
start = Sys.time()
result = st_nn(x, x, k = 3)
end = Sys.time()
end - start


## End(Not run)
```

`st_postgis` *Send 'sf' layer to a PostGIS query*

Description

The function sends a query plus an `sf` layer to PostGIS, saving the trouble of manually importing the layer and exporting the result

Usage

```
st_postgis(x, con, query, prefix = "temporary_nngeo_layer_")
```

Arguments

x	Object of class sf
con	Connection to PostgreSQL database with PostGIS extension enabled. Can be created using function RPostgreSQL::dbConnect
query	SQL query, which may refer to layer x as x and to the geometry column of the x layer as geom (see examples)
prefix	Prefix for storage of temporarily layer in the database

Value

Returned result from the database: an sf layer in case the result includes a geometry column, otherwise a data.frame

Examples

```
## Not run:

# Database connection and 'sf' layer
source("~/Dropbox/postgis_159.R")  ## Creates connection object 'con'
x = towns

# Query 1: Buffer
query = "SELECT ST_Buffer(geom, 0.1, 'quad_segs=2') AS geom FROM x LIMIT 5;"
st_postgis(x, con, query)

# Query 2: Extrusion
query = "SELECT ST_Extrude(geom, 0, 0, 30) AS geom FROM x LIMIT 5;"
st_postgis(x, con, query)

## End(Not run)
```

---

st_remove_holes *Remove polygon holes*

Description

The function removes all polygon holes and return the modified layer

Usage

```
st_remove_holes(x)
```

Arguments

x   Object of class `sf`, `sfc` or `sfg`, of type "POLYGON" or "MULTIPOLYGON"

Value

Object of same class as `x`, with holes removed

References

Following the StackOverflow answer by user 'lbusett' -

https://stackoverflow.com/questions/52654701/removing-holes-from-polygons-in-r-sf

Examples

```
opar = par(mfrow = c(1, 2))

# Example with 'sfg' - POLYGON
p1 = rbind(c(0,0), c(1,0), c(3,2), c(2,4), c(1,4), c(0,0))
p2 = rbind(c(1,1), c(1,2), c(2,2), c(1,1))
pol = st_polygon(list(p1, p2))
pol
result = st_remove_holes(pol)
result
plot(pol, col = "#FF000033", main = "Before")
plot(result, col = "#FF000033", main = "After")

# Example with 'sfg' - MULTIPOLYGON
p3 = rbind(c(3,0), c(4,0), c(4,1), c(3,1), c(3,0))
p4 = rbind(c(3.3,0.3), c(3.8,0.3), c(3.8,0.8), c(3.3,0.8), c(3.3,0.3))[5:1,]
p5 = rbind(c(3,3), c(4,2), c(4,3), c(3,3))
mpol = st_multipolygon(list(list(p1,p2), list(p3,p4), list(p5)))
mpol
result = st_remove_holes(mpol)
result
plot(mpol, col = "#FF000033", main = "Before")
plot(result, col = "#FF000033", main = "After")

# Example with 'sfc' - POLYGON
x = st_sfc(pol, pol * 0.75 + c(3.5, 2))
x
result = st_remove_holes(x)
result
plot(x, col = "#FF000033", main = "Before")
plot(result, col = "#FF000033", main = "After")

# Example with 'sfc' - MULTIPOLYGON
x = st_sfc(pol, mpol * 0.75 + c(3.5, 2))
x
result = st_remove_holes(x)
result
plot(x, col = "#FF000033", main = "Before")
```

```
plot(result, col = "#FF000033", main = "After")

par(opar)

# Example with 'sf'
x = st_sfc(pol, mpol * 0.75 + c(3.5, 2))
x = st_sf(geom = x, data.frame(id = 1:length(x)))
result = st_remove_holes(x)
result
plot(x, main = "Before")
plot(result, main = "After")
```

`st_segments` *Split polygons or lines to segments*

Description

Split lines or polygons to separate segments.

Usage

```
st_segments(x)
```

Arguments

x	An object of class `sfg`, `sfc` or `sf`, with geometry type `LINESTRING`, `MULTILINESTRING`, `POLYGON` or `MULTIPOLYGON`

Value

An `sf` layer of type `LINESTRING` where each segment is represented by a separate feature

Examples

```
# Sample geometries
s1 = rbind(c(0,3),c(0,4),c(1,5),c(2,5))
ls = st_linestring(s1)
s2 = rbind(c(0.2,3), c(0.2,4), c(1,4.8), c(2,4.8))
s3 = rbind(c(0,4.4), c(0.6,5))
mls = st_multilinestring(list(s1,s2,s3))
p1 = rbind(c(0,0), c(1,0), c(3,2), c(2,4), c(1,4), c(0,0))
p2 = rbind(c(1,1), c(1,2), c(2,2), c(1,1))
pol = st_polygon(list(p1,p2))
p3 = rbind(c(3,0), c(4,0), c(4,1), c(3,1), c(3,0))
p4 = rbind(c(3.3,0.3), c(3.8,0.3), c(3.8,0.8), c(3.3,0.8), c(3.3,0.3))[5:1,]
p5 = rbind(c(3,3), c(4,2), c(4,3), c(3,3))
mpol = st_multipolygon(list(list(p1,p2), list(p3,p4), list(p5)))

# Geometries ('sfg')
```

```
opar = par(mfrow = c(1, 2))

plot(ls)
seg = st_segments(ls)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

plot(mls)
seg = st_segments(mls)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

plot(pol)
seg = st_segments(pol)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

plot(mpol)
seg = st_segments(mpol)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

par(opar)

# Columns ('sfc')
opar = par(mfrow = c(1, 2))

ls = st_sfc(ls)
plot(ls)
seg = st_segments(ls)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

ls2 = st_sfc(c(ls, ls + c(1, -1), ls + c(-3, -1)))
plot(ls2)
seg = st_segments(ls2)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

mls = st_sfc(mls)
plot(mls)
seg = st_segments(mls)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

mls2 = st_sfc(c(mls, mls + c(1, -2)))
plot(mls2)
seg = st_segments(mls2)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

pol = st_sfc(pol)
plot(pol)
```

```
seg = st_segments(pol)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

mpol = st_sfc(mpol)
plot(mpol)
seg = st_segments(mpol)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

mpol2 = st_sfc(c(mpol, mpol + c(5, 2)))
plot(mpol2)
seg = st_segments(mpol2)
plot(seg, col = rainbow(length(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:length(seg)))

par(opar)

# Layers ('sf')
opar = par(mfrow = c(1, 2))

mpol_sf = st_sf(id = 1:2, type = c("a", "b"), geom = st_sfc(c(mpol, mpol + c(5, 2))))
plot(st_geometry(mpol_sf))
seg = st_segments(mpol_sf)
plot(st_geometry(seg), col = rainbow(nrow(seg)))
text(st_coordinates(st_centroid(seg)), as.character(1:nrow(seg)))

par(opar)
```

`towns` *Point layer of towns in Israel*

Description

A `sf` POINT layer of all towns in Israel whose name starts with the letter "A".

Usage

```
towns
```

Format

A `sf` POINT layer with 93 features and 1 attribute:

name Town name

`water` *Polygonal layer of water bodies in Israel*

Description

A `sf` POLYGON layer of the four large water bodies in Israel:

- Mediterranean Sea
- Red Sea
- Sea of Galilee
- Dead Sea

Usage

```
water
```

Format

A `sf` POLYGON layer with 4 features and 1 attribute:

name Water body name

Index