Introduction to package nngeo

Michael Dorman

2019-05-12

Contents

Introduction

Package purpose

This document introduces the `nngeo` package. The `nngeo` package includes functions for spatial join of laters based on *k-nearest neighbor* relation between features. The functions work with spatial layer object defined in package `sf`, namely classes `sfc` and `sf`.

Installation

GitHub version -

```
install.packages("devtools")
devtools::install_github("michaeldorman/nngeo")
```

Sample data

The `nngeo` package comes with three sample datasets -

- `cities`
- `towns`
- `water`

The `cities` layer is a **point** layer representing the location of the three largest cities in Israel.

```r
cities
#> Simple feature collection with 3 features and 1 field
#> geometry type:  POINT
#> dimension:      XY
#> bbox:           xmin: 34.78177 ymin: 31.76832 xmax: 35.21371 ymax: 32.79405
#> epsg (SRID):    4326
#> proj4string:    +proj=longlat +datum=WGS84 +no_defs
#>        name                 geometry
#> 1 Jerusalem POINT (35.21371 31.76832)
#> 2  Tel-Aviv  POINT (34.78177 32.0853)
#> 3     Haifa POINT (34.98957 32.79405)
```

The `towns` layer is another **point** layer, with the location of all towns in Israel whose name begins with the letter A.

```r
towns
#> Simple feature collection with 93 features and 1 field
#> geometry type:  POINT
#> dimension:      XY
#> bbox:           xmin: 34.3309 ymin: 30.96493 xmax: 35.83863 ymax: 33.17806
#> epsg (SRID):    4326
#> proj4string:    +proj=longlat +datum=WGS84 +no_defs
#> First 10 features:
#>                     geometry                    name
#> 1  POINT (35.54639 32.70683)                 ALUMMOT
#> 2  POINT (35.12573 31.65512)            ALLON SHEVUT
#> 3  POINT (35.18041 33.04801)                   AVDON
#> 4  POINT (35.48441 32.81265)                   ARBEL
#> 5   POINT (35.5824 32.66228) ASHDOT YA'AQOV(ME'UHAD)
#> 6  POINT (35.33804 32.85159)                  ARRABE
#> 7    POINT (35.25207 32.866)            ATSMON SEGEV
#> 8  POINT (35.22568 33.08865)                 ARAMSHA
#> 9   POINT (35.4369 31.67897)                  AVENAT
#> 10 POINT (34.90936 31.89039)                  AZARYA
```

The `water` layer is an example of a **polygonal** layer. This layer contains four polygons of water bodies in Israel.

```r
water
#> Simple feature collection with 4 features and 1 field
#> geometry type:  POLYGON
#> dimension:      XY
#> bbox:           xmin: 34.1388 ymin: 29.45338 xmax: 35.64979 ymax: 33.1164
#> epsg (SRID):    4326
#> proj4string:    +proj=longlat +datum=WGS84 +no_defs
#>                name                       geometry
#> 1           Red Sea POLYGON ((34.96428 29.54775...
#> 2 Mediterranean Sea POLYGON ((35.10533 33.07661...
#> 3          Dead Sea POLYGON ((35.54743 31.37881...
#> 4    Sea of Galilee POLYGON ((35.6014 32.89248,...
```

Figure 1 shows the spatial configuration of the `cities`, `towns` and `water` layers.

```r
plot(st_geometry(towns), col = NA)
plot(st_geometry(water), col = "lightblue", add = TRUE)
plot(st_geometry(towns), col = "grey", pch = 1, add = TRUE)
```

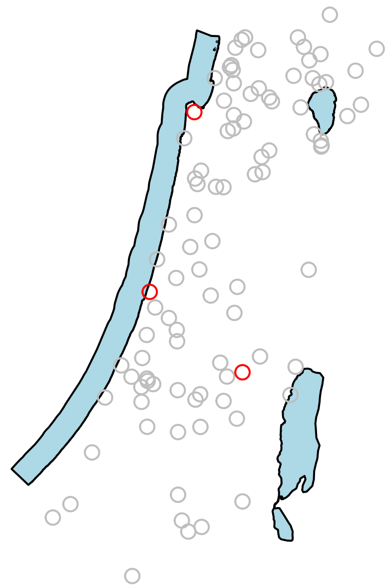

Figure 1: Visualization of the `water`, `towns` and `cities` layers

```
plot(st_geometry(cities), col = "red", pch = 1, add = TRUE)
```

Usage examples

The `st_nn` function

The main function in the `nngeo` package is `st_nn`.

The `st_nn` function accepts two layers, `x` and `y`, and returns a list with the same number of elements as `x` features. Each list element `i` is an integer vector with all indices `j` for which `x[i]` and `y[j]` are **nearest neighbors**.

For example, the following expression finds which feature in `towns[1:5, ]` is the nearest neighbor to each feature in `cities`.

```
nn = st_nn(cities, towns[1:5, ], progress = FALSE)
nn
#> [[1]]
#> [1] 2
#>
#> [[2]]
#> [1] 2
#>
#> [[3]]
#> [1] 3
```

This output tells us that `towns[2, ]` is the nearest among the five features of `towns[1:5, ]` to `cities[1, ]`, etc.

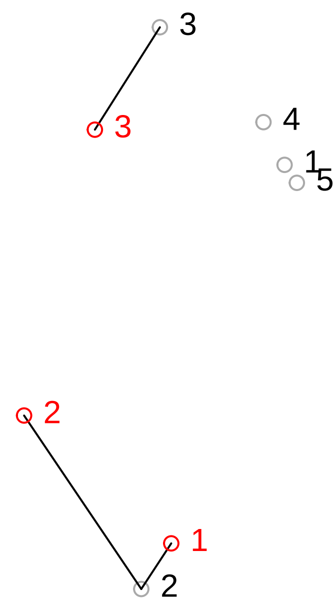

Figure 2: Nearest neighbor match between `cities` (in red) and `towns[1:5, ]` (in grey)

The `st_connect` function

The resulting nearest neighbor matches can be visualized using the `st_connect` function. This function builds a line layer connecting features from two layers `x` and `y` based on the relations defined in a list such the one returned by `st_nn` -

```
l = st_connect(cities, towns[1:5, ], ids = nn, progress = FALSE)
l
#> Geometry set for 3 features
#> geometry type:  LINESTRING
#> dimension:      XY
#> bbox:           xmin: 34.78177 ymin: 31.65512 xmax: 35.21371 ymax: 33.04801
#> epsg (SRID):    4326
#> proj4string:    +proj=longlat +datum=WGS84 +no_defs
#> LINESTRING (35.21371 31.76832, 35.12573 31.65512)
#> LINESTRING (34.78177 32.0853, 35.12573 31.65512)
#> LINESTRING (34.98957 32.79405, 35.18041 33.04801)
```

Plotting the line layer `l` gives a visual demonstration of the nearest neighbors match, as shown in Figure 2.

```
plot(st_geometry(towns[1:5, ]), col = "darkgrey")
plot(st_geometry(l), add = TRUE)
plot(st_geometry(cities), col = "red", add = TRUE)
text(
  st_coordinates(cities)[, 1],
  st_coordinates(cities)[, 2],
  1:3, col = "red", pos = 4
)
text(
  st_coordinates(towns[1:5, ])[, 1],
  st_coordinates(towns[1:5, ])[, 2],
  1:5, pos = 4
)
```

Dense matrix representation

The `st_nn` can also return the complete logical matrix indicating whether each feature in `x` is a neighbor of `y`. To get the dense matrix, instead of a list, use `sparse=FALSE`.

```
nn = st_nn(cities, towns[1:5, ], sparse = FALSE, progress = FALSE)
nn
#>       [,1]  [,2]  [,3]  [,4]  [,5]
#> [1,] FALSE  TRUE FALSE FALSE FALSE
#> [2,] FALSE  TRUE FALSE FALSE FALSE
#> [3,] FALSE FALSE  TRUE FALSE FALSE
```

k-Nearest neighbors where `k>0`

It is also possible to return any **k-nearest** neighbors, rather than just one. For example, setting `k=2` returns the **two** nearest neighbors -

```
nn = st_nn(cities, towns[1:5, ], k = 2, progress = FALSE)
nn
#> [[1]]
#> [1] 2 5
#>
#> [[2]]
#> [1] 2 5
#>
#> [[3]]
#> [1] 3 4

nn = st_nn(cities, towns[1:5, ], sparse = FALSE, k = 2, progress = FALSE)
nn
#>       [,1]  [,2]  [,3]  [,4]  [,5]
#> [1,] FALSE  TRUE FALSE FALSE  TRUE
#> [2,] FALSE  TRUE FALSE FALSE  TRUE
#> [3,] FALSE FALSE  TRUE  TRUE FALSE
```

Distance matrix

Using `returnDist=TRUE` the distances matrix is also returned, in addition the the neighbor matches, with both componenets now comprising a list -

```
nn = st_nn(
  cities, towns[1:5, ], sparse = FALSE, k = 2, returnDist = TRUE,
  progress = FALSE
)
nn
#> $nn
#>       [,1]  [,2]  [,3]  [,4]  [,5]
#> [1,] FALSE  TRUE FALSE FALSE  TRUE
#> [2,] FALSE  TRUE FALSE FALSE  TRUE
#> [3,] FALSE FALSE  TRUE  TRUE FALSE
#>
#> $dist
#>          [,1]      [,2]
```

```
#> [1,] 15069.49 105048.39
#> [2,] 57746.32  98846.89
#> [3,] 33345.18  46392.06
```

Search radius

Finally, the search for nearest neighbors can be limited to a **search radius** using `maxdist`. In the following example, the search radius is set to 50,000 meters (50 kilometers). Note that no neighbors are found within the search radius for `cities[2, ]`.

```
nn = st_nn(
  cities, towns[1:5, ], sparse = FALSE, k = 2, returnDist = TRUE, maxdist = 50000,
  progress = FALSE
)
nn
#> $nn
#>       [,1]  [,2]  [,3]  [,4]  [,5]
#> [1,] FALSE  TRUE FALSE FALSE FALSE
#> [2,] FALSE FALSE FALSE FALSE FALSE
#> [3,] FALSE FALSE  TRUE  TRUE FALSE
#>
#> $dist
#>          [,1]     [,2]
#> [1,] 15069.49       NA
#> [2,]       NA       NA
#> [3,] 33345.18 46392.06
```

Spatial join

The `st_nn` function can also be used as a **geometry predicate function** when performing spatial join with `sf::st_join`.

For example, the following expression spatially joins the two nearest `towns[1:5, ]` features to each `cities` features, using a search radius of 50 km.

```
cities1 = st_join(cities, towns[1:5, ], join = st_nn, k = 2, maxdist = 50000)
```

Here is the resulting layer -

```
cities1
#> Simple feature collection with 4 features and 2 fields
#> geometry type:  POINT
#> dimension:      XY
#> bbox:           xmin: 34.78177 ymin: 31.76832 xmax: 35.21371 ymax: 32.79405
#> epsg (SRID):    4326
#> proj4string:    +proj=longlat +datum=WGS84 +no_defs
#>        name.x       name.y                  geometry
#> 1   Jerusalem ALLON SHEVUT POINT (35.21371 31.76832)
#> 2    Tel-Aviv         <NA>  POINT (34.78177 32.0853)
#> 3       Haifa        AVDON POINT (34.98957 32.79405)
#> 3.1     Haifa        ARBEL POINT (34.98957 32.79405)
```

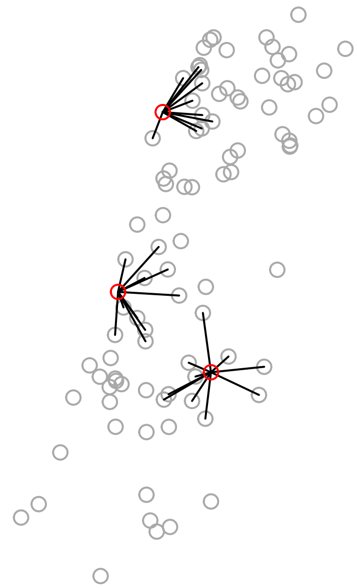

Figure 3: Nearest 10 `towns` features from each `cities` feature

Another example

Here is another example, finding the 10-nearest neighbor `towns` features for each `cities` feature.

```
x = st_nn(cities, towns, k = 10)
l = st_connect(cities, towns, ids = x)
```

The result is visualized in Figure 3.

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

Polygons

Nearest neighbor search also works for non-point layers. The following code section finds the 20-nearest `towns` features for each water body in `water[-1, ]`.

```
nn = st_nn(water[-1, ], towns, k = 20, progress = FALSE)
```

Again, we can calculate the respective lines for the above result using `st_connect`. Since one of the inputs is line/polygon, we need to specify a sampling distance `dist`, which sets the resolution of connecting points on the shape exterior boundary.

```
l = st_connect(water[-1, ], towns, ids = nn, progress = FALSE, dist = 100)
```

The result is visualized on Figure 4.

```
plot(st_geometry(water[-1, ]), col = "lightblue", border = "grey")
plot(st_geometry(towns), col = "darkgrey", add = TRUE)
plot(st_geometry(l), col = "red", add = TRUE)
```

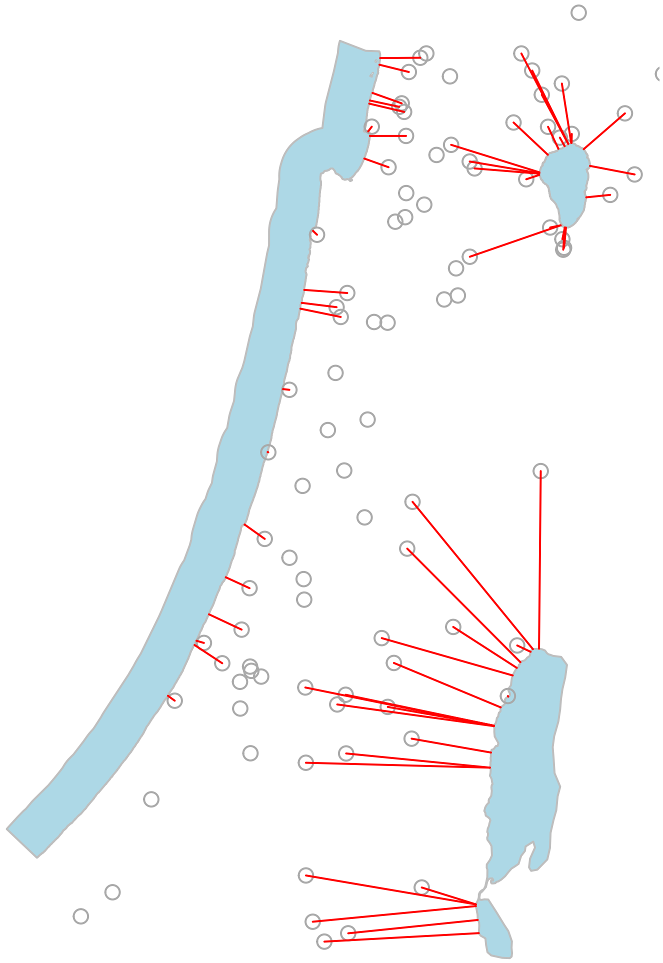

Figure 4: Nearest 20 `towns` features from each `water` polygon