Introduction to package **nngeo**

Michael Dorman

2020-08-11

**Contents**

Introduction ........................................... 1
  Package purpose ........................................ 1
  Installation ........................................... 1
  Sample data ............................................ 1

Usage examples ........................................ 3
  The **st_nn** function .................................. 3
  The **st_connect** function ............................ 3
  Dense matrix representation .......................... 4
  k-Nearest neighbors where \( k > 0 \) .................. 5
  Distance matrix ........................................ 5
  Search radius .......................................... 6
  Spatial join ............................................. 6
  Another example ....................................... 6

Polygons .................................................. 7

**Introduction**

**Package purpose**

This document introduces the **nngeo** package. The **nngeo** package includes functions for spatial join of layers 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:

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

**Sample data**

The **nngeo** package comes with three sample datasets:

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

```r
#> 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
#> geographic CRS: WGS 84
#> 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 large towns in Israel, compiled from a different data source:

```r
#> Simple feature collection with 193 features and 4 fields
#> geometry type: POINT
#> dimension: XY
#> bbox: xmin: 34.27 ymin: 29.56 xmax: 35.6 ymax: 33.21
#> geographic CRS: WGS 84
#> name country.etc pop capital geometry
#> 12 'Afula Israel 39151 0 POINT (35.29 32.62)
#> 17 'Akko Israel 45606 0 POINT (35.08 32.94)
#> 40 'Ar'ara Israel 15841 0 POINT (35.1 32.49)
#> 41 'Arad Israel 22757 0 POINT (35.22 31.26)
#> 43 'Arrabe Israel 20316 0 POINT (35.33 32.85)
#> 52 'Atlit Israel 4686 0 POINT (34.93 32.68)
#> 103 Eilabun Israel 4296 0 POINT (35.4 32.83)
#> 104 'Ein Mahel Israel 11014 0 POINT (35.35 32.72)
#> 105 'Ein Qiniyye Israel 2101 0 POINT (35.15 31.93)
#> 112 'Itlul Israel 6536 0 POINT (35.25 32.72)
```

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

```r
#> 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
#> geographic CRS: WGS 84
#> 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(water), col = "lightblue")
plot(st_geometry(towns), col = "grey", pch = 1, add = TRUE)
plot(st_geometry(cities), col = "red", pch = 1, add = TRUE)
```
Usage examples

The \texttt{st\_nn} function

The main function in the \texttt{nngeo} package is \texttt{st\_nn}.

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

For example, the following expression finds which feature in \texttt{towns[1:5, ]} is the nearest neighbor to each feature in \texttt{cities}:

\begin{verbatim}
  nn = st.nn(cities, towns[1:5, ], progress = FALSE)
  #> lon-lat points
  nn
  #> [[1]]
  #> [1] 4
  #>
  #> [[2]]
  #> [1] 3
  #>
  #> [[3]]
  #> [1] 2
\end{verbatim}

This output tells us that \texttt{towns[4, ]} is the nearest among the five features of \texttt{towns[1:5, ]} to \texttt{cities[1, ]}, etc.

The \texttt{st\_connect} function

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

```r
l = st_connect(cities, towns[1:5, ], ids = nn)
```

```
#> Calculating nearest IDs
#> Calculating lines
l
#> Geometry set for 3 features
#> geometry type: LINESTRING
#> dimension: XY
#> bbox: xmin: 34.78177 ymin: 31.26 xmax: 35.22 ymax: 32.94
#> geographic CRS: WGS 84
#> LINESTRING (35.21371 31.76832, 35.22 31.26)
#> LINESTRING (34.78177 32.0853, 35.1 32.49)
#> LINESTRING (34.98957 32.79405, 35.08 32.94)
```

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

```r
plot(st_geometry(towns[1:5, ]), col = "darkgrey")
plot(st_geometry(l), add = TRUE)
plot(st_geometry(cities), col = "red", add = TRUE)
```

```
   2
  / |
 / 1
```

```
   3
  / 
 2
```

```
  1
 / 
 4
```

```
1
2
3
4
5
```

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

### 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`.

```r
nn = st_nn(cities, towns[1:5, ], sparse = FALSE, progress = FALSE)
```

```
#> lon-lat points
nn
#> [1,] FALSE FALSE FALSE  TRUE FALSE
```

4
k-Nearest neighbors where \( k > 0 \)

It is also possible to return any \texttt{k-nearest} neighbors, rather than just one. For example, setting \( k=2 \) returns both the 1\textsuperscript{st} and 2\textsuperscript{nd} nearest neighbors:

\begin{verbatim}
nn = st_nn(cities, towns[1:5, ], k = 2, progress = FALSE)
nn
[[1]]
[1] 4 3

[[2]]
[1] 3 1

[[3]]
[1] 2 5
\end{verbatim}

\begin{verbatim}
nn = st_nn(cities, towns[1:5, ], sparse = FALSE, k = 2, progress = FALSE)
nn
[1,] FALSE FALSE TRUE TRUE FALSE
[2,] TRUE FALSE TRUE FALSE FALSE
[3,] FALSE TRUE FALSE FALSE TRUE
\end{verbatim}

Distance matrix

Using \texttt{returnDist=TRUE} the distances \texttt{list} is also returned, in addition the the neighbor matches, with both components now comprising a \texttt{list}:

\begin{verbatim}
nn = st_nn(cities, towns[1:5, ], k = 2, returnDist = TRUE, progress = FALSE)
nn
$nn
$nn[[1]]
[1] 4 3

$nn[[2]]
[1] 3 1

$nn[[3]]
[1] 2 5

$dist
$dist[[1]]
[1] 56364.74 80742.62

$dist[[2]]
[1] 53968.63 76186.87
\end{verbatim}
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, ]:

```r
nn = st_nn(cities, towns[1:5, ], k = 2, maxdist = 50000, progress = FALSE)
```

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:

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

Another example

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

```r
x = st_nn(cities, towns, k = 10)
l = st_connect(cities, towns, ids = x)
```
The result is visualized in Figure 3.

```r
plot(st_geometry(1))
plot(st_geometry(cities), col = "red", add = TRUE)
plot(st_geometry(towns), col = "darkgrey", add = TRUE)
```

Figure 3: Nearest 10 towns features from each cities feature

### 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, ].

```r
nn = st_nn(water[-1, ], towns, k = 20, progress = FALSE)
#> lines or polygons
```

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.

```r
l = st_connect(water[-1, ], towns, ids = nn, dist = 100)
#> Calculating nearest IDs
#> Calculating lines
```

The result is visualized in Figure 4.

```r
plot(st_geometry(water[-1, ], col = "lightblue", border = "grey")
plot(st_geometry(towns), col = "darkgrey", add = TRUE)
plot(st_geometry(1), col = "red", add = TRUE)
```
Figure 4: Nearest 20 towns features from each water polygon