# Package ‘donut’

November 10, 2021

**Title**  Nearest Neighbour Search with Variables on a Torus  
**Version**  1.0.2  
**Date**  2021-10-10  
**Description**  Finds the k nearest neighbours in a dataset of specified points, adding the option to wrap certain variables on a torus. The user chooses the algorithm to use to find the nearest neighbours. Two such algorithms, provided by the packages 'RANN' <https://cran.r-project.org/package=RANN>, and 'nabor' <https://cran.r-project.org/package=nabor>, are suggested.

**Imports**  graphics  
**License**  GPL (>= 2)  
**Encoding**  UTF-8  
**Depends**  R (>= 3.3.0)  
**RoxygenNote**  7.1.1  
**Suggests**  knitr, nabor, RANN, rmarkdown, testthat (>= 2.1.0)  
**VignetteBuilder**  knitr  
**BugReports**  https://github.com/paulnorthrop/donut/issues  
**NeedsCompilation**  no  
**Author**  Paul J. Northrop [aut, cre, cph]  
**Maintainer**  Paul J. Northrop <p.northrop@ucl.ac.uk>  
**Repository**  CRAN  
**Date/Publication**  2021-11-10 17:50:02 UTC

## R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>donut</td>
<td>2</td>
</tr>
<tr>
<td>nnt</td>
<td>2</td>
</tr>
<tr>
<td>plot.nnt</td>
<td>5</td>
</tr>
</tbody>
</table>

Index  7
**donut**

*donut: Nearest Neighbour Search with Variables on a Torus*

**Description**

Finds the k nearest neighbours in a dataset of specified points, adding the option to wrap certain variables on a torus. The user chooses the algorithm to use to find the nearest neighbours.

**Details**

The function `nnt` performs the nearest neighbour search. There is also a rudimentary plot method: `plot.nnt`.

The default algorithm is that provided by the function `nn2` in the `RANN-package`. Another possibility is the `knn` function in the `nabor-package`.

See vignette("donut-vignette",package = "donut") for an overview of the package.

**References**


**See Also**

`nnt` for nearest neighbour with some variables wrapped on a torus.

`plot.nnt` plot method for objects returned from `nnt` (1 and 2 dimensional data only).

---

**nnt**

*Nearest Neighbour Search with Variables on a Torus*

**Description**

Uses a user-supplied function to find the k nearest neighbours of specified points in a dataset, adding the option to wrap certain variables on a torus.
Usage

```r
nnt(
  data,
  query = data,
  k = min(10, nrow(data)),
  fn = RANN::nn2,
  torus,
  ranges,
  method = 1,
  ...
)
```

Arguments

- **data**: An $M$ by $d$ numeric matrix or data frame. Each of the $M$ rows contains a $d$-dimensional observation.
- **query**: An $N$ by $d$ numeric matrix or data frame. Each row contains an $d$-dimensional point that will be queried against `data`.
- **k**: An integer scalar. The number of nearest neighbours, of the points in the rows of `query`, to find.
- **fn**: The function with which to calculate the nearest neighbours. The syntax of this function must be `fn(data, query, k, ...)`. The default is `RANN::nn2`. Another possibility is `nabor::knn`.
- **torus**: An integer vector with element(s) in $\{1, \ldots, ncol(data)\}$. The corresponding variables are wrapped on the respective range given in `ranges`.
- **ranges**: A `length(torus)` by 2 numeric matrix. Row $i$ gives the range of variation of the variable indexed by `torus[i]`. `ranges[i, 1]` and `ranges[i, 2]` are equivalent values of the variable, such as 0 degrees and 360 degrees. If `length(torus)` = 1 then `ranges` may be a vector of length 2.
- **method**: An integer scalar, equal to 1 or 2. See Details.
- **...**: Further arguments to be passed to `fn`.

Details

If `method = 1` then the data are partially replicated, arranged around the original data in a way that wraps the variables in `torus` on their respective ranges in `ranges`. Then `fn` is called using this replicated dataset as the argument `data`. If `k` is large and/or `data` is a sparse dataset then it is possible that a single observation contributes more than once to a set of nearest neighbours, which is incorrect. If this occurs then `nnt` uses method 2 to correct the offending rows in `nn.idx` and `nn.dists` in the returned list object.

If `method = 2` then the following approach is used for the point in each row in `query`. The data indexed by `torus` are shifted (and wrapped) so that the point is located at the respective midpoints of ranges. Method 2 is efficient only if the number of points in `query` is small.

If `torus` is missing then `fn` is called using `fn(data = data, query = query, k = k, ...)`, so that a call to `nnt` is equivalent to a call to the function chosen by `fn`. 
Value

An object (a list) of class c("nnt", "donut") containing the following components.

- `nn.idx`: An $N$ by $d$ integer matrix of the $k$ nearest neighbour indices, i.e. the rows of data.
- `nn.dists`: An $N$ by $d$ numeric matrix of the $k$ nearest neighbour distances.
- `data`, `query`, `k`, `fn`: The arguments `data`, `query`, `k` and `fn` (in fact substitute(fn)).
- `torus`, `ranges`, `method`: If `torus` is supplied, the arguments `torus`, `ranges` and `method`.
- `call`: The call to `spm`.

References


See Also

RANN::nn2, nabor::knn: nearest neighbour searches.

plot.nnt plot method for objects returned from nnt (1 and 2 dimensional data only).

Examples

got_RANN <- requireNamespace("RANN", quietly = TRUE)
got_nabor <- requireNamespace("nabor", quietly = TRUE)

set.seed(20092019)
# 2D example from the RANN:nn2 documentation (L2 metric)
x1 <- runif(100, 0, 2 * pi)
x2 <- runif(100, 0, 3)
DATA <- data.frame(x1, x2)
if (got_RANN) {
  nearest <- nnt(DATA, DATA)
}

# Suppose that x1 should be wrapped
ranges1 <- c(0, 2 * pi)
query1 <- rbind(c(6, 1.3), c(2 * pi, 3), c(3, 1.5), c(4, 0))
if (got_RANN) {
  res1 <- nnt(DATA, query1, k = 8, torus = 1, ranges = ranges1)
  plot(res1, ylim = c(0, 3))
}

# Suppose that x1 and x2 should be wrapped
ranges2 <- rbind(c(0, 2 * pi), c(0, 3))
query2 <- rbind(c(6, 1.3), c(2 * pi, 3), c(3, 1.5), c(4, 0))
if (got_RANN) {
  res2 <- nnt(DATA, query2, k = 8, torus = 1:2, ranges = ranges2)
  plot(res2)
}

# Use nabor::knn (L2 metric) instead of RANN::nn2
if (got_nabor) {
  res3 <- nnt(DATA, query2, k = 8, fn = nabor::knn, torus = 1:2,
               ranges = ranges2)
  plot(res3)
}

# 1D example
ranges <- c(0, 2 * pi)
query <- c(4, 0.1)
if (got_RANN) {
  res <- nnt(x1, query, torus = 1, ranges = ranges, method = 1)
  plot(res)
}

---

**plot.nnt**  
*Plot diagnostics for an nnt object*

**Description**

plot method for an object of class "nnt".

**Usage**

```r
## S3 method for class 'nnt'
plot(x, ...)
```

**Arguments**

- `x`: an object of class "nnt", a result of a call to `nnt`.
- `...`: Further arguments to be passed to `plot`, or `points`.

**Details**

This function is only applicable in 1 or 2 dimensions, that is, when `ncol(x$data) = 1` or `2`. It provides a visual check that the wrapping of variables is working as intended, in cases where the number of query points, that is, `nrow(x$query)` is small enough that sets of nearest neighbours do not overlap much.

If `ncol(x$data) = 1` then the index of each observation is plotted against its value, using a plotting character `pch = 1`. A vertical line is superimposed at each value in `x$query` and the `x$k` nearest neighbours of each line are colour-coded.
If `ncol(x$data) = 2` then `x$data[,2]` is plotted against `x$data[,1]`, using a plotting character `pch = 1`. Each point in `x$query` is plotted with a cross and the `x$k` nearest neighbours of each point are colour-coded.

Colours of the lines/crosses and nearest neighbour points can be set by an argument `col`. If a variable is wrapped then the default plotting limits are set using the corresponding values in `x$ranges`.

**Value**

Nothing is returned.

**Examples**

See the examples in `nnt`.

**See Also**

`nnt` for nearest neighbour with some variables wrapped on a torus.
Index

donut, 2

knn, 2

nabor::knn, 4
nn2, 2
nnt, 2, 2, 4–6

plot, 5
plot.nnt, 2, 4, 5
points, 5

RANN::nn2, 4