Package ‘distances’
March 18, 2019

Type Package
Title Tools for Distance Metrics
Version 0.1.7
Date 2019-03-18
Description Provides tools for constructing, manipulating and using distance metrics.
Depends R (>= 3.4.0)
Imports stats
Suggests testthat
NeedsCompilation yes
License GPL (>= 3)
LicenseNote The distances package includes the ANN library (distributed under the LGPLv2.1 license).
URL https://github.com/fsavje/distances
BugReports https://github.com/fsavje/distances/issues
Encoding UTF-8
RoxygenNote 6.1.1
Author Fredrik Savje [aut, cre]
Maintainer Fredrik Savje <fredrik.savje@yale.edu>
Repository CRAN
Date/Publication 2019-03-18 06:13:28 UTC

R topics documented:

distances-package .................................................. 2
distances ............................................................ 2
distance_columns .................................................... 5
distance_matrix ...................................................... 5
is.distances .......................................................... 6
max_distance_search ................................................ 6
nearest_neighbor_search .......................................... 7
distances-package

distances: Tools for Distance Metrics

Description

The distances package provides tools for constructing, manipulating and using distance metrics in R. It calculates distances only as needed (unlike the standard `dist` function which derives the complete distance matrix when called). This saves memory and can increase speed. The package also includes functions for fast nearest and farthest neighbor searching.

Details

See the package’s website for more information: https://github.com/fsavje/distances.

Bug reports and suggestions are greatly appreciated. They are best reported here: https://github.com/fsavje/distances/issues/new.

distances

Constructor for distance metric objects

Description

distances constructs a distance metric for a set of points. Currently, it only creates Euclidean distances. It can, however, create distances in any linear projection of Euclidean space. In other words, Mahalanobis distances or normalized Euclidean distances are both possible. It is also possible to give each dimension of the space different weights.

Usage

distances(data, id_variable = NULL, dist_variables = NULL,
normalize = NULL, weights = NULL)

Arguments

data a matrix or data frame containing the data points between distances should be derived.
id_variable optional IDs of the data points. If id_variable is a single string and data is a data frame, the corresponding column in data will be taken as IDs. That column will be excluded from data when constructing distances (unless it is listed in dist_variables). If id_variable is NULL, the IDs are set to 1:nrow(data). Otherwise, id_variable must be of length nrow(data) and will be used directly as IDs.
distances

**dist variables**
optional names of the columns in data that should be used when constructing distances. If dist variables is NULL, all columns will be used (net of eventual column specified by id_variable). If data is a matrix, dist variables must be NULL.

**normalize**
optional normalization of the data prior to distance construction. If normalize is NULL or "none", no normalization will be done (effectively setting normalize to the identity matrix). If normalize is "mahalanobize", normalization will be done with var(data) (i.e., resulting in Mahalanobis distances). If normalize is "studentize", normalization is done with the diagonal of var(data). If normalize is a matrix, it will be used in the normalization. If normalize is a vector, a diagonal matrix with the supplied vector as its diagonal will be used. The matrix used for normalization must be positive-semidefinite.

**weights**
optional weighting of the data prior to distance construction. If weights is NULL no weighting will be done (effectively setting weights to the identity matrix). If weights is a matrix, that will be used in the weighting. If weights is a vector, a diagonal matrix with the supplied vector as its diagonal will be used. The matrix used for weighting must be positive-semidefinite.

**Details**
Let \( x \) and \( y \) be two data points in data described by two vectors. distances uses the following metric to derive the distance between \( x \) and \( y \):

\[
\sqrt{(x - y)N^{-0.5}W(N^{-0.5})'(x - y)}
\]

where \( N^{-0.5} \) is the Cholesky decomposition (lower triangular) of the inverse of the matrix specified by normalize, and \( W \) is the matrix specified by weights.

When normalize is var(data) (i.e., using the "mahalanobize" option), the function gives (weighted) Mahalanobis distances. When normalize is diag(var(data)) (i.e., using the "studentize" option), the function divides each column by its variance leading to (weighted) normalized Euclidean distances. If normalize is the identity matrix (i.e., using the "none" or NULL option), the function derives ordinary Euclidean distances.

**Value**
Returns a distances object.

**Examples**
```
my_data_points <- data.frame(x = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
                           y = c(10, 9, 8, 7, 6, 6, 7, 8, 9, 10))

# Euclidean distances
my_distances1 <- distances(my_data_points)

# Euclidean distances in only one dimension
my_distances2 <- distances(my_data_points, dist_variables = "x")
```
# Mahalanobis distances
my_distances3 <- distances(my_data_points, 
    normalize = "mahalanobize")

# Custom normalization matrix
my_norm_mat <- matrix(c(3, 1, 1, 3), nrow = 2)
my_distances4 <- distances(my_data_points, 
    normalize = my_norm_mat)

# Give "x" twice the weight compared to "y"
my_distances5 <- distances(my_data_points, 
    weights = c(2, 1))

# Use normalization and weighting
my_distances6 <- distances(my_data_points, 
    normalize = "mahalanobize", 
    weights = c(2, 1))

# Custom ID labels
my_data_points_withID <- data.frame(my_data_points, 
    my_ids = letters[1:10])
my_distances7 <- distances(my_data_points_withID, 
    id_variable = "my_ids")

# Compare to standard R functions
all.equal(as.matrix(my_distances1), as.matrix(dist(my_data_points)))
# TRUE

all.equal(as.matrix(my_distances2), as.matrix(dist(my_data_points[, "x"])))
# TRUE

tmp_distances <- sqrt(mahalanobis(as.matrix(my_data_points), 
    unlist(my_data_points[1, ]), 
    var(my_data_points)))
names(tmp_distances) <- 1:10
all.equal(as.matrix(my_distances3)[1, ], tmp_distances)
# TRUE

tmp_data_points <- as.matrix(my_data_points)
tmp_data_points[, 1] <- sqrt(2) * tmp_data_points[, 1]
all.equal(as.matrix(my_distances5), as.matrix(dist(tmp_data_points)))
# TRUE

tmp_data_points <- as.matrix(my_data_points)
tmp_cov_mat <- var(tmp_data_points)
tmp_data_points[, 1] <- sqrt(2) * tmp_data_points[, 1]
tmp_distances <- sqrt(mahalanobis(tmp_data_points, 
    tmp_data_points[1, ], 
    tmp_cov_mat))
names(tmp_distances) <- 1:10
## distance_columns

Distance matrix columns

### Description

distance_columns extracts columns from the distance matrix.

### Usage

distance_columns(distances, column_indices, row_indices = NULL)

### Arguments

- **distances**: A distances object.
- **column_indices**: An integer vector with point indices indicating which columns to be extracted.
- **row_indices**: If NULL, complete rows will be extracted. If integer vector with point indices, only the indicated rows will be extracted.

### Details

If the complete distance matrix is desired, distance_matrix is faster than distance_columns.

### Value

Returns a matrix with the requested columns.

## distance_matrix

Distance matrix

### Description

distance_matrix makes distance matrices (complete and partial) from distances objects.

### Usage

distance_matrix(distances, indices = NULL)
**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>distances</code></td>
<td>A <code>distances</code> object.</td>
</tr>
<tr>
<td><code>indices</code></td>
<td>If <code>NULL</code>, the complete distance matrix is made. If integer vector with point indices, a partial matrix including only the indicated data points is made.</td>
</tr>
</tbody>
</table>

**Value**

Returns a distance matrix of class `dist`.

---

**is.distances**

*Check distances object*

**Description**

`is.distances` checks whether the provided object is a valid instance of the `distances` class.

**Usage**

`is.distances(x)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>object to check.</td>
</tr>
</tbody>
</table>

**Value**

Returns `TRUE` if `x` is a valid `distances` object, otherwise `FALSE`.

---

**max_distance_search**

*Max distance search*

**Description**

`max_distance_search` searches for the data point furthest from a set of query points.

**Usage**

`max_distance_search(distances, query_indices = NULL, search_indices = NULL)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>distances</code></td>
<td>A <code>distances</code> object.</td>
</tr>
<tr>
<td><code>query_indices</code></td>
<td>An integer vector with point indices to query. If <code>NULL</code>, all data points in distances are queried.</td>
</tr>
<tr>
<td><code>search_indices</code></td>
<td>An integer vector with point indices to search among. If <code>NULL</code>, all data points in distances are searched over.</td>
</tr>
</tbody>
</table>
nearest_neighbor_search

**Value**

An integer vector with point indices for the data point furthest from each query.

---

**Description**

nearest_neighbor_search searches for the k nearest neighbors of a set of query points.

**Usage**

```r
nearest_neighbor_search(distances, k, query_indices = NULL, 
                        search_indices = NULL, radius = NULL)
```

**Arguments**

- **distances**: A `distances` object.
- **k**: The number of neighbors to search for.
- **query_indices**: An integer vector with point indices to query. If NULL, all data points in `distances` are queried.
- **search_indices**: An integer vector with point indices to search among. If NULL, all data points in `distances` are searched over.
- **radius**: Restrict the search to a fixed radius around each query. If fewer than k search points exist within this radius, no neighbors are reported (indicated by `NA`).

**Value**

A matrix with point indices for the nearest neighbors. Columns in this matrix indicate queries, and rows are ordered by distances from the query.
Index

dist, 2, 6
distance_columns, 5
distance_matrix, 5, 5
distances, 2, 5–7
distances-package, 2

is.distances, 6

max_distance_search, 6

nearest_neighbor_search, 7