Package ‘emstreeR’

May 8, 2019

Type Package

Title Tools for Fast Computing and Plotting Euclidean Minimum Spanning Trees

Version 2.2.0

Date 2019-05-07

Description Computes Euclidean Minimum Spanning Trees (EMST) using the fast Dual-Tree Boruvka algorithm (March, Ram, Gray, 2010, <doi:10.1145/1835804.1835882>) implemented in 'mlpack' - the C++ Machine Learning library (Curtin et al., 2013). 'emstreeR' heavily relies on 'RcppMLPACK' and 'Rcpp', working as a wrapper to the C++ fast EMST algorithm. Thus, R users do not have to deal with the R-'Rcpp'-C++ integration. The package also provides functions and an S3 method for readily plotting Minimum Spanning Trees (MST) using either 'base' R, 'scatterplot3d' or 'ggplot2' style.

License BSD_3_clause + file LICENSE

Encoding UTF-8

Imports Rcpp (>= 0.12.18), scatterplot3d, ggplot2, BBmisc

Depends R (>= 3.5.0)

LinkingTo Rcpp, RcppMLPACK, RcppArmadillo, BH

BugReports https://github.com/allanvc/emstreeR/issues

SystemRequirements C++11 compiler.

RoxygenNote 6.1.1

NeedsCompilation yes

Author Allan Quadros [aut, cre],
Andre Cancado [ctb]

Maintainer Allan Quadros <allanvcq@gmail.com>

Repository CRAN

Date/Publication 2019-05-08 04:50:03 UTC

R topics documented:

`emstreeR-package` *Euclidean Minimum Spanning Tree*

Description

The **emstreeR** package allows R users to fast and easily compute an Euclidean Minimum Spanning Tree from data.

Introduction

This package relies on RcppMLPACK to provide an R interface to the Dual-Tree Boruvka algorithm (March, Ram, Gray, 2010) implemented in 'mlpack' - the C++ Machine Learning Library (Curtin et. al., 2013). The Dual-Tree Boruvka is theoretically and emipirically the fastest algorithm for computing an Euclidean Minimum Spanning Tree (EMST).

Computing the Minimum Spanning Tree

`ComputeMST` is the main function of this package. It is a fast wrapper to its `C++` homonym from 'mlpack' for computing an Euclidean Minimum Spanning Tree. Compared to functions in other MST related R packages, `ComputeMST` is easier to use because you can pass your data as a `numeric matrix` or a `data.frame`, which are the most common data input formats in the wild. You do not have to put it into a graph format as you otherwise would in other packages.

Plotting

'emstreeR' also provides wrapper functions and an S3 method for plotting the resulting MST from `ComputeMST`.

- `plot.MST` is an S3 method to the generic function `plot` and produces 2D scatter plots with segments between the points in a 'base' R style, following the linkage order in the MST.
- `plotMST3D` produces a 3D point cloud with segments between the points, following the linkage order in the MST and using the 'scatterplot3d' package style for plotting.
- `stat_MST` is a 'ggplot2' Stat extension which produces 2D scatter plots with segments based on the linkage order in the MST using the 'ggplot2' style.

Author(s)

Author & Mantainer: Allan Quadros <`allanvcq@gmail.com`>

Contributors:

- Andre Cancado <`cancado@unb.br`>

References

March, W. B., and Ram, P., and Gray, A. G. (2010). *Fast euclidian minimum spanning tree: algorithm analysis, and applications*. 16th ACM SIGKDD International Conference on Knowledge Discovery and Data mining, July 25-28 2010. Washington, DC, USA. doi:10.1145/1835804.1835882.

Curtin, R. R. et al. (2013). Mlpack: A scalable C++ machine learning library. *Journal of Machine Learning Research*, v. 14, 2013.

See Also

Useful links:

- `mlpack`: `http://www.mlpack.org/`

`ComputeMST` *Euclidean Minimum Spanning Tree*

Description

Computes an Euclidean Minimum Spanning Tree (EMST) from the data. `ComputeMST` is a wrapper around the homonym function in the 'mlpack' library.

Usage

```
ComputeMST(x, verbose = TRUE, scale = FALSE)
```

Arguments

`x` a `numeric matrix` or `data.frame`.

`verbose` If `TRUE`, mutes the output from the C++ code.

`scale` If `TRUE`, it will scale your data with `scale` before computing the the minimum spanning tree and the distances to be presented will refer to the scaled data.

Details

Before the computation, ComputeMST runs some checks and transformations (if needed) on the provided data using the data_check function. After the computation, it returns the 'cleaned' data plus 3 columns: `from`, `to`, and `distance`. Those columns show each pair of start and end points, and the distance between them, forming the Minimum Spanning Tree (MST).

Value

an object of class MST and data.frame.

Note

It is worth noting that the afore mentioned columns (from, to, and distance) have no relationship with their respective row in the output MST/data.frame object. The authors chose the data.frame format for the output rather than a list because it is more suitable for plotting the MST with the new 'ggplot2' Stat (stat_MST) provided with this package. The last row of the output at these three columns will always be the same: 1 1 0.0000000. This is because we always have n-1 edges for n points. Hence, this is done to 'complete' the data.frame that is returned.

Examples

```
## artifical data
set.seed(1984)
n <- 15
c1 <- data.frame(x = rnorm(n, -0.2, sd = 0.2), y = rnorm(n, -2, sd = 0.2))
c2 <- data.frame(x = rnorm(n, -1.1, sd = 0.15), y = rnorm(n, -2, sd = 0.3))
d <- rbind(c1, c2)
d <- as.data.frame(d)

## MST:
out <- ComputeMST(d)
out
```

data_check *Data checking*

Description

Checks the integrity of the inputed data before computing the Euclidian Minimum Spanning Tree (EMST)

Usage

```
data_check(x)
```

Arguments

x	a matrix or data.frame.

Details

data_check is called from inside ComputeMST before the computation begins. First, it evaluates the object format. Afterwards, it checks whether the inputed data has at least two columns and tries to coerce all columns into numeric, beyond removing all rows containing NA's entries.

Value

a `matrix` containing the cleaned data after running the necessary checks.

`mlpack_mst` *`mlpack`'s Euclidean Minimum Spanning Tree*

Description

Fast computes an EMST using the `mlpack C++` library.

Usage

```
mlpack_mst(data)
```

Arguments

`data`	a `matrix`.

Value

a `matrix` containing each pair of start and end points on its columns, and the distance between these points in order to produce the Minimum Spanning Tree.

`plot.MST` *Plot method for 'MST' objects*

Description

Plots a 2D Minimum Spanning Tree (MST) by producing a scatter plot with segments using the generic function `plot`.

Usage

```
## S3 method for class 'MST'
plot(x, ..., V1 = 1, V2 = 2, col.pts = "black",
  col.segts = "black", lty = 3)
```

Arguments

`x` a `MST` class object returned by the `ComputeMST` function.

`...` further graphical parameters.

`V1` the numeric position or the name of the column to be used as the x coordinates of the points in the plot.

`V2` the numeric position or the name of the column to be used as the y coordinates of the points in the plot.

`col.pts` color of the points (vertices/nodes) in the plot.

`col.segts` color of the segments (edges) in the plot.

`lty` line type. An integer or name: `0 = "blank"`, `1 = "solid"`, `2 = "dashed"`, `3 = "dotted"`, `4 = "dotdash"`, `5 = "longdash"`, `6 = "twodash"`. The default for `'MST'` objects is `"dotted"`.

Examples

```
## 2D artifical data
set.seed(1984)
n <- 15
c1 <- data.frame(x = rnorm(n, -0.2, sd = 0.2), y = rnorm(n, -2, sd = 0.2))
c2 <- data.frame(x = rnorm(n, -1.1, sd = 0.15), y = rnorm(n, -2, sd = 0.3))
c3 <- c(0.55, -2.4)
d <- rbind(c1, c2, c3)
d <- as.data.frame(d)

## MST:
out <- ComputeMST(d)
out

## 2D plot:
plot(out)

# using different parameters
plot(out, col.pts = "blue", col.segts = "red", lty = 2)
```

`plotMST3D` *3D Minimum Spanning Tree Plot*

Description

Plots a 3D MST by producing a point cloud with segments as a ’scatterplot3d’ graphic.

Usage

```
plotMST3D(tree, x = 1, y = 2, z = 3, col.pts = "black",
  col.segts = "black", angle = 40, ...)
```

Arguments

`tree` a `MST` class object returned by the `ComputeMST()` function.

`x` the numeric position or the name of the column to be used as the x coordinates of points in the plot.

`y` the numeric position or the name of the column to be used as the y coordinates of points in the plot.

`z` the numeric position or the name of the column to be used as the z coordinates of points in the plot.

`col.pts` color of points (vertices/nodes) in the plot.

`col.segts` color of segments (edges) in the plot.

`angle` angle between x and y axis (Attention: result depends on scaling).

`...` further graphical parameters.

Examples

```
## 3D artificial data:
n1 = 12
n2 = 22
n3 = 7
n = n1 + n2 + n3
set.seed(1984)

mean_vector <- sample(seq(1, 10, by = 2), 3)
sd_vector <- sample(seq(0.01, 0.8, by = 0.01), 3)
c1 <- matrix(rnorm(n1*3, mean = mean_vector[1], sd = .3), n1, 3)
c2 <- matrix(rnorm(n2*3, mean = mean_vector[2], sd = .5), n2, 3)
c3 <- matrix(rnorm(n3*3, mean = mean_vector[3], sd = 1), n3, 3)
d<-rbind(c1, c2, c3)

## MST:
out <- ComputeMST(d)

## 3D PLOT:
plotMST3D(out)
```

`stat_MST` *Euclidean Minimum Spanning Tree Stat Function*

Description

A Stat extension for 'ggplot2' to plot a 2D MST by making a scatter plot with segments.

`stat_MST` uses the information returned by `ComputeMST` for producing a 2D Minimum Spanning Tree plot with 'ggplot2' and should be combined with `geom_point()`.

Usage

```
stat_MST(mapping = NULL, data = NULL, geom = "segment",
  position = "identity", na.rm = FALSE, linetype = "dotted",
  show.legend = NA, inherit.aes = TRUE, ...)
```

Arguments

`mapping` The aesthetic mapping, usually constructed with `aes` or `aes_`. The required aesthetics are `x`, `y`, `from`, and `to`. Those are columns of the `mst` object returned by `ComputeMST`.

`data` a `mst` class object returned by the `ComputeMST` function.

`geom` The geometric object to display the data. The default value is `"segment"` in order to produce the edges between the vertices.

`position` The position adjustment to use for overlapping points on this layer

`na.rm` a logical value indicating whether `NA` values should be stripped before the computation proceeds.

`linetype` an integer or name: `0 = "blank"`, `1 = "solid"`, `2 = "dashed"`, `3 = "dotted"`, `4 = "dotdash"`, `5 = "longdash"`, `6 = "twodash"`. The default for `'MST'` objects is "dotted".

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders`.

`...` other arguments passed on to `layer`. This can include aesthetics whose values you want to set, not map. See `layer` for more details.

Computed variables

x x coordinates of the MST start points

y y coordinates of the MST start points

xend x coordinates of the MST end points

yend y coordinates of the MST end points

Examples

```
## 2D artificial data:
set.seed(1984)
n <- 15
c1 <- data.frame(x = rnorm(n, -0.2, sd = 0.2), y = rnorm(n, -2, sd = 0.2))
c2 <- data.frame(x = rnorm(n, -1.1, sd = 0.15), y = rnorm(n, -2, sd = 0.3))
d <- rbind(c1, c2)
d <- as.data.frame(d)

## MST:
```

```
out <- ComputeMST(d)

#1) simple plot
library(ggplot2)
ggplot(data = out,
    aes(x = x, y = y,
    from = from, to = to))+
    geom_point()+
    stat_MST(colour = "red", linetype = 2)

#2) curved edges
library(ggplot2)
ggplot(data = out,
    aes(x = x, y = y,
    from = from, to = to))+
    geom_point()+
    stat_MST(geom = "curve", colour = "red", linetype = 2)

## Not run:
## plotting MST on maps:
library(ggmap)

#3) honeymoon cruise example
# define ports
df.port_locations <- data.frame(location = c("Civitavecchia, Italy",
                                             "Genova, Italy",
                                             "Marseille, France",
                                             "Barcelona, Spain",
                                             "Tunis, Tunisia",
                                             "Palermo, Italy"),
                                stringsAsFactors = FALSE)

# get latitude and longitude
geo.port_locations <- geocode(df.port_locations$location, source = "dsk")

# combine data
df.port_locations <- cbind(df.port_locations, geo.port_locations)

# MST
out <- ComputeMST(df.port_locations[,2:3])
plot(out) #just to check

# Plot
#' map <- c(left = -8, bottom = 32, right = 20, top = 47)

get_stamenmap(map, zoom = 5) %>% ggmap()+
  stat_MST(data = out,
           aes(x = lon, y = lat, from = from, to = to),
           colour = "red", linetype = 2)+
  geom_point(data = out, aes(x = lon, y = lat), size = 3)


#4) World Map travels:
```

```
library(ggplot2)
library(ggmaps)

country_coords_txt <- ”
   1     3.00000  28.00000       Algeria
   2    54.00000  24.00000           UAE
   3   139.75309  35.68536         Japan
   4    45.00000  25.00000 'Saudi Arabia'
   5     9.00000  34.00000       Tunisia
   6     5.75000  52.50000   Netherlands
   7   103.80000   1.36667     Singapore
   8   124.10000  -8.36667         Korea
   9    -2.69531  54.75844            UK
   10    34.91155  39.05901        Turkey
   11  -113.64258  60.10867        Canada
   12    77.00000  20.00000         India
   13    25.00000  46.00000       Romania
   14   135.00000 -25.00000     Australia
   15    10.00000  62.00000        Norway”


 d <- read.delim(text = country_coords_txt, header = FALSE,
   quote = ”'”, sep = ””, col.names = c('id', 'lon', 'lat', 'name'))

 out <- ComputeMST(d[,2:3])

 country_shapes <- geom_polygon(aes(x = long, y = lat, group = group),
   data = map_data('world'), fill = ”#CECECE”, color = ”#515151”,
   size = 0.15)

 ggplot()+ country_shapes+
   stat_MST(geomdata = out, aes(x = lon, y = lat, from = from, to = to),
     colour = ”red”, linetype = 2)+
   geom_point(data = out, aes(x = lon, y = lat), size=2)

## End(Not run)
```

Index