

Package ‘lumbermark’

October 1, 2026

Type Package

Title Resistant Clustering via Chopping Up Mutual Reachability Minimum Spanning Trees

Version 0.9.1

Date 2026-10-01

Description Implements a fast and resistant divisive clustering algorithm which identifies a specified number of clusters: 'lumbermark' iteratively chops off sizeable limbs that are joined by protruding segments of a dataset's mutual reachability minimum spanning tree (Gagolewski, 2026 <[DOI:10.48550/arXiv.2604.07143](https://doi.org/10.48550/arXiv.2604.07143)>). The use of a mutual reachability distance pulls peripheral points farther away from each other. It is a viable alternative to the 'HDBSCAN*' algorithm and can be viewed as a divisive version of Genie. The resulting partitions of different granularities are properly nested. When combined with the 'deadwood' package, it can act as an outlier detector. The 'Python' version of 'lumbermark' is available via 'PyPI'.

BugReports <https://github.com/gagolews/lumbermark/issues>

URL <https://lumbermark.gagolewski.com/>,
<https://clustering-benchmarks.gagolewski.com/>,
<https://github.com/gagolews/lumbermark>

License AGPL-3

Imports Rcpp, deadwood

Suggests datasets,

LinkingTo Rcpp

Encoding UTF-8

SystemRequirements OpenMP

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Marek Gagolewski [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0003-0637-6028>>)

Maintainer Marek Gagolewski <marek@gagolewski.com>

Repository CRAN

Date/Publication 2026-10-01 15:40:33 UTC

Contents

lumbermark	2
Index	6

lumbermark	<i>Lumbermark: Fast and Robust Clustering</i>
------------	---

Description

Lumbermark is a fast and robust divisive clustering algorithm which identifies a specified number of clusters.

It iteratively chops off sizeable limbs that are joined by protruding segments of a dataset's mutual reachability minimum spanning tree.

The use of a mutual reachability distance ($M > 1$; Campello et al., 2013) pulls peripheral points farther away from each other.

Usage

```
lumbermark(d, ...)

## Default S3 method:
lumbermark(
  d,
  k,
  min_cluster_factor = 0.25,
  M = 5L,
  nested = TRUE,
  skip_leaves = (M > 0L),
  min_cluster_size = 10,
  distance = c("euclidean", "l2", "manhattan", "cityblock", "l1", "cosine"),
  verbose = FALSE,
  ...
)

## S3 method for class 'dist'
lumbermark(
  d,
  k,
  min_cluster_factor = 0.25,
  M = 5L,
```

```

    nested = TRUE,
    skip_leaves = (M > 0L),
    min_cluster_size = 10,
    verbose = FALSE,
    ...
)

## S3 method for class 'mst'
lumbermark(
  d,
  k,
  min_cluster_factor = 0.25,
  nested = TRUE,
  skip_leaves = TRUE,
  min_cluster_size = 10,
  verbose = FALSE,
  ...
)
```

Arguments

<code>d</code>	a numeric matrix (or an object coercible to one, e.g., a data frame with numeric-like columns) or an object of class <code>dist</code> (see dist), or an object of class <code>mst</code> (see mst)
<code>...</code>	further arguments passed to mst()
<code>k</code>	integer; the desired number of clusters to detect
<code>min_cluster_factor</code>	numeric value in (0,1); output cluster sizes will not be smaller than <code>min_cluster_factor*n/k</code> (excluding leaves if <code>skip_leaves</code> is requested)
<code>M</code>	integer; smoothing factor; $M \leq 1$ gives the selected distance; otherwise, the mutual reachability distance is used
<code>nested</code>	if TRUE, generating l clusters in the $(l-1)$ -th iteration uses the <code>min_cluster_factor*n/l</code> size bound; this way, the consecutively generated partitions are not dependent on the initial <code>k</code> and are thus properly nested. On the other hand, if <code>nested</code> is FALSE, then all clusters are of size at least <code>min_cluster_factor*n/k</code> ; this is somewhat faster to compute.
<code>skip_leaves</code>	logical; whether the MST leaves should be omitted from cluster size counting
<code>min_cluster_size</code>	integer; minimal cluster size; used as a safeguard for small <code>n</code> and large <code>k</code>
<code>distance</code>	metric used to compute the linkage, one of: "euclidean" (synonym: "l2"), "manhattan" (a.k.a. "l1" and "cityblock"), "cosine"
<code>verbose</code>	logical; whether to print diagnostic messages and progress information

Details

Lumbermark is a viable alternative to the HDBSCAN* algorithm that is able to detect a predefined number of clusters and indicate outliers (via **deadwood**; see Gagolewski, 2026).

As with all distance-based methods (this includes k-means and DBSCAN as well), applying data preprocessing and feature engineering techniques (e.g., feature scaling, feature selection, dimensionality reduction) might lead to more meaningful results.

If `d` is a numeric matrix or an object of class `dist`, `mst()` will be called to compute an MST, which generally takes at most $O(n^2)$ time. However, by default, a faster algorithm based on K-d trees is selected automatically for low-dimensional Euclidean spaces; see `mst_euclid` from the **quitefastmst** package.

Once a minimum spanning tree with increasingly sorted edge weights is determined, the Lumbermark algorithm runs in $O(kn)$ time. If you want to test different parameters or *ks*, it is best to compute the MST explicitly beforehand.

Value

`lumbermark()` returns an object of class `mstclust`, which defines a *k*-partition of a given dataset, i.e., a vector whose *i*-th element denotes the *i*-th input point's cluster label between 1 and *k*.

The `mst` attribute gives the computed minimum spanning tree which can be reused in further calls to the functions from **genieclust**, **lumbermark**, and **deadwood**.

The `cut_edges` attribute gives the *k* − 1 indexes of the MST edges whose omission leads to the requested *k*-partition (connected components of the resulting spanning forest).

Author(s)

Marek Gagolewski

References

M. Gagolewski, Lumbermark: Resistant clustering by chopping up mutual reachability minimum spanning trees, 2026, doi:10.48550/arXiv.2604.07143

R.J.G.B. Campello, D. Moulavi, J. Sander, Density-based clustering based on hierarchical density estimates, *Lecture Notes in Computer Science* 7819, 2013, 160-172, doi:10.1007/978364237456-2_14

M. Gagolewski, A. Cena, M. Bartoszek, Ł. Brzozowski, Clustering with minimum spanning trees: How good can it be?, *Journal of Classification* 42, 2025, 90-112, doi:10.1007/s00357024094831

M. Gagolewski, *deadwood*, in preparation, 2026

M. Gagolewski, *quitefastmst*, in preparation, 2026

See Also

The official online manual of **lumbermark** at <https://lumbermark.gagolewski.com/mst/> for the minimum spanning tree routines

Examples

```
library("datasets")
data("iris")
X <- jitter(as.matrix(iris[3:4]))
y_pred <- lumbermark(X, k=3)
```

```
y_test <- as.integer(iris[,5])
plot(X, col=y_pred, pch=y_test, asp=1, las=1)

# detect three clusters and find outliers with Deadwood
library("deadwood")
y_pred2 <- lumbermark(X, k=3)
plot(X, col=y_pred2, asp=1, las=1)
is_outlier <- deadwood(y_pred2)
points(X[!is_outlier, ], col=y_pred2[!is_outlier], pch=16)
```

Index

dist, [3](#)

lumbermark, [2](#)

mst, [3](#), [4](#)

mst_euclid, [4](#)