

# Package ‘netboost’

September 19, 2024

**Type** Package

**Title** Network Analysis Supported by Boosting

**Version** 2.13.1

**Date** 2024-06-12

**Description** Boosting supported network analysis for high-dimensional omics applications.  
This package comes bundled with the MC-UPGMA clustering package by Yaniv Loewenstein.

**Depends** R (>= 4.0.0)

**biocViews** Software, StatisticalMethod, GraphAndNetwork, Network,  
Clustering, DimensionReduction, BiomedicalInformatics,  
Epigenetics, Metabolomics, Transcriptomics

**Imports** Rcpp, RcppParallel, parallel, grDevices, graphics, stats,  
utils, dynamicTreeCut, WGCNA, impute, colorspace, methods,  
BiocStyle, R.utils

**LinkingTo** Rcpp, RcppParallel

**Suggests** knitr, rmarkdown

**VignetteBuilder** knitr

**SystemRequirements** GNU make, Bash, Perl, Gzip

**OS\_type** unix

**ByteCompile** yes

**NeedsCompilation** yes

**LazyData** true

**License** GPL-3

**URL** <https://bioconductor.org/packages/release/bioc/html/netboost.html>

**BugReports** pascal.schlosser@uniklinik-freiburg.de

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**git\_url** <https://git.bioconductor.org/packages/netboost>

**git\_branch** devel

**git\_last\_commit** 09d03ed

**git\_last\_commit\_date** 2024-06-12

**Repository** Bioconductor 3.20

**Date/Publication** 2024-09-18

**Author** Pascal Schlosser [aut, cre],  
 Jochen Knaus [aut, ctb],  
 Yaniv Loewenstein [aut]

**Maintainer** Pascal Schlosser <pascal.schlosser@uniklinik-freiburg.de>

## Contents

.onAttach	2
calculate_adjacency	3
cpp_dist_tom	4
cpp_filter_base	4
cpp_filter_end	5
cpp_filter_step	5
cpp_tree_search	6
cut_dendro	6
cut_trees	7
mcupgma_exec	8
nb_clust	9
nb_dist	10
nb_filter	11
nb_mcupgma	12
nb_moduleEigengenes	13
nb_plot_dendro	15
nb_summary	16
nb_transfer	17
netboost	18
netboostMCUPGMAPath	19
netboostPackagePath	20
netboostTmpCleanup	20
netboostTmpPath	21
tcga_aml_meth_rna_chr18	21
tree_dendro	22
tree_search	22

**Index** **24**

---

.onAttach	<i>Package startup: used to fetch installation path of the own package, as required for executing binary programs delivered with it.</i>
-----------	--

---

## Description

Package startup: used to fetch installation path of the own package, as required for executing binary programs delivered with it.

## Usage

```
.onAttach(libname, pkgname)
```

**Arguments**

libname	Path to R installation (base package dir)
pkpname	Package name (should be "netboost")

**Value**

none

**Examples**

```
## Not run: nb_example()
```

---

calculate_adjacency	<i>Calculate network adjacencies for filter</i>
---------------------	---

---

**Description**

Calculate network adjacencies for filter

**Usage**

```
calculate_adjacency(  
  datan,  
  filter,  
  soft_power = 2,  
  method = c("pearson", "kendall", "spearman"),  
  cores = getOption("mc.cores", 2L)  
)
```

**Arguments**

datan	Data frame where rows correspond to samples and columns to features.
filter	Filter-Matrix as generated by the nb_filter function.
soft_power	Integer. Exponent of the transformation. Set automatically based on the scale free topology criterion if unspecified.
method	A character string specifying the method to be used for correlation coefficients.
cores	Integer. Amount of CPU cores used (<=1 : sequential).

**Value**

Vector with adjacencies for the filter

cpp\_dist\_tom            *Function to calculate distance*

---

**Description**

Function to calculate distance

**Usage**

```
cpp_dist_tom(filter, adjacency)
```

**Arguments**

filter	Filter matrix
adjacency	Vector

**Details**

Steps: 1. - Sequential preparation of index and partner caches per value in filter 2. - Parallel calculation of the distances with cached vectors

**Value**

numeric vector

---

cpp\_filter\_base            *Initialise boosting with chosen accelerator hardware (x86, AVX, FMA)*

---

**Description**

Initialise boosting with chosen accelerator hardware (x86, AVX, FMA)

**Usage**

```
cpp_filter_base(data, stepno = 20L, mode_ = 2L)
```

**Arguments**

data	Matrix
stepno	Amount of steps
mode_	Accelerator mode (0: x86, 1: FMA, 2: AVX)

**Value**

none

---

cpp_filter_end	<i>Boosting cleanup (required to free memory)</i>
----------------	---

---

**Description**

Boosting cleanup (required to free memory)

**Usage**

```
cpp_filter_end()
```

**Value**

none

---

cpp_filter_step	<i>Single boosting step</i>
-----------------	-----------------------------

---

**Description**

Single boosting step

**Usage**

```
cpp_filter_step(col_y)
```

**Arguments**

col_y	Row in data matrix
-------	--------------------

**Details**

Must be initialised before using @seefilter\_base

**Value**

integer vector

---

cpp\_tree\_search      *Tree search.*

---

### Description

Constraint: IDs  $0 \leq x$  (Integer)

### Usage

```
cpp_tree_search(netboost_forest)
```

### Arguments

netboost\_forest  
Input-matrix (4 columns, ids in column 0,1,3)

### Value

List

---

cut\_dendro      *Module detection for an individual tree*

---

### Description

Module detection for an individual tree

### Usage

```
cut_dendro(
  tree_dendro,
  min_cluster_size = 2L,
  datan,
  ME_diss_thres,
  name_of_tree = "",
  qc_plot = TRUE,
  n_pc = 1,
  robust_PCs = FALSE,
  nb_min_varExpl = 0.5,
  method = c("pearson", "kendall", "spearman")
)
```

### Arguments

tree\_dendro      List of tree specific objects including dendrogram, tree data and features originating from the tree\_dendro function.

min\_cluster\_size  
Integer. The minimum number of features in one module.

datan              Data frame where rows correspond to samples and columns to features.

ME_diss_thres	Numeric. Module Eigengene Dissimilarity Threshold for merging close modules.
name_of_tree	String. Annotating plots and messages.
qc_plot	Logical. Should plots be created?
n_pc	Number of principal components and variance explained entries to be calculated. The number of returned variance explained entries is currently 'min(n_pc,10)'. If given 'n_pc' is greater than 10, a warning is issued.
robust_PCs	Should PCA be calculated on ranked data (Spearman PCA)? Rotations will not correspond to original data if this is applied.
nb_min_varExpl	Minimum proportion of variance explained for returned module eigengenes. The number of PCs is capped at n_pc.
method	A character string specifying the method to be used for correlation coefficients.

**Value**

List

---

cut_trees	<i>Module detection for the results from a nb_mcupgma call</i>
-----------	--

---

**Description**

Module detection for the results from a nb\_mcupgma call

**Usage**

```
cut_trees(
  trees,
  datan,
  forest,
  min_cluster_size = 2L,
  ME_diss_thres,
  qc_plot = TRUE,
  n_pc = 1,
  robust_PCs = FALSE,
  nb_min_varExpl = 0.5,
  method = c("pearson", "kendall", "spearman")
)
```

**Arguments**

trees	List of trees, where one tree is a list of ids and rows
datan	Data frame where rows correspond to samples and columns to features.
forest	Raw dendrogram-matrix as generated by the nb_mcupgma function.
min_cluster_size	Integer. The minimum number of features in one module.
ME_diss_thres	Numeric. Module Eigengene Dissimilarity Threshold for merging close modules.

qc_plot	Logical. Should plots be created?
n_pc	Number of principal components and variance explained entries to be calculated. The number of returned variance explained entries is currently 'min(n_pc,10)'. If given 'n_pc' is greater than 10, a warning is issued.
robust_PCs	Should PCA be calculated on ranked data (Spearman PCA)? Rotations will not correspond to original data if this is applied.
nb_min_varExpl	Minimum proportion of variance explained for returned module eigengenes. The number of PCs is capped at n_pc.
method	A character string specifying the method to be used for correlation coefficients.

**Value**

List

**Examples**

```
data('tcga_aml_meth_rna_chr18', package='netboost')
cores <- as.integer(getOption('mc.cores', 2))
datan <- as.data.frame(scale(tcga_aml_meth_rna_chr18, center=TRUE,
scale=TRUE))
filter <- nb_filter(datan=datan, stepno=20L, until=0L, progress=1000L,
cores=cores,mode=2L)
dist <- nb_dist(datan=datan, filter=filter, soft_power=3L, cores=cores)
max_singleton = dim(tcga_aml_meth_rna_chr18)[2]
forest <- nb_mcupgma(filter=filter, dist=dist, max_singleton=max_singleton,
cores=cores)
trees <- tree_search(forest)
results <- cut_trees(trees=trees,datan=datan, forest=forest,
min_cluster_size=10L, ME_diss_thres=0.25, qc_plot=TRUE)
```

---

mcupgma\_exec

---

*Execute a program/script from the installed MCUPGMA suite.*


---

**Description**

Execute a program/script from the installed MCUPGMA suite.

**Usage**

```
mcupgma_exec(exec = NULL, ..., console = TRUE)
```

**Arguments**

exec	Name of the file of the executable.
...	Arguments passed to mcupgma executable in order required by program
console	Logical. Print output to R console or fetch for return to caller

**Value**

console=TRUE: exit code (0: no error). console=FALSE: STDOUT/STDERR output

**Examples**

```
mcupgma_exec(exec="cluster.pl", "--help")
```

---

 nb\_clust

*Netboost clustering step*


---

**Description**

Netboost clustering step

**Usage**

```
nb_clust(
  filter,
  dist,
  datan,
  max_singleton = dim(datan)[2],
  min_cluster_size = 2L,
  ME_diss_thres = 0.25,
  cores = getOption("mc.cores", 2L),
  qc_plot = TRUE,
  n_pc = 1,
  robust_PCs = FALSE,
  nb_min_varExpl = 0.5,
  method = c("pearson", "kendall", "spearman")
)
```

**Arguments**

filter	Filter-Matrix as generated by the nb_filter function.
dist	Distance-Matrix as generated by the nb_dist function.
datan	Data frame where rows correspond to samples and columns to features.
max_singleton	Integer. The maximal singleton in the clustering. Usually equals the number of features.
min_cluster_size	Integer. The minimum number of features in one module.
ME_diss_thres	Numeric. Module Eigengene Dissimilarity Threshold for merging close modules.
cores	Integer. Amount of CPU cores used (<=1 : sequential)
qc_plot	Logical. Create plot.
n_pc	Number of principal components and variance explained entries to be calculated. The number of returned variance explained entries is currently 'min(n_pc,10)'. If given 'n_pc' is greater than 10, a warning is issued.
robust_PCs	Should PCA be calculated on ranked data (Spearman PCA)? Rotations will not correspond to original data if this is applied.
nb_min_varExpl	Minimum proportion of variance explained for returned module eigengenes. The number of PCs is capped at n_pc.
method	A character string specifying the method to be used for correlation coefficients.

**Value**

List

**Examples**

```

data('tcga_aml_meth_rna_chr18', package='netboost')
cores <- as.integer(getOption('mc.cores', 2))
datan <- as.data.frame(scale(tcga_aml_meth_rna_chr18, center=TRUE,
scale=TRUE))
filter <- nb_filter(datan=datan, stepno=20L, until=0L, progress=1000L,
cores=cores,mode=2L)
dist <- nb_dist(datan=datan, filter=filter, soft_power=3L, cores=cores)
max_singleton = dim(tcga_aml_meth_rna_chr18)[2]
pdf("test.pdf",width=30)
sum_res <- nb_clust(filter=filter, dist=dist, datan=datan,
max_singleton=max_singleton, min_cluster_size=10L, ME_diss_thres=0.25,
cores=cores, qc_plot=TRUE, n_pc=2L, nb_min_varExpl=0.5)
dev.off()

```

nb\_dist

*Calculate distance (external wrapper for internal C++ function) Parallelisation inside C++ program with RcppParallel.*

**Description**

Calculate distance (external wrapper for internal C++ function) Parallelisation inside C++ program with RcppParallel.

**Usage**

```

nb_dist(
  filter,
  datan,
  soft_power = 2,
  cores = getOption("mc.cores", 2L),
  verbose = getOption("verbose"),
  method = c("pearson", "kendall", "spearman")
)

```

**Arguments**

filter	Filter-Matrix as generated by the nb_filter function.
datan	Data frame were rows correspond to samples and columns to features.
soft_power	Integer. Exponent of the transformation. Set automatically based on the scale free topology criterion if unspecified.
cores	Integer. Amount of CPU cores used (<=1 : sequential).
verbose	Additional diagnostic messages.
method	A character string specifying the method to be used for correlation coefficients.

**Value**

Vector with distances.

**Examples**

```
data('tcga_aml_meth_rna_chr18', package='netboost')
cores <- as.integer(getOption('mc.cores', 2))
datan <- as.data.frame(scale(tcga_aml_meth_rna_chr18, center=TRUE,
scale=TRUE))
filter <- nb_filter(datan=datan, stepno=20L, until=0L, progress=1000L,
cores=cores,mode=2L)
dist <- nb_dist(datan=datan, filter=filter, soft_power=3L, cores=cores)
summary(dist)
```

---

nb_filter	<i>Boosting via C++ function. Parallelisation by R-package parallel with forking (overhead of this method does not fall into account as single steps are ~10s).</i>
-----------	---

---

**Description**

Parallelisation via multicore (via 'parallel'-package). So \*nix only atm.

**Usage**

```
nb_filter(
  datan,
  stepno = 20L,
  until = 0L,
  progress = 1000L,
  filter_method = c("spearman", "skip", "kendall", "boosting", "pearson"),
  cores = getOption("mc.cores", 2L),
  mode = 2L,
  verbose = getOption("verbose")
)
```

**Arguments**

datan	Data frame where rows correspond to samples and columns to features.
stepno	Integer amount of boosting steps
until	Stop at index/column (if 0: iterate through all columns)
progress	Integer. If > 0, print progress after every X steps (mind: parallel!)
filter_method	The following filtering methods are supported: "boosting" (non-zero coefficients in likelihood based boosting), "skip" (no filter), "kendall" (stats::cor.test), "spearman" (stats::cor.test), "pearson" (stats::cor.test)
cores	Integer. Amount of CPU cores used (<=1 : sequential)
mode	Integer. Mode (0: x86, 1: FMA, 2: AVX). Features are only available if compiled accordingly and available on the hardware.
verbose	Additional diagnostic messages.

**Value**

matrix n times 2 matrix with the indices of the n unique entries of the filter

**Examples**

```
data('tcga_aml_meth_rna_chr18', package='netboost')
cores <- as.integer(getOption('mc.cores', 2))
datan <- as.data.frame(scale(tcga_aml_meth_rna_chr18, center=TRUE,
scale=TRUE))
filter <- nb_filter(datan=datan, stepno=20L, until=0L, progress=1000L,
cores=cores,mode=2L)
head(filter)
nrow(filter)/(ncol(datan)*(ncol(datan)-1)/2) # proportion of potential undirected edges
```

---

nb_mcupgma	<i>Calculate dendrogram for a sparse distance matrix (external wrapper MC-UPGMA clustering package Loewenstein et al.</i>
------------	---

---

**Description**

Calculate dendrogram for a sparse distance matrix (external wrapper MC-UPGMA clustering package Loewenstein et al.

**Usage**

```
nb_mcupgma(
  filter,
  dist,
  max_singleton,
  cores = getOption("mc.cores", 2L),
  verbose = getOption("verbose")
)
```

**Arguments**

filter	Filter-Matrix as generated by the nb_filter function.
dist	Distance-Matrix as generated by the nb_dist function.
max_singleton	Integer The maximal singleton in the clustering. Usually equals the number of features.
cores	Integer Amount of CPU cores used (<=1 : sequential)
verbose	Logical Additional diagnostic messages.

**Value**

Raw dendrogram to be processed by tree\_search and tree\_dendro.

**Examples**

```

data('tcga_aml_meth_rna_chr18', package='netboost')
cores <- as.integer(getOption('mc.cores', 2))
datan <- as.data.frame(scale(tcga_aml_meth_rna_chr18,
center=TRUE, scale=TRUE))
filter <- nb_filter(datan=datan, stepno=20L, until=0L,
                    progress=1000L, cores=cores, mode=2L)
dist <- nb_dist(datan=datan, filter=filter, soft_power=3L, cores=cores)
max_singleton = dim(tcga_aml_meth_rna_chr18)[2]
forest <- nb_mcupgma(filter=filter, dist=dist,
                    max_singleton=max_singleton, cores=cores)
head(forest)

```

---

nb\_moduleEigengenes     *Netboost module aggregate extraction.*

---

**Description**

This is a modification of `WGCNA::moduleEigengenes()` (version `WGCNA_1.66`) to include more than the first principal component. For details see `WGCNA::moduleEigengenes()`.

**Usage**

```

nb_moduleEigengenes(
  expr,
  colors,
  n_pc = 1,
  align = "along average",
  exclude_grey = FALSE,
  grey = if (is.numeric(colors)) 0 else "grey",
  subHubs = TRUE,
  robust = FALSE,
  trapErrors = FALSE,
  return_valid_only = trapErrors,
  soft_power = 6,
  scale = TRUE,
  verbose = 0,
  indent = 0,
  nb_min_varExpl = 0.5
)

```

**Arguments**

expr	Expression data for a single set in the form of a data frame where rows are samples and columns are genes (probes).
colors	A vector of the same length as the number of probes in 'expr', giving module color for all probes (genes). Color "grey" is reserved for unassigned genes. Expression
n_pc	Number of principal components and variance explained entries to be calculated. The number of returned variance explained entries is currently 'min(n_pc,10)'. If given 'n_pc' is greater than 10, a warning is issued.

align	Controls whether eigengenes, whose orientation is undetermined, should be aligned with average expression ('align = 'along average', the default) or left as they are ('align = '). Any other value will trigger an error.
exclude_grey	Should the improper module consisting of 'grey' genes be excluded from the eigengenes?
grey	Value of 'colors' designating the improper module. Note that if 'colors' is a factor of numbers, the default value will be incorrect.
subHubs	Controls whether hub genes should be substituted for missing eigengenes. If 'TRUE', each missing eigengene (i.e., eigengene whose calculation failed and the error was trapped) will be replaced by a weighted average of the most connected hub genes in the corresponding module. If this calculation fails, or if 'subHubs==FALSE', the value of 'trapErrors' will determine whether the offending module will be removed or whether the function will issue an error and stop.
robust	Should PCA be calculated on ranked data (Spearman PCA)? Rotations will not correspond to original data if this is applied.
trapErrors	Controls handling of errors from that may arise when there are too many 'NA' entries in expression data. If 'TRUE', errors from calling these functions will be trapped without abnormal exit. If 'FALSE', errors will cause the function to stop. Note, however, that 'subHubs' takes precedence in the sense that if 'subHubs==TRUE' and 'trapErrors==FALSE', an error will be issued only if both the principal component and the hubgene calculations have failed.
return_valid_only	logical; controls whether the returned data frame of module eigengenes contains columns corresponding only to modules whose eigengenes or hub genes could be calculated correctly ('TRUE'), or whether the data frame should have columns for each of the input color labels ('FALSE').
soft_power	The power used in soft-thresholding the adjacency matrix. Only used when the hubgene approximation is necessary because the principal component calculation failed. It must be non-negative. The default value should only be changed if there is a clear indication that it leads to incorrect results.
scale	logical; can be used to turn off scaling of the expression data before calculating the singular value decomposition. The scaling should only be turned off if the data has been scaled previously, in which case the function can run a bit faster. Note however that the function first imputes, then scales the expression data in each module. If the expression contain missing data, scaling outside of the function and letting the function impute missing data may lead to slightly different results than if the data is scaled within the function.
verbose	Controls verbosity of printed progress messages. 0 means silent, up to (about) 5 the verbosity gradually increases.
indent	A single non-negative integer controlling indentation of printed messages. 0 means no indentation, each unit above that adds two spaces.
nb_min_varExpl	Minimum proportion of variance explained for returned module eigengenes. Is capped at n_pc.

### Value

eigengenes Module eigengenes in a dataframe, with each column corresponding to one eigengene. The columns are named by the corresponding color with an "ME" prepended, e.g., 'MEturquoise'

etc. If `'return_valid_only==FALSE'`, module eigengenes whose calculation failed have all components set to `'NA'`.

`averageExpr` If `'align == 'along average''`, a dataframe containing average normalized expression in each module. The columns are named by the corresponding color with an `'AE'` prepended, e.g., `'Aeturquoise'` etc.

`var_explained` A dataframe in which each column corresponds to a module, with the component `'var_explained[PC, module]'` giving the variance of module `'module'` explained by the principal component no. `'PC'`. The calculation is exact irrespective of the number of computed principal components. At most 10 variance explained values are recorded in this dataframe.

`n_pc` A copy of the input `'n_pc'`.

`validMEs` A boolean vector. Each component (corresponding to the columns in `'data'`) is `'TRUE'` if the corresponding eigengene is valid, and `'FALSE'` if it is invalid. Valid eigengenes include both principal components and their hubgene approximations. When `'return_valid_only==FALSE'`, by definition all returned eigengenes are valid and the entries of `'validMEs'` are all `'TRUE'`.

`validColors` A copy of the input colors with entries corresponding to invalid modules set to `'grey'` if given, otherwise 0 if `'colors'` is numeric and `'grey'` otherwise.

`allOK` Boolean flag signalling whether all eigengenes have been calculated correctly, either as principal components or as the hubgene average approximation.

`allPC` Boolean flag signalling whether all returned eigengenes are principal components.

`isPC` Boolean vector. Each component (corresponding to the columns in `'eigengenes'`) is `'TRUE'` if the corresponding eigengene is the first principal component and `'FALSE'` if it is the hubgene approximation or is invalid.

`isHub` Boolean vector. Each component (corresponding to the columns in `'eigengenes'`) is `'TRUE'` if the corresponding eigengene is the hubgene approximation and `'FALSE'` if it is the first principal component or is invalid.

`validAEs` Boolean vector. Each component (corresponding to the columns in `'eigengenes'`) is `'TRUE'` if the corresponding module average expression is valid.

`allAEOK` Boolean flag signalling whether all returned module average expressions contain valid data. Note that `'return_valid_only==TRUE'` does not imply `'allAEOK==TRUE'`: some invalid average expressions may be returned if their corresponding eigengenes have been calculated correctly.

---

 nb\_plot\_dendro

*Plot dendrogram from Netboost output.*


---

## Description

Plot dendrogram from Netboost output.

## Usage

```
nb_plot_dendro(
  nb_summary = NULL,
  labels = FALSE,
  main = "",
  colorsrandom = FALSE
)
```

**Arguments**

nb_summary	Netboost results as generated by the nb_summary function.
labels	Boolean flag whether labels should be attached to the leaves.
main	Plot title.
colorsrandom	Boolean flag whether module colors should be shuffled.

**Value**

invisible null

**Examples**

```
data('tcga_aml_meth_rna_chr18', package='netboost')
results <- netboost(datan = tcga_aml_meth_rna_chr18, stepno = 20L,
soft_power = 3L, min_cluster_size = 10L, n_pc = 2, scale=TRUE,
ME_diss_thres = 0.25, qc_plot = FALSE)
set.seed(1234) # reproducible but shuffled color-module matching
nb_plot_dendro(nb_summary = results, labels = FALSE, main = 'Test',
colorsrandom = TRUE)
```

---

nb\_summary

*Summarize results from a forest. Plot trees together.*


---

**Description**

Summarize results from a forest. Plot trees together.

**Usage**

```
nb_summary(clust_res)
```

**Arguments**

clust_res	Clustering results from cut_trees call.
-----------	---

**Value**

List

**Examples**

```
data('tcga_aml_meth_rna_chr18', package='netboost')
cores <- as.integer(getOption('mc.cores', 2))
datan <- as.data.frame(scale(tcga_aml_meth_rna_chr18, center=TRUE,
scale=TRUE))
filter <- nb_filter(datan=datan, stepno=20L, until=0L, progress=1000L,
cores=cores,mode=2L)
dist <- nb_dist(datan=datan, filter=filter, soft_power=3L, cores=cores)
max_singleton = dim(tcga_aml_meth_rna_chr18)[2]
forest <- nb_mcupgma(filter=filter,dist=dist,max_singleton=max_singleton,
cores=cores)
```

```
trees <- tree_search(forest)
results <- cut_trees(trees=trees, datan=datan, forest=forest,
  min_cluster_size=10L, ME_diss_thres=0.25, qc_plot=FALSE)
sum_res <- nb_summary(clust_res=results)
```

---

nb_transfer	<i>Transfer of Netboost clustering to new data.</i>
-------------	---

---

## Description

Transfer of Netboost clustering to new data.

## Usage

```
nb_transfer(
  nb_summary = NULL,
  new_data = NULL,
  scale = FALSE,
  robust_PCs = FALSE,
  only_module_membership = FALSE
)
```

## Arguments

nb_summary	Netboost results as generated by the nb_summary function.
new_data	Data frame where rows correspond to samples and columns to features.
scale	Logical. Should data be scaled and centered?
robust_PCs	Should PCA be calculated on ranked data (Spearman PCA)? Rotations will not correspond to original data if this is applied.
only_module_membership	Logical. Should only module memberships be transferred and PCs be newly computed?

## Value

List

## Examples

```
data('tcga_aml_meth_rna_chr18', package='netboost')
results <- netboost(datan = tcga_aml_meth_rna_chr18, stepno = 20L,
  soft_power = 3L, min_cluster_size = 10L, n_pc = 2, scale=TRUE,
  ME_diss_thres = 0.25, qc_plot=FALSE)
ME_transfer <- nb_transfer(nb_summary = results,
  new_data = tcga_aml_meth_rna_chr18,
  scale = TRUE)
all(round(results[["MEs"]], 12) == round(ME_transfer, 12))
```

netboost

*Netboost clustering.***Description**

The Netboost clustering is performed in three subsequent steps. First, a filter of important edges in the network is calculated. Next, pairwise distances are calculated. Last, clustering is performed. For details see Schlosser et al. doi...

**Usage**

```
netboost(
  datan = NULL,
  stepno = 20L,
  filter_method = c("boosting", "skip", "kendall", "spearman", "pearson"),
  until = 0L,
  progress = 1000L,
  mode = 2L,
  soft_power = NULL,
  max_singleton = ncol(datan),
  qc_plot = TRUE,
  min_cluster_size = 2L,
  ME_diss_thres = 0.25,
  n_pc = 1,
  robust_PCs = FALSE,
  nb_min_varExpl = 0.5,
  cores = as.integer(getOption("mc.cores", 2)),
  scale = TRUE,
  method = c("pearson", "kendall", "spearman"),
  verbose = getOption("verbose")
)
```

**Arguments**

datan	Data frame where rows correspond to samples and columns to features.
stepno	Integer amount of boosting steps applied in the filtering step
filter_method	The following filtering methods are supported: "boosting" (non-zero coefficients in likelihood based boosting), "skip" (no filter), "kendall" (stats::cor.test), "spearman" (stats::cor.test), "pearson" (stats::cor.test)
until	Stop at index/column (if 0: iterate through all columns). For testing purposes in large datasets.
progress	Integer. If > 0, print progress after every X steps (Progress might not be reported completely accurate due to parallel execution)
mode	Integer. Mode (0: x86, 1: FMA, 2: AVX). Features are only available if compiled accordingly and available on the hardware.
soft_power	Integer. Exponent of the transformation. Set automatically based on the scale free topology criterion if unspecified.
max_singleton	Integer. The maximal singleton in the clustering. Usually equals the number of features.

qc_plot	Logical. Should plots be created?
min_cluster_size	Integer. The minimum number of features in one module.
ME_diss_thres	Numeric. Module Eigengene Dissimilarity Threshold for merging close modules.
n_pc	Number of principal components and variance explained entries to be calculated. The number of returned variance explained entries is currently 'min(n_pc,10)'. If given 'n_pc' is greater than 10, a warning is issued.
robust_PCs	Should PCA be calculated on ranked data (Spearman PCA)? Rotations will not correspond to original data if this is applied.
nb_min_varExpl	Minimum proportion of variance explained for returned module eigengenes. The number of PCs is capped at n_pc.
cores	Integer. Amount of CPU cores used (<=1 : sequential)
scale	Logical. Should data be scaled and centered?
method	A character string specifying the method to be used for correlation coefficients.
verbose	Additional diagnostic messages.

### Value

dendros A list of dendrograms. For each fully separate part of the network an individual dendrogram.

names A vector of feature names.

colors A vector of numeric color coding in matching order of names and module eigengene names (color = 3 -> variable in ME3).

MEs Aggregated module measures (Module eigengenes).

var\_explained Proportion of variance explained per module eigengene per principal component (max n\_pc principal components are listed).

rotation Matrix of variable loadings divided by their singular values. datan

filter Filter-Matrix as generated by the nb\_filter function.

### Examples

```
data('tcga_aml_meth_rna_chr18', package='netboost')
results <- netboost(datan=tcga_aml_meth_rna_chr18, stepno=20L,
  soft_power=3L, min_cluster_size=10L, n_pc=2, scale=TRUE,
  ME_diss_thres=0.25, qc_plot=TRUE)
```

---

netboostMCUPGMAPath	<i>Returns the absolute path to folder with mcupgma executables and scripts.</i>
---------------------	--

---

### Description

Returns the absolute path to folder with mcupgma executables and scripts.

**Usage**

```
netboostMCUPGMAPath()
```

**Value**

Absolute path for "mcupgma" folder

---

```
netboostPackagePath
```

*Returns the absolute path to "exec" folder in the package.*

---

**Description**

Returns the absolute path to "exec" folder in the package.

**Usage**

```
netboostPackagePath()
```

**Value**

Absolute path of installed package

---

```
netboostTmpCleanup
```

*Cleans the netboost temporary folder. This can be useful during the session as mcupgma creates vast directory structures (for iterations). Creates the own folder (all netboost temporary data is stored in netboostTmpPath(), which is equal to tempdir()/netboost). Also used for first time setup of folder.*

---

**Description**

Cleans the netboost temporary folder. This can be useful during the session as mcupgma creates vast directory structures (for iterations). Creates the own folder (all netboost temporary data is stored in netboostTmpPath(), which is equal to tempdir()/netboost). Also used for first time setup of folder.

**Usage**

```
netboostTmpCleanup(verbose = FALSE)
```

**Arguments**

verbose            Flag verbose

**Value**

none

---

netboostTmpPath	<i>Returns the absolute path to temporary folder of the package. To change temporary path, use normal R variables (TEMPDIR etc).</i>
-----------------	--

---

**Description**

Returns the absolute path to temporary folder of the package. To change temporary path, use normal R variables (TEMPDIR etc).

**Usage**

```
netboostTmpPath()
```

**Value**

Absolute path for "exec" folder

---

tcga_aml_meth_rna_chr18	<i>TCGA RNA and methylation measurement on a subset of chromosome 18 for 80 AML patients.</i>
-------------------------	---

---

**Description**

TCGA RNA and methylation measurement on a subset of chromosome 18 for 80 AML patients.

**Usage**

```
tcga_aml_meth_rna_chr18
```

**Format**

A data frame with 80 rows (patients) and 500 variables (features).

**Source**

<https://portal.gdc.cancer.gov/>

---

tree_dendro	<i>Calculate the dendrogram for an individual tree</i>
-------------	--

---

**Description**

Calculate the dendrogram for an individual tree

**Usage**

```
tree_dendro(tree, datan, forest)
```

**Arguments**

tree	A list with two elements. ids, which is an integer vector of feature identifiers and rows, which is an integer vector of selected rows in the corresponding forest
datan	Data frame where rows correspond to samples and columns to features.
forest	Raw dendrogram-matrix as generated by the nb_mcupgma function.

**Value**

List of tree specific objects including dendrogram, tree data and features.

---

tree_search	<i>Extracts independent trees from nb_mcupgma results (external wrapper for internal C++ function)</i>
-------------	--

---

**Description**

Extracts independent trees from nb\_mcupgma results (external wrapper for internal C++ function)

**Usage**

```
tree_search(forest = NULL)
```

**Arguments**

forest	Raw dendrogram-matrix as generated by the nb_mcupgma function.
--------	--

**Value**

List

**Examples**

```
data('tcga_aml_meth_rna_chr18', package='netboost')
cores <- as.integer(getOption('mc.cores', 2))
datan <- as.data.frame(scale(tcga_aml_meth_rna_chr18, center=TRUE,
                             scale=TRUE))
filter <- nb_filter(datan=datan, stepno=20L, until=0L, progress=1000L,
                    cores=cores,mode=2L)
dist <- nb_dist(datan=datan, filter=filter, soft_power=3L, cores=cores)
max_singleton = dim(tcga_aml_meth_rna_chr18)[2]
forest <- nb_mcupgma(filter=filter, dist=dist,
                    max_singleton=max_singleton, cores=cores)
trees <- tree_search(forest)
str(trees[[length(trees)]])
```

# Index

[.onAttach](#), 2

[calculate\\_adjacency](#), 3

[cpp\\_dist\\_tom](#), 4

[cpp\\_filter\\_base](#), 4

[cpp\\_filter\\_end](#), 5

[cpp\\_filter\\_step](#), 5

[cpp\\_tree\\_search](#), 6

[cut\\_dendro](#), 6

[cut\\_trees](#), 7

[mcupgma\\_exec](#), 8

[nb\\_clust](#), 9

[nb\\_dist](#), 10

[nb\\_filter](#), 11

[nb\\_mcupgma](#), 12

[nb\\_moduleEigengenes](#), 13

[nb\\_plot\\_dendro](#), 15

[nb\\_summary](#), 16

[nb\\_transfer](#), 17

[netboost](#), 18

[netboostMCUPGMAPath](#), 19

[netboostPackagePath](#), 20

[netboostTmpCleanup](#), 20

[netboostTmpPath](#), 21

[tcga\\_aml\\_meth\\_rna\\_chr18](#), 21

[tree\\_dendro](#), 22

[tree\\_search](#), 22