

# Package ‘treethresh’

January 2, 2012

**Version** 0.1-5

**Date** 2009-01-06

**Title** Methods for Tree-based Local Adaptive Thresholding

**Author** Ludger Evers <ludger@stats.gla.ac.uk> and Tim Heaton <t.heaton@sheffield.ac.uk>

**Maintainer** Ludger Evers <ludger@stats.gla.ac.uk>

**Depends** EbayesThresh, wavethresh

**Description** This package implements TreeThresh, a locally adaptive version of EbayesThresh.

**License** GPL (>= 3)

**Repository** CRAN

**Date/Publication** 2010-01-07 15:42:08

## R topics documented:

coefficients . . . . .	2
estimate.sdev . . . . .	4
get.t . . . . .	5
get.w . . . . .	6
prune . . . . .	7
row263 . . . . .	8
subtree . . . . .	9
thresh . . . . .	10
tiles . . . . .	11
trees . . . . .	11
treethresh . . . . .	12
wavelet.treethresh . . . . .	14
wtthresh . . . . .	16

<b>Index</b>	<b>19</b>
--------------	-----------

---

coefficients

*Extracting and updating coefficients stored in wd or imwd objects*


---

### Description

`extract.coefficients` extracts wavelet coefficient vectors (in case of `wd`) and coefficient matrices (in case of `imwd`), so that these can be thresholded by `treethresh` or `wtthresh`. `update.coefficients` re-inserts these vector or matrices into the `wd` or `imwd` objects, such that the inverse transform can be computed using the thresholded coefficients.

### Usage

```
extract.coefficients(object, start.level=5)
insert.coefficients(object, update)
extract.coefficients.wd(object, start.level=5)
insert.coefficients.wd(object, update)
extract.coefficients.imwd(object, start.level=5)
insert.coefficients.imwd(object, update)
```

### Arguments

<code>object</code>	For <code>extract.coefficients</code> the <code>wd</code> or <code>imwd</code> object to extract the coefficients from. For <code>insert.coefficients</code> the <code>wd</code> or <code>imwd</code> object to be updated.
<code>start.level</code>	The coarsest level of coefficients to be extracted ( <code>extract.coefficients</code> only)
<code>update</code>	A list with the matrices that should be copied into the <code>wd</code> or <code>imwd</code> object. ( <code>update.coefficients</code> only)
<code>...</code>	additional arguments (see above for supported arguments).

### Value

`extract.coefficients` returns the coefficient matrices to be extracted. `update.coefficients` returns the updated `wd` or `imwd` object.

`insert.coefficients` returns the updated `wd` or `imwd` object into which the coefficients from `update` have been inserted.

### Note

`extract.coefficients.wd` and `extract.coefficients.imwd` should rarely be directly called by the user. The more user-friendly S3 function `extract.coefficients` will take care of calling the right function (idem for `insert.coefficients`).

### See Also

[treethresh](#), [wtthresh](#), [wavelet.treethresh](#)

**Examples**

```
## The following examples shows how an example image can be
## thresholded step by step. All the steps are combined in the more
## user-friendly function wavelet.treethresh

## (01) Load the example image
data(tiles)

## (02) Display the image
par(mai=rep(0,4)) ; image(tiles,col=grey(0:255/255))

## (03) Add noise to the image
corrupted <- tiles + rnorm(length(tiles))

## (04) Display the corrupted image
par(mai=rep(0,4)) ; image(corrupted, col=grey(0:255/255))

## (05) Compute the wavelet transform
corrupted.wt <- imwd(corrupted)

## (06) Estimate the standard error
dev <- estimate.sdev(corrupted.wt)

## (07) Extract the coefficient matrices to be thresholded
coefs <- extract.coefficients(corrupted.wt)

## (08) Rescale the coefficients using the estimated standard error
## (should be around 1)
for (nm in names(coefs))
  coefs[[nm]] <- coefs[[nm]] / dev

## (09) Compute the tree
coefs.tree <- wthresh(coefs)

## (10) Prune the tree
coefs.pruned.tree <- prune.wthresh(coefs.tree)

## (11) Threshold according to the pruned tree
coefs.threshed <- thresh(coefs.pruned.tree)

## (12) Undo the rescaling
for (nm in names(coefs))
  coefs.threshed[[nm]] <- coefs.threshed[[nm]] * dev

## (13) Update coefficients
denoised.wt <- insert.coefficients(corrupted.wt, coefs.threshed)

## (14) Compute inverse wavelet transform
denoised <- imwr(denoised.wt)

## (15) Display denoised image
par(mai=rep(0,4)) ; image(denoised, col=grey(0:255/255))
```

```
## (16) Compute l2 loss
sum((denoised-tiles)^2)

## Equivalently we could have called
## denoised.wt <- wavelet.treethresh(corrupted.wt)
## instead of steps (06) - (13)
```

---

estimate.sdev

*Estimate noise level from wavelet coefficients*

---

## Description

estimate.sdev estimates by analysing the the wavelet coefficients at the finest level(s) of an [wd](#) or [imwd](#) object.

## Usage

```
estimate.sdev(object, dev=mad)
estimate.sdev.wd(object, dev=mad)
estimate.sdev.imwd(object, dev=mad)
```

## Arguments

object	For <code>extract.coefficients</code> the <a href="#">wd</a> or <a href="#">imwd</a> object for which to estimate the standard error.
dev	The function used to estimate the standard error. Typical examples are <a href="#">sd</a> , or the more robust <a href="#">mad</a> (default).
...	additional arguments (see above for supported arguments).

## Value

estimate.sdev returns an estimate of the standard error.

## Note

estimate.dev.wd and estimate.dev.imwd should rarely be directly called by the user. The more user-friendly S3 function estimate.dev will take care of calling the right function.

For an example of the use of estimate.sdev, see [coefficients](#).

---

get.t	<i>Extract estimated hard threshold</i>
-------	---

---

### Description

Extracts from a tree object of the classes [treethresh](#) or [wtthresh](#) the estimated value of the hard threshold  $t$  for each data point.

### Usage

```
get.t(object, C)
get.t.treethresh(object, C)
get.t.wtthresh(object, C)
```

### Arguments

object	An object of the class <a href="#">treethresh</a> or <a href="#">wtthresh</a> from which the thresholds are to be computed.
C	If C is given, then the tree is pruned with constant $C$ (see the help of <a href="#">prune</a> for details on the pruning process).

### Value

`get.t` returns an estimated value of the hard threshold  $t$  for each data point used to estimate `object`. If `object` is of the class [treethresh](#), then `get.t` returns an array (or matrix) of exactly the same dimension as the data that was used to estimate `object`. If `object` is of the class [wtthresh](#) then it returns a list with elements of the same name and dimensions as the list that was used to estimate `object`.

### Note

`get.t.treethresh` and `get.t.wtthresh` should rarely be directly called by the user. The more user-friendly S3 function `get.t` will take care of calling the right function.

### See Also

[treethresh](#), [wtthresh](#), [get.w](#), [prune](#)

---

`get.w`*Extract estimated weights*

---

### Description

Extracts from a tree object of the classes [treethresh](#) or [wtthresh](#) the estimated value of the weight  $w$  for each data point.

### Usage

```
get.w(object, C)
get.w.treethresh(object, C)
get.w.wtthresh(object, C)
```

### Arguments

<code>object</code>	An object of the class <a href="#">treethresh</a> or <a href="#">wtthresh</a> from which the weights are to be extracted.
<code>C</code>	If $C$ is given then the tree is pruned with constant $C$ (see the help of <a href="#">prune</a> for details on the pruning).

### Value

`get.w` returns an estimated value of the weight  $w$  for each data point used to estimate `object`. If `object` is of the class [treethresh](#), then `get.w` returns an array (or matrix) of exactly the same dimension as the data that was used to estimate `object`. If `object` is of the class [wtthresh](#) then it returns a list with elements of the same name and dimensions as the list that was used to estimate `object`.

### Note

`get.w.treethresh` and `get.w.wtthresh` should rarely be directly called by the user. The more user-friendly S3 function `get.w` will take care of calling the right function.

### See Also

[treethresh](#), [wtthresh](#), [get.t](#), [prune](#)

---

prune	<i>Prune a tree using cross-validation</i>
-------	--

---

### Description

Extracts an optimal subtree from a tree object of the classes `treethresh` or `wtthresh`. Contrary to `subtree` the values of the complexity parameter  $C$  does not need to be given, but is determined using cross-validation.

### Usage

```
prune(object, v=5, sd.mult=0.5, plot=TRUE)
prune.treethresh(object, v=5, sd.mult=0.5, plot=TRUE)
prune.wtthresh(object, v=5, sd.mult=0.5, plot=TRUE)
```

### Arguments

object	An object of the class <code>treethresh</code> or <code>wtthresh</code> according to which thresholding is to be carried out.
v	The number of folds in the cross-validation used to determine the optimal subtree in the pruning step (see below for details).
sd.mult	The smallest subtree that is not <code>sd.mult</code> times the standard error worse than the best loglikelihood will be chosen as the optimal tree in the pruning step. (see below for details).
plot	If <code>plot=TRUE</code> a plot of the relative predicted loglikelihood estimated in the cross-validation against the complexity parameter $C$ is produced.
...	additional arguments (see above for supported arguments).

### Details

The tree grown by `treethresh` or `wtthresh` often yields too many partitions leading to an overfit. The resulting tree has to be 'pruned', i.e. the branches corresponding to the least important regions have to be 'snipped off'.

As the `TreeThresh` model is a special case of a classification and regression tree, there exists a sequence of nested subtrees (i.e. a sequence of nested partitions) that maximises the regularised loglikelihood

$$\ell + \alpha \cdot \# \text{ partitions.}$$

The parameter  $\alpha$  controls the complexity of the resulting partition. For  $\alpha = 0$  no pruning is carried out. If a large enough  $\alpha$  is chosen, only the root node of the tree is retained, i.e. no partitioning is done. Denote this value of  $\alpha$  by

$\alpha_0$ . The complexity parameter can thus be rescaled to

$$C = \frac{\alpha}{\alpha_0}$$

yielding a complexity parameter ranging from 0 (no pruning) to 1 (only retain the root node).

The optimal value of the complexity parameter  $C$  (or, equivalently,  $\alpha$ ) depends on the problem at hand and thus has to be chosen carefully. `prune` estimates the optimal complexity parameter  $C$  by a  $v$ -fold cross-validation. If `sd.mult=0` the value of  $C$  that yields the highest predictive loglikelihood in the cross-validation is used to prune the tree object. If `sd.mult` is not 0 the largest  $C$  that is not `sd.mult` standard errors worse than the best  $C$  is used.

### Value

`prune` returns an object of the class `treethresh` or `wtthresh` that contains a tree pruned at value  $C$  (see the function `prune` for details on the pruning process).

### Note

`prune.treethresh` and `prune.wtthresh` should rarely be directly called by the user. The more user-friendly S3 function `prune` will take care of calling the right function.

For an example of the use of `prune`, see `coefficients`.

### See Also

`treethresh`, `wtthresh`, `get.t`, `prune`

---

row263

*Example signal sequence*

---

### Description

Sequence of length 512 corresponding to the 263rd row of `trees`. The sequence was used as an example in Evers and Heaton (2009).

### Usage

```
row263
```

### References

Evers, L. and Heaton, T. (2009) Locally Adaptive Tree-Based Thresholding. *Journal of Computational and Graphical Statistics*. Dec 2009, Vol. 18, No. 4: 961-977.

### See Also

`trees`

### Examples

```
data(row263)
plot(row263, type="b")
```

---

subtree	<i>Extract subtree by pruning according to a specified complexity parameter</i>
---------	---

---

### Description

Extracts a subtree from a tree object of the classes [treethresh](#) or [wtthresh](#) by pruning according to a specified value of the complexity parameter  $C$ .

### Usage

```
subtree(object, C)
subtree.treethresh(object, C)
subtree.wtthresh(object, C)
```

### Arguments

object	An object of the class <a href="#">treethresh</a> or <a href="#">wtthresh</a> from which a subtree is to be extracted.
C	The value of the complexity parameter $C$ to be used for the pruning.

### Value

subtree returns an object of the class [treethresh](#) or [wtthresh](#) that contains a tree pruned at value  $C$  (see the function [prune](#) for details on the pruning process).

### Note

subtree.treethresh and subtree.wtthresh should rarely be directly called by the user. The more user-friendly S3 function subtree will take care of calling the right function.

Use the function [prune](#) to carry out the pruning without having to specify the complexity parameter  $C$ .

### See Also

[treethresh](#), [wtthresh](#), [prune](#)

---

thresh	<i>Perform thresholding according to a tree</i>
--------	---

---

### Description

Thresholds according to a tree object of the classes [treethresh](#) or [wtthresh](#).

### Usage

```
thresh(object, data, C, postmed=TRUE)
thresh.treethresh(object, data, C, postmed=TRUE)
thresh.wtthresh(object, data, C, postmed=TRUE)
```

### Arguments

object	An object of the class <a href="#">treethresh</a> or <a href="#">wtthresh</a> according to which thresholding is to be carried out.
data	The data to be thresholded. If object is of the class <a href="#">treethresh</a> , then object has to be an array (or matrix) of exactly the same dimension as the data that was used to estimate object. If object is of the class <a href="#">wtthresh</a> it has to be a list with elements of the same name and dimensions as the list that was used to estimate object. If no data object is given and if object is not estimated directly from the $\beta$ then the data used to estimate object is used.
C	If C is given then the tree is pruned with constant $C$ (see the help of <a href="#">prune</a> for details on the pruning).
postmed	If postmed=TRUE then the thresholding is done by returning the posterior median $\mu_i X_i$ . If postmed=FALSE then hard thresholding is performed.

### Value

thresh returns the thresholded data.

### Note

thresh.treethresh and thresh.wtthresh should rarely be directly called by the user. The more user-friendly S3 function thresh will take care of calling the right function.

For an example of the use of thresh, see [coefficients](#).

### See Also

[treethresh](#), [wtthresh](#), [get.t](#), [prune](#)

---

tiles	<i>Example image</i>
-------	----------------------

---

**Description**

Greyscale image of 128 x 128 pixels containing three tiles of concentric circles.

**Usage**

```
tiles
```

**Examples**

```
data(tiles)
image(tiles)
```

---

trees	<i>Example image</i>
-------	----------------------

---

**Description**

Greyscale image of 512 x 512 pixels showing tree close to Swellendam, Western Cape, South Africa. The image was used as an example in Evers and Heaton (2009).

**Usage**

```
trees
```

**References**

Evers, L. and Heaton, T. (2009) Locally Adaptive Tree-Based Thresholding. *Journal of Computational and Graphical Statistics*. Dec 2009, Vol. 18, No. 4: 961-977.

**See Also**

[row263](#)

**Examples**

```
data(trees)
image(trees, col=grey(seq(0,1,length.out=64)))
```

---

treethresh	<i>Compute optimal thresholding partition for a sequence, matrix, or array of data.</i>
------------	---

---

### Description

This function carries out the tree-based thresholding algorithm described in section 3 of Evers and Heaton (2009).

### Usage

```
treethresh(data, beta, criterion="score", control=list(),
           rho=sys.frame(sys.parent()))
```

### Arguments

data	An array (or an object coercible to an array, i.e. a vector or matrix) containing the data. The data is assumed to have noise of unit variance, thus the data needs to be rescaled a priori (e.g. in the case of wavelet coefficients using function <a href="#">estimate.sdev</a> ).
beta	Instead of using the original data, one can call <code>wthresh</code> using the $\beta_i$ instead of the observed data. These can be computed using <a href="#">beta.laplace</a> .
criterion	The criterion to be used. Can be "score" (default) for using the score test, "likelihood" for using the likelihood ratio test (slower), "heuristic" for using a heuristic criterion based on the original data, or a user-specified function that computes the goodness of a split. This function should take four arguments (which should be self-explanatory), <code>left_data</code> , <code>left_betas</code> , <code>right_data</code> , and <code>right_betas</code> .
control	A list that allows the user to tweak the behaviour of <code>treethresh</code> . It can contain the following elements: <ul style="list-style-type: none"> <li><b>max.depth</b> The maximum depth of the tree. Defaults to 10.</li> <li><b>minimum.width</b> The minimum width of a region of the partitions. This setting avoids creating too small regions. Defaults to 3.</li> <li><b>minimum.size</b> The minimum size of a region of the partitions. This setting avoids creating too small regions. Defaults to <math>5^d</math>, where <math>d</math> is the dimension of the array.</li> <li><b>lr.signif</b> If the p-value of the corresponding likelihood ratio test is larger than <code>1-lr.signif</code> a split will be discarded. Defaults to 0.5.</li> <li><b>absolute.improvement</b> The minimum absolute improvement of the above criterion necessary such that a split is retained. Defaults to <code>-Inf</code>, i.e. deactivated.</li> <li><b>relative.improvement</b> The minimum relative improvement of the above criterion necessary such that a split is retained. Defaults to <code>-Inf</code>, i.e. deactivated.</li> </ul>

- absolute.criterion** The minimum value of the above criterion necessary such that a split is retained. Defaults to 0, i.e. deactivated.
- a** The parameter  $a$  of the Laplace distribution  $\gamma(\mu) \propto \exp(-a|\mu|)$  corresponding to the signal. Defaults to 0.5.
- beta.max** The maximum value of  $\beta$ . Defaults to 1e5.
- max.iter** The maximum number of iterations when computing the estimate of the weight  $w$  in a certain region. Defaults to 30.
- tolerance.grad** The estimate of the weight  $w$  in a certain region is considered having converged, if the gradient of the likelihood is less than `tolerance.grad`. Defaults to 1e-8.
- tolerance** The estimate of the weight  $w$  in a certain region is considered having converged, if the estimates of the weight  $w$  change less than `tolerance`. Defaults to 1e-6.
- rho The environment used to evaluate the user-specified criterion function if one is supplied). (You want to change this argument only in very rare circumstances).

## Value

treethresh returns an object of the class `c("treethresh")`, which is a list containing the following elements:

splits	A table describing the detailed structure of the fitted tree together with the local loglikelihoods required for the pruning.
membership	An array of the same dimension as <code>data</code> or <code>beta</code> indicating to which region each entry of the array of data belongs.
beta	The values of $\beta$ for each observation / coefficient.
data	The data used.
criterion	The criterion used to decide on splits (see argument <code>criterion</code> ).
control	The control list of tuning options used (see argument <code>control</code> ).

## References

Evers, L. and Heaton, T. (2009) Locally Adaptive Tree-Based Thresholding. *Journal of Computational and Graphical Statistics*. Dec 2009, Vol. 18, No. 4: 961-977.

## Examples

```
# (1) Create a vector with the probabilities of a signal being present
w.true <- c(rep(0.1,400),rep(0.7,300),rep(0.1,300))

# (2) Generate the signal
mu <- numeric(length(w.true))
non.zero.entry <- runif(length(mu))<w.true
num.non.zero.entries <- sum(non.zero.entry)
mu[non.zero.entry] <- rexp(num.non.zero.entries,rate=0.5)*sample(c(-1,1),num.non.zero.entries,replace=TRUE)

# (3) Split graphics device
par(mfrow=c(2,2))
```

```

# (3) Draw the true signal (signal present in red)
plot(mu,col=non.zero.entry+1)
title("True signal")

# (4) Add noise to the signal
x <- mu + rnorm(length(mu))

# (5) Plot the noisy signal (signal present in red)
plot(x,col=non.zero.entry+1)
title("Noisy signal")

# (6) Carry out the tree-based thresholding
tt <- treethresh(x)

# (7) Prune the tree
tt.pruned <- prune(tt)

# (8) Threshold the signal according to the pruned tree
mu.hat <- thresh(tt.pruned)

# (9) Print the denoised signal
plot(mu.hat,col=non.zero.entry+1)
title("Denoised signal")

# (10) Add solid lines for splits (lines removed by the pruing are dashed)
abline(v=tt$split[,"pos"],lty=2)
abline(v=tt.pruned$split[,"pos"],lty=1)

```

---

wavelet.treethresh      *Threshold wavelet coefficients*

---

## Description

wavelet.threshold is a more user-friendly function for thresholding wavelet coefficients stored in an `wd` or `imwd` object. It combines the functions `extract.coefficients`, `estimate.sdev` (rescales the coefficients accordingly), `treethresh` or `wtthresh`, `prune`, `thresh`, and `insert.coefficients`

## Usage

```

wavelet.treethresh(object, sdev=NA, dev=mad, start.level=5,
                   levelwise=FALSE, v=5, sd.mult=0.5, postmed=TRUE, ...)

```

## Arguments

object	An object of the class <code>wd</code> or <code>wtthresh</code> to be smoothed by thresholding.
sdev	The standard error of the noise (if known), otherwise NA.
dev	The function used to estimate the standard error. Typical examples are <code>sd</code> (the default), or the more robust <code>mad</code> . Not used if <code>sdev</code> is given.

start.level	The level in the wavelet transform from which to commence thresholding the wavelet coefficients. This level and all those finer levels will be thresholded.
levelwise	Indicates whether the thresholding should be carried out independently for each level and type of coefficients as in the function <a href="#">treethresh</a> (if levelwise=TRUE), or whether a common partition is to be found jointly for all coefficients as in the function <a href="#">wtthresh</a> (if levelwise=FALSE).
v	The number of folds in the cross-validation used to determine the optimal subtree in the pruning step (see the function <a href="#">prune</a> for details). NA if no pruning is to be carried out.
sd.mult	The smallest subtree that is not sd.mult times the standard error worse than the best loglikelihood will be chosen as the optimal tree in the pruning step. (see the function <a href="#">prune</a> for details).
postmed	Controls whether thresholding is to be carried out by using the posterior median of the coefficient $\mu y$ (postmed=true) or by using a hard threshold (postmed=FALSE). (See the function <a href="#">thresh</a> for more details.)
...	arguments passed to <a href="#">wtthresh</a> (if levelwise=FALSE) or <a href="#">treethresh</a> (if levelwise=TRUE).

**Value**

Returns an object of the class [wd](#) or [wtthresh](#), where the coefficients have been thresholded using the TreeThresh algorithm.

**Note**

wavelet.treethresh combines the functions [extract.coefficients](#), [estimate.sdev](#) (and the appropriate scaling), [treethresh](#) / [wtthresh](#), [prune](#), [insert.coefficients](#) into a single (hopefully) more user-friendly function.

**References**

Evers, L. and Heaton, T. (2009) Locally Adaptive Tree-Based Thresholding. Journal of Computational and Graphical Statistics. Dec 2009, Vol. 18, No. 4: 961-977.

**See Also**

[extract.coefficients](#), [estimate.sdev](#), [treethresh](#), [wtthresh](#), [prune](#), [thresh](#), [insert.coefficients](#)

**Examples**

```
## The following examples shows how an example image can be
## thresholded using the more user-friendly function
## wavelet.treethresh

## (01) Load the example image
data(tiles)

## (02) Display the image
par(mai=rep(0,4)) ; image(tiles,col=grey(0:255/255))
```

```

## (03) Add noise to the image
corrupted <- tiles + rnorm(length(tiles))

## (04) Display the corrupted image
par(mai=rep(0,4)) ; image(corrupted,col=grey(0:255/255))

## (05) Compute the wavelet transform
corrupted.wt <- imwd(corrupted)

## (06) Perform the thresholding
denoised.wt <- wavelet.treethresh(corrupted.wt)

## (07) Compute inverse wavelet transform
denoised <- imwr(denoised.wt)

## (08) Display denoised image
par(mai=rep(0,4)) ; image(denoised,col=grey(0:255/255))

## (09) Compute l2 loss
sum((denoised-tiles)^2)

## The call to wavelet.treethresh is equivalent to steps (06) to (13)
## of the example in the help section "coefficients".

```

---

wtthresh

*Compute optimal thresholding partition for a sequence of linked arrays*

---

## Description

This function carries out the joint thresholding algorithm described in section 5 of Evers and Heaton (2009). Though the function, in principle, can work any sequence of arrays, it is designed to work with blocks of wavelet coefficients. These can be extracted from an `wd` or `imwd` object using the function `extract.coefficients`.

## Usage

```
wtthresh(data, beta, weights, control=list())
```

## Arguments

`data` A list containing the arrays to be thresholded. The elements of the array have to be arrays of the same number of dimensions. The arrays can be of different sizes, however the ratio of their side lengths has to be the same. Can be a list of wavelet coefficients extracted using `extract.coefficients`. The data is assumed to have noise of unit variance, thus the data needs to be rescaled a priori (e.g. in the case of wavelet coefficients using function `estimate.sdev`).

beta	Instead of using the original data, one can call wtthresh using the $\beta_i$ instead of the observed data. These can be computed using <code>beta.laplace</code> .
weights	The different elements of the list can be weighted. This allows for giving greater weight to certain arrays. By default, no weights are used.
control	A list that allows the user to tweak the behaviour of wtthresh. It can contain the following elements: <ul style="list-style-type: none"> <li><b>max.depth</b> The maximum depth of the tree. Defaults to 10.</li> <li><b>minimum.width</b> The minimum width of a region of the partitioning of the largest array. This setting avoids creating too small regions. Defaults to 4.</li> <li><b>min.width.scale.factor</b> If the minimum width of the largest array of size <math>l_0^d</math> is <code>minimum.width</code>, then the minimum width for an array of the size <math>l^2</math> is <math>\text{minimum.width} \cdot \left(\frac{l}{l_0}\right)^{d \cdot \text{minimum.width.scale.factor}}</math>. Defaults to 1.</li> <li><b>min.minimum.width</b> Do not allow the minimum width of a region to become less than <code>min.minimum.width</code> at any level. Defaults to 1.</li> <li><b>minimum.size</b> The minimum size of a region of the partitioning of the largest array. This setting avoids creating too small regions. Defaults to <math>8^d</math>, where <math>d</math> is the dimension of the arrays.</li> <li><b>min.size.scale.factor</b> If the minimum size of the largest array of size <math>l_0^d</math> is <code>minimum.size</code>, then the minimum size for an array of the size <math>l^2</math> is <math>\text{minimum.size} \cdot \left(\frac{l}{l_0}\right)^{\text{minimum.size.scale.factor}}</math>. Defaults to 1.</li> <li><b>min.minimum.size</b> Do not allow the minimum size of a region to become less than <code>min.minimum.size</code> at any level. Defaults to <math>4^d</math>, where <math>d</math> is the dimension of the arrays.</li> <li><b>rescale.quantile</b> In order to correct for different depths of the splits (due to minimum size and width requirements) the score statistic <math>s</math> is rescaled: <math>(s - q_\alpha(df)) / q_\alpha(df)</math>, where <math>q_\alpha(df)</math> is the <math>\alpha</math> quantile of a <math>\chi^2</math> distribution with <math>df</math> degrees of freedom, and <math>\alpha</math> is set to <code>rescale.quantile</code>. Defaults to 0.5.</li> <li><b>lr.signif</b> If the p-value of the corresponding likelihood ratio test is larger than <code>1-lr.signif</code> a split will be discarded. Defaults to 0.5.</li> <li><b>absolute.improvement</b> The minimum absolute improvement of the above criterion necessary such that a split is retained. Defaults to <code>-Inf</code>, i.e. deactivated.</li> <li><b>relative.improvement</b> The minimum relative improvement of the above criterion necessary such that a split is retained. Defaults to <code>-Inf</code>, i.e. deactivated.</li> <li><b>a</b> The parameter <math>a</math> of the Laplace distribution <math>\gamma(\mu) \propto \exp(-a \mu )</math> corresponding to the signal. Defaults to 0.5.</li> <li><b>beta.max</b> The maximum value of <math>\beta</math>. Defaults to <code>1e5</code>.</li> <li><b>max.iter</b> The maximum number of iterations when computing the estimate of the weight <math>w</math> in a certain region. Defaults to 30.</li> <li><b>tolerance.grad</b> The estimate of the weight <math>w</math> in a certain region is considered having converged, if the gradient of the likelihood is less than <code>tolerance.grad</code>. Defaults to <code>1e-8</code>.</li> </ul>

**tolerance** The estimate of the weight  $w$  in a certain region is considered having converged, if the estimates of the weight  $w$  change less than tolerance. Defaults to  $1e-6$ .

### Value

wtthresh returns an object of the class `c("wtthresh")`, which is a list containing the following elements:

splits	A table describing the structure of the fitted tree together with the local loglikelihoods required for the pruning.
details	A table giving the details about where the split was carried out for each of the arrays (i.e. for each block of coefficients).
w	The weights $w$ of the mixture component corresponding to the signal for each region as described by the corresponding row of splits.
t	The corresponding hard threshold $t$ for each region as described by the corresponding row of splits.
membership	A list of the same length as data indicating to which region each entry of the arrays of data belongs.
beta	The values of $\beta$ for each coefficient (as a list).
data	The data used (as a list).
weights	The weights used for each array of observations/coefficients.
control	The control list of tuning options used. (see argument control).

### Note

For an example of the use of wtthresh, see [coefficients](#).

### References

Evers, L. and Heaton T. (2009) Locally Adaptive Tree-Based Thresholding. Journal of Computational and Graphical Statistics. Dec 2009, Vol. 18, No. 4: 961-977.

# Index

## \*Topic **datasets**

row263, 8  
tiles, 11  
trees, 11

## \*Topic **nonparametric**

estimate.sdev, 4  
get.t, 5  
get.w, 6  
prune, 7  
subtree, 9  
thresh, 10  
treethresh, 12  
wavelet.treethresh, 14  
wthresh, 16

## \*Topic **tree**

estimate.sdev, 4  
get.t, 5  
get.w, 6  
prune, 7  
subtree, 9  
thresh, 10  
treethresh, 12  
wavelet.treethresh, 14  
wthresh, 16

## \*Topic **utilities**

coefficients, 2

beta.laplace, 12, 17

coefficients, 2, 4, 8, 10, 18

estimate.sdev, 4, 12, 14–16

extract.coefficients, 14–16

extract.coefficients(coefficients), 2

get.t, 5, 6, 8, 10

get.w, 5, 6

imwd, 2, 4, 14, 16

insert.coefficients, 14, 15

insert.coefficients(coefficients), 2

mad, 4, 14

prune, 5, 6, 7, 8–10, 14, 15

row263, 8, 11

sd, 4, 14

subtree, 7, 9

thresh, 10, 14, 15

tiles, 11

trees, 8, 11

treethresh, 2, 5–10, 12, 14, 15

wavelet.treethresh, 2, 14

wd, 2, 4, 14–16

wthresh, 2, 5–10, 14, 15, 16