Package ‘conTree’

May 8, 2026

Type Package

Title Contrast Trees and Boosting

Description Contrast trees represent a new approach for assessing the accuracy of many types of machine learning estimates that are not amenable to standard (cross) validation methods; see ``Contrast trees and distribution boosting'', Jerome H. Friedman (2020) <doi:10.1073/pnas.1921562117>. In situations where inaccuracies are detected, boosted contrast trees can often improve performance. Functions are provided to to build such trees in addition to a special case, distribution boosting, an assumption free method for estimating the full probability distribution of an outcome variable given any set of joint input predictor variable values.

Version 0.3-1

VignetteBuilder knitr

License Apache License 2.0

Encoding UTF-8

URL `https://jhfhub.github.io/conTree_tutorial/`

BugReports `https://github.com/bnaras/conTree/issues`

Imports stats, graphics

RoxygenNote 7.2.3

Suggests randomForest, knitr, rmarkdown

LazyData true

Depends R (>= 3.5)

NeedsCompilation yes

Author Jerome Friedman [aut, cph],
Balasubramanian Narasimhan [aut, cre]

Maintainer Balasubramanian Narasimhan <`naras@stanford.edu`>

Repository CRAN

Date/Publication 2023-11-22 09:20:12 UTC

Contents

`conTree-package` *Contrast and Boosted Trees*

Description

Contrast trees represent a new approach for assessing the accuracy of many types of machine learning estimates that are not amenable to standard (cross) validation methods. In situations where inaccuracies are detected, boosted contrast trees can often improve performance. Functions are provided to to build such trees in addition to a special case, distribution boosting, an assumption free method for estimating the full probability distribution of an outcome variable given any set of joint input predictor variable values.

Author(s)

Original code (C) by Jerome H. Friedman, minor modifications, formatting, and packaging by Balasubramanian Narasimhan

References

Jerome Friedman (2019). *Contrast Trees and Distribution Boosting* `https://arxiv.org/abs/1912.03785`

`age_data` *Age and Demographics data*

Description

The data come from 9243 questionnaires filled out by shopping mall customers in the San Francisco Bay Area (Impact Resources, Inc., Columbus, OH). Here we attempt to estimate a persons age as a function of the other 13 demographic variables. For this data set age value is reported as being in one of seven intervals `{13-17, 18-24, 25-34, 35-44, 45-54, 55-64, >= 65}`. Each persons age is randomly generated uniformly within its corresponding reported interval. For the last interval an exponential distribution was used with mean corresponding to life expectancy after reaching age 65.

Usage

```
age_data
```

Format

`age_data`**:**

A list of 3 items.

xage data frame of 8856 observations on 13 variables

yage Randomly generated age in the range above

gbage gradient boosting model for median age given x

Source

The Elements of Statistical Learning, Data Mining, Second Edition, by Hastie, Tibshirani, and Friedman.

`air_quality` *Air Quality Data from UC Irvine Machine Learning Repository*

Description

The data set consists of hourly averaged measurements from an array of 5 metal oxide chemical sensors embedded in an air quality chemical multisensor device. The outcome variable `y` is the corresponding true hourly averaged concentration CO taken from a reference analyzer. The input variables `x` are taken to be the corresponding hourly averaged measurements of the 13 other quantities.

Usage

```
air_quality
```

Format

`air_quality`:

A list with 4 items.

- **xco** data frame of 9357 observations on 13 variables
- **yco** hourly averaged CO concentration
- **zco** sample membership indicator
- **pr2** probability propensity score

Source

https://archive.ics.uci.edu/ml/datasets/air+quality

`census` *Census Data Example from UC Irvine Machine Learning Repository*

Description

Includes a data frame of 1994 US census income from 48,842 people divided into a training set of 32,561 and an independent test set of 16,281. The training outcome variable `y` (`yt` for test) is binary and indicates whether or not a person's income is greater than $50,000 per year. There are 12 predictor variables `x` (`xt` for test) consisting of various demographic and financial properties associated with each person. It also included estimates of $Pr(y = 1|x)$ obtained by several machine learning methods: gradient boosting on logistic scale using maximum likelihood (GBL), random forest (RF), and gradient boosting on the probability scale (GBP) using least–squares.

Usage

```
census
```

Format

`census`:

A list of 10 items.

- **x** training data frame of 32561 observations on 12 predictor variables
- **y** training binary response whether salary is above $50K or not
- **xt** test data frame of 16281 observations predictor variables
- **yt** test binary response whether salary is above $50K or not
- **gbl** training GBL response variable
- **gblt** test GBL response variable
- **gbp** training GBP response variable
- **gbpt** test GBP response variable
- **rf** training RF response probabilities
- **rft** test GBP response probabilities

Source

https://archive.ics.uci.edu/ml/datasets/census+income

`contrast` *Build contrast tree*

Description

Build contrast tree

Build boosted contrast tree model

Bootstrap contrast trees

Usage

```
contrast(
  x,
  y,
  z,
  w = rep(1, nrow(x)),
  cat.vars = NULL,
  not.used = NULL,
  qint = 10,
  xmiss = 9e+35,
  tree.size = 10,
  min.node = 500,
  mode = c("onesamp", "twosamp"),
  type = "dist",
  pwr = 2,
  quant = 0.5,
  nclass = NULL,
  costs = NULL,
  cdfsamp = 500,
  verbose = FALSE,
  tree.store = 1e+06,
  cat.store = 1e+05,
  nbump = 1,
  fnodes = 0.25,
  fsamp = 1,
  doprint = FALSE
)

modtrast(
  x,
  y,
  z,
  w = rep(1, nrow(x)),
```

```
  cat.vars = NULL,
  not.used = NULL,
  qint = 10,
  xmiss = 9e+35,
  tree.size = 10,
  min.node = 500,
  learn.rate = 0.1,
  type = c("dist", "diff", "class", "quant", "prob", "maxmean", "diffmean"),
  pwr = 2,
  quant = 0.5,
  cdfsamp = 500,
  verbose = FALSE,
  tree.store = 1e+06,
  cat.store = 1e+05,
  nbump = 1,
  fnodes = 0.25,
  fsamp = 1,
  doprint = FALSE,
  niter = 100,
  doplot = FALSE,
  span = 0,
  plot.span = 0.15,
  print.itr = 10
)

bootcri(
  x,
  y,
  z,
  w = rep(1, nrow(x)),
  cat.vars = NULL,
  not.used = NULL,
  qint = 10,
  xmiss = 9e+35,
  tree.size = 10,
  min.node = 500,
  mode = "onesamp",
  type = "dist",
  pwr = 2,
  quant = 0.5,
  nclass = NULL,
  costs = NULL,
  cdfsamp = 500,
  verbose = FALSE,
  tree.store = 1e+06,
  cat.store = 1e+05,
  nbump = 100,
  fnodes = 1,
```

```
  fsamp = 1,
  doprint = FALSE
)
```

Arguments

`x`	training input predictor data matrix or data frame. Rows are observations and columns are variables. Must be a numeric matrix or a data frame.
`y`	vector, or matrix containing training data input outcome values or censoring intervals for each observation. if y is a vector then it implies that y uncensored outcome values or other contrasting quantity. If y is a matrix, then then y is assumed to be censoring intervals for each observation; see details below
`z`	vector containing values of a second contrasting quantity for each observation
`w`	training observation weights
`cat.vars`	vector of column labels (numbers or names) indicating categorical variables (factors). All variables not so indicated are assumed to be orderable numeric; see details below
`not.used`	vector of column labels (numbers or names) indicating predictor variables not to be used in the model
`qint`	maximum number of split evaluation points on each predictor variable
`xmiss`	missing value flag. Must be numeric and larger than any non missing predictor/abs(response) variable value. Predictor variable values greater than or equal to xmiss are regarded as missing. Predictor variable data values of `NA` are internally set to the value of xmiss and thereby regarded as missing
`tree.size`	maximum number of terminal nodes in generated trees
`min.node`	minimum number of training observations in each tree terminal node
`mode`	indicating one or two-sample contrast; see details below for how it works with type
`type`	type of contrast; see details below for how it works with mode
`pwr`	center split bias parameter. Larger values produce less center split bias.
`quant`	specified quantile p (type='quant' only)
`nclass`	number of classes (type ='class' only) default=2
`costs`	nclass by nclass misclassification cost matrix (type='class' only); default is equal valued diagonal (error rate)
`cdfsamp`	= maximum subsample size used to compute censored CDF (censoring only)
`verbose`	a logical flag indicating print/don't print censored CDF computation progress, default `FALSE`
`tree.store`	size of internal tree storage. Decrease value in response to memory allocation error. Increase value for very large values of max.trees and/or tree.size, or in response to diagnostic message or erratic program behavior
`cat.store`	size of internal categorical value storage. Decrease value in response to memory allocation error. Increase value for very large values of max.trees and/or tree.size in the presence of many categorical variables (factors) with many levels, or in response to diagnostic message or erratic program behavior

`nbump`	number of bootstrap replications
`fnodes`	top fraction of node criteria used to evaluate trial bumped trees
`fsamp`	fraction of observations used in each bootstrap sample for bumped trees
`doprint`	logical flag `TRUE/FALSE` implies do/don't plot iteration progress
`learn.rate`	learning rate parameter in `(0,1]`
`niter`	number of trees
`doplot`	a flag to display/not display graphical plots
`span`	span for qq-plot transformation smoother
`plot.span`	running median smoother span for discrepancy plot (`doplot = TRUE`, only)
`print.itr`	tree discrepancy printing iteration interval

Details

The varible `xmiss` is the missing value flag, Must be numeric and larger than any non missing predictor/abs(response) variable value. Predictor variable values greater than or equal to `xmiss` are regarded as missing. Predictor variable data values of `NA` are internally set to the value of xmiss and thereby regarded as missing.

If the response y is a matrix, it is assumed to contain censoring intervals for each observation. Rows are observations.

- First/second column are lower/upper boundary of censoring interval (Can be same value for uncensored observations) respectively
- `y[,1] = -xmiss` implies outcome less than or equal to `y[,2]` (censored from above)
- `y[,2] = xmiss` implies outcome greater than or equal to `y[,1]`

Note that censoring is only allowed for `type='dist'`; see further below.

If `x` is a data frame and `cat.vars` (the columns indicating categorical variables), is missing, then components of type factor are treated as categorical variables. Ordered factors should be input as type numeric with appropriate numerical scores. If `cat.vars` is present it will over ride the data frame typing.

The `mode` argument is either

- `'onesamp'` (default) meaning one `x`-vector for each `(x,z)` pair
- `'twosamp'` implies two-sample contrast with
  - `x` are predictor variables for both samples
  - `y` are outcomes for both samples
  - `z` is sample identity flag with `z < 0` implying first sample observations and `z > 0`, the second sample observations. The `type` argument indicates the type of contrast. It can be either a user defined function or a string. If `mode` is `'onesamp'`, the default,
- `type = 'dist'` (default) implies contrast distribution of `y` with that of `z` (`y` may be censored - see above)
- `type = 'diff'` implies contrast joint paired values of `y` and `z`
- `type = 'class'` implies classification: contrast class labels `y[i]` and `z[i]` are two class labels (in `1:nclass`) for each observation.

- `type = 'prob'` implies contrast predicted with empirical probabilities: `y[i] = 0/1` and `z[i]` is predicted probability $P(y = 1)$ for i-th observation
- `type = 'quant'` is contrast predicted with empirical quantiles: `y[i]` is outcome value for i-th observation and `z[i]` is predicted p-th quantile value (see below) for i-th observation $(0 < p < 1)$
- `type = 'diffmean'` implies maximize absolute mean difference between `y` and `z`
- `type = 'maxmean'` implies maximize signed mean difference between `y` and `z`

When mode is `'twosamp'`

- `type= 'dist'` (default) implies contrast `y` distributions of both samples
- `type = 'diffmean'` implies maximize absolute difference between means of two samples
- `type = 'maxmean'` maximize signed difference between means of two samples

When `type` is a function, it must be a function of three arguments `f(y,z,w)` where `y` and `z` are double vectors and `w` is a weight vector, not necessarily normalized. The function should return a double vector of length 1 as the result. See example below.

Value

a contrast model object use as input to interpretation procedures

a contrast model object to be used with predtrast()

a named list with `out$bcri` the bootstraped discrepancy values

Author(s)

Jerome H. Friedman

References

Jerome H. Friedman (2020). `doi:10.1073/pnas.1921562117`

Examples

```
data(census, package = "conTree")
dx <- 1:10000; dxt <- 10001:16281;
# Build contrast tree
tree <- contrast(census$xt[dx,], census$yt[dx], census$gblt[dx], type = 'prob')
# Summarize tree
treesum(tree)
# Get terminal node identifiers for regions containing observations 1 through 10
getnodes(tree, x = census$xt[1:10, ])
# Plot nodes
nodeplots(tree, x = census$xt[dx, ], y = census$yt[dx], z = census$gblt[dx])
# Summarize contrast tree against (precomputed) gradient boosting
# on logistic scale using maximum likelihood (GBL)
nodesum(tree, census$xt[dxt,], census$yt[dxt], census$gblt[dxt])
# Use a custom R discrepancy function to build a contrast tree
dfun <- function(y, z, w) {
   w  <- w / sum(w)
```

```
    abs(sum(w * (y - z)))
}
tree2 <- contrast(census$xt[dx,], census$yt[dx], census$gblt[dx], type = dfun)
nodesum(tree2, census$xt[dxt,], census$yt[dxt], census$gblt[dxt])
# Generate lack of fit curve
lofcurve(tree, census$xt[dx,], census$yt[dx], census$gblt[dx])
# Build contrast tree boosting models
# Use small # of iterations for illustration (typically >= 200)
modgbl = modtrast(census$x, census$y, census$gbl, type = 'prob', niter = 10)
# Plot model accuracy as a function of iteration number
xval(modgbl, census$x, census$y, census$gbl, col = 'red')
# Produce predictions from modtrast() for new data.
ypred <- predtrast(modgbl, census$xt, census$gblt, num = modgbl$niter)
# Produce distribution boosting estimates
yhat <- predtrast(modgbl, census$xt, census$gblt, num = modgbl$niter)
```

`getnodes` *Get terminal node observation assignments*

Description

Get terminal node observation assignments

Usage

```
getnodes(tree, x)
```

Arguments

`tree`	model object output from contrast() or prune()
`x`	training input predictor data matrix or data frame in same format as in contrast()

Value

vector of tree terminal node identifiers (numbers) corresponding to each observation (row of x)

See Also

`contrast()`

`lofcurve` *Produce lack-of-fit curve for a contrast tree*

Description

Produce lack-of-fit curve for a contrast tree

Usage

```
lofcurve(
  tree,
  x,
  y,
  z,
  w = rep(1, length(y)),
  doplot = "first",
  col = "black",
  ylim = NULL
)
```

Arguments

`tree`	model object output from contrast() or prune()
`x`	training input predictor data matrix or data frame in same format as in contrast()
`y`	vector, or matrix containing training data input outcome values or censoring intervals for each observation in same format as in contrast()
`z`	vector containing values of a second contrasting quantity for each observation in same observation format as in contrast ()
`w`	observation weights
`doplot`	logical flag. doplot="first" implies start new display. doplot="next" implies super impose plot on existing display. doplot="none" implies no plot displayed.
`col`	color of plotted curve
`ylim`	y-axis limit

Value

a named list of plotted `x` and `y` points

`nodesum` *Summarize contrast tree*

Description

Summarize contrast tree

Show graphical terminal node summaries

Usage

```
nodesum(tree, x, y, z, w = rep(1, nrow(x)), doplot = FALSE)

nodeplots(
  tree,
  x,
  y,
  z,
  w = rep(1, nrow(x)),
  nodes = NULL,
  xlim = NULL,
  ylim = NULL,
  pts = "FALSE",
  span = 0.15
)
```

Arguments

`tree`	model object output from contrast() or prune()
`x`	training input predictor data matrix or data frame in same format as in contrast()
`y`	vector, or matrix containing training data input outcome values or censoring intervals for each observation in same format as in contrast()
`z`	vector containing values of a second contrasting quantity for each observation in same observation format as in contrast()
`w`	observation weights
`doplot`	a flag to display/not display plots of output quantities
`nodes`	selected tree terminal node identifiers. Default is all terminal nodes
`xlim`	x-axis limit
`ylim`	y-axis limit
`pts`	logical flag indicating whether to show y-values as circles/points (`type = 'pp'` only)
`span`	running median smoother span (`type = 'diff'` only)

Details

The graphical representations of terminal node contrasts depend on the tree type graphical representations of terminal node contrasts depending on tree type `-type = 'dist'` implies CDFs of y and z in each terminal node. (Only top nine nodes are shown). Note that y can be censored (see above) `-type = 'diff'` implies plot y versus z in each terminal node. (Only top nine nodes are shown). `-type = 'class'` implies barplot of misclassification risk (upper) amd total weight (lower) in each terminal node `-type = 'prob'` implies upper barplot contrasting empirical (blue) and predicted (red) $p(y = 1)$ in each terminal node. Lower barplot showing total weight in each terminal node.

- type = 'quant' => upper barplot of fraction of y-values greater than or equal to corresponding z-values (quantile prediction) in each terminal node. Horizontal line reflects specified target quantile. Lower barplot showing total weight in each terminal node.
- `type = 'diffmean'` or `type = 'maxmean'` implies upper barplot contrasting y-mean (blue) and z-mean (red) in each terminal node. Lower barplot showing total weight in each terminal node.

Value

a named list of four items:

- `nodes` the tree terminal node identifiers
- `cri` the terminal node criterion values (depends on contrast type see above)
- `wt` sum of weights in each terminal node
- `avecri` weighted criterion average over all terminal nodes

See Also

`contrast()`

`onesample_parameters` *Return the one sample parameters used in fortran discrepancy functions*

Description

These functions are mostly useful when one wants to test one's own discrepancy function in R `f(y, z, w)` to determine if the results are correct. So a natural test is to experiment by programming one of the already implemented discrepancy functions in R. However, the Fortran implementations of such discrepancy measures use some parameters in the computations and therefore the returned results from a simple R implementation may not exactly match. Using these parameters, one can ensure that they do. These are to be interpreted as follows. For one sample, the `type = "dist"` implementation in the package returns 0 if the length of `y` is less than `nmin` which is (100L). The `eps = 1.0e-5` parameter is used to ensure that the denominator in the formula for the Anderson-Darling statistic is at least `eps`. Next, for `type = "prob"`, if the length of the vector is less than `nmin = 20` the discrepancy is computed to be 0. And so on. Refer to the R and Fortran source for further details as this is an advanced topic.

Usage

```
onesample_parameters()

twosample_parameters()
```

Value

a named list for each of the types.

`predtrast` *Predict y-values from boosted contrast model*

Description

Predict y-values from boosted contrast model

Usage

```
predtrast(model, x, z, num = model$niter)
```

Arguments

`model`	model object output from modtrast()
`x`	x-values for new data
`z`	z-values for new data
`num`	number of trees used to compute model values

Value

predicted y-values for new data from model

See Also

`contrast()`

`prune` *Prune a contrast tree*

Description

Prune a contrast tree

Usage

```
prune(tree, thr = 0.1)
```

Arguments

`tree`	a tree model object output from contrast
`thr`	a split improvement threshold, default is 0.1

Value

a bottom-up pruned tree with splits corresponding to improvement less than threshold `thr` removed

---

`prune.seq` *Show all possible pruned subtrees*

Description

Show all possible pruned subtrees

Usage

```
prune.seq(tree, eps = 0.01, plot.it = TRUE)
```

Arguments

`tree`	a tree model object output from contrast
`eps`	small increment defining grid of threshold values
`plot.it`	a logical flag indicating plot/don't plot of number of nodes versus threshold value for all pruned subtrees, default `TRUE`

Value

a named list of two items:

- `thr` a set of threshold values that sequentially reduce tree size
- `nodes` the corresponding tree sizes (number of terminal nodes)

`save_rfun` *Save the function f for calling from fortran*

Description

Save the function f for calling from fortran

Usage

```
save_rfun(f)
```

Arguments

`f`	the R function to be called using `.Fortran`

Value

TRUE, invisibly.

`treesum` *Print terminal node x-region boundaries*

Description

Print terminal node x-region boundaries

Usage

```
treesum(tree, nodes = NULL)
```

Arguments

`tree`	model object output from contrast() or prune()
`nodes`	vector of terminal node identifiers for the tree specifying the desired regions. The default is all terminal nodes.

Details

The predictor variable x-boundaries defining each terminal node are printed.

For numeric variables: variable | sign | value

- sign + => value=lower boundary on variable
- sign - => value upper boundary on variable

For categorical variables: cat variable | sign | set of values

- sign + => values in node
- sign - => values not in node (compliment values in node) graphical representations of terminal node contrasts depend on the tree type

Value

No return value (invisble `NULL`)

See Also

`contrast()`

`xval` *Cross-validate boosted contrast tree boosted with (new) data*

Description

Cross-validate boosted contrast tree boosted with (new) data

Usage

```
xval(
  mdl,
  x,
  y,
  z,
  num = length(mdl$tree),
  del = 10,
  span = 0.15,
  ylab = "Average  Discrepancy",
  doplot = "first",
  doprint = FALSE,
  col = "red"
)
```

Arguments

`mdl`	model output from modtrast()
`x`	data predictor variables is same format as input to modtrast
`y`	data y values is same format as input to modtrast
`z`	data z values is same format as input to modtrast
`num`	number of trees used to compute model values
`del`	plot discrepancy value computed every del-th iteration (tree)
`span`	running median smoother span (`doplot=TRUE`, only)
`ylab`	graphical parameter (‘doplot="first", only)
`doplot`	logical flag. doplot="first" implies start new display. doplot="next" implies super impose plot on existing display. doplot="none" implies no plot displayed.
`doprint`	logical flag `TRUE/FALSE` implies do/don’t print progress while executing, default `FALSE`
`col`	color of plotted curve

Value

a named list of two items: `ntree` the iteration numbers, and `error` the corresponding discrepancy values

See Also

`contrast()`

`ydist` *Transform z-values t(z) such that the distribution of $p(t(z)|x)$ approximates $p(t(y|x)$ for type = 'dist' only*

Description

Transform z-values t(z) such that the distribution of $p(t(z)|x)$ approximates $p(t(y|x)$ for type = 'dist' only

Usage

```
ydist(model, x, z, num = model$niter)
```

Arguments

`model`	model object output from modtrast()
`x`	vector of predictor variable values for a (single) observation
`z`	sample of z-values drawn from $p(z|x)$
`num`	number of trees used to compute model values

Value

vector of `length(z)` containing transformed values t(z) approximating $p(y|x)$

See Also

`contrast()`

Index