

# Package ‘drf’

March 29, 2021

**Title** Distributional Random Forests

**Version** 1.1.0

**Author** Loris Michel, Domagoj Cevic

**Maintainer** Loris Michel <michel@stat.math.ethz.ch>

**BugReports** <https://github.com/lorismichel/drf/issues>

**Description** An implementation of distributional random forests as introduced in Cevic & Michel & Meinshausen & Buhlmann (2020) <arXiv:2005.14458>.

**License** GPL-3

**LinkingTo** Rcpp, RcppEigen

**Depends** R (>= 3.4.4)

**Imports** stats, fastDummies, Matrix, methods, Rcpp (>= 0.12.15),  
transport

**RoxygenNote** 7.1.1

**Suggests** DiagrammeR

**SystemRequirements** GNU make

**URL** <https://github.com/lorismichel/drf>

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2021-03-29 09:30:05 UTC

## R topics documented:

drf . . . . .	2
get_sample_weights . . . . .	5
get_tree . . . . .	6
leaf_stats.default . . . . .	7
leaf_stats.drf . . . . .	7
medianHeuristic . . . . .	8
plot.drf_tree . . . . .	8
predict.drf . . . . .	9

print.drf . . . . .	11
print.drf_tree . . . . .	11
split_frequencies . . . . .	12
variableImportance . . . . .	12
variable_importance . . . . .	13
weighted.quantile . . . . .	14
<b>Index</b>	<b>15</b>

---

drf                                      *Distributional Random Forests*

---

## Description

Trains a distributional random forest that can be used to estimate statistical functional  $F(P(Y | X))$  for possibly multivariate response  $Y$ .

## Usage

```
drf(
  X,
  Y,
  num.trees = 500,
  splitting.rule = "FourierMMD",
  num.features = 10,
  bandwidth = NULL,
  response.scaling = TRUE,
  node.scaling = FALSE,
  sample.weights = NULL,
  clusters = NULL,
  equalize.cluster.weights = FALSE,
  sample.fraction = 0.5,
  mtry = min(ceiling(sqrt(ncol(X)) + 20), ncol(X)),
  min.node.size = 15,
  honesty = TRUE,
  honesty.fraction = 0.5,
  honesty.prune.leaves = TRUE,
  alpha = 0.05,
  imbalance.penalty = 0,
  ci.group.size = 2,
  compute.oob.predictions = TRUE,
  num.threads = NULL,
  seed = stats::runif(1, 0, .Machine$integer.max),
  compute.variable.importance = FALSE
)
```

**Arguments**

<code>X</code>	The covariates used in the regression. Can be either a matrix of numerical values, or a <code>data.frame</code> with characters and factors. In the latter case, one-hot-encoding will be implicitly used.
<code>Y</code>	The (multivariate) outcome. A matrix or <code>data.frame</code> of numeric values.
<code>num.trees</code>	Number of trees grown in the forest. Default is 500.
<code>splitting.rule</code>	a character value. The type of splitting rule used, can be either "CART" or "FourierMMD".
<code>num.features</code>	a numeric value, in case of "FourierMMD", the number of random features to sample.
<code>bandwidth</code>	a numeric value, the bandwidth of the Gaussian kernel used in case of "FourierMMD", by default the value is NULL and the median heuristic is used.
<code>response.scaling</code>	a boolean value, should the responses be globally scaled at first.
<code>node.scaling</code>	a boolean value, should the responses be scaled or not by node.
<code>sample.weights</code>	(experimental) Weights given to an observation in estimation. If NULL, each observation is given the same weight. Default is NULL.
<code>clusters</code>	Vector of integers or factors specifying which cluster each observation corresponds to. Default is NULL (ignored).
<code>equalize.cluster.weights</code>	If FALSE, each unit is given the same weight (so that bigger clusters get more weight). If TRUE, each cluster is given equal weight in the forest. In this case, during training, each tree uses the same number of observations from each drawn cluster: If the smallest cluster has K units, then when we sample a cluster during training, we only give a random K elements of the cluster to the tree-growing procedure. When estimating average treatment effects, each observation is given weight 1/cluster size, so that the total weight of each cluster is the same. Note that, if this argument is FALSE, sample weights may also be directly adjusted via the <code>sample.weights</code> argument. If this argument is TRUE, <code>sample.weights</code> must be set to NULL. Default is FALSE.
<code>sample.fraction</code>	Fraction of the data used to build each tree. Note: If <code>honesty = TRUE</code> , these subsamples will further be cut by a factor of <code>honesty.fraction</code> . Default is 0.5.
<code>mtry</code>	Number of variables tried for each split. Default is $\sqrt{p} + 20$ where p is the number of variables.
<code>min.node.size</code>	A target for the minimum number of observations in each tree leaf. Note that nodes with size smaller than <code>min.node.size</code> can occur, as in the original random-Forest package. Default is 5.
<code>honesty</code>	Whether to use honest splitting (i.e., sub-sample splitting). Default is TRUE. For a detailed description of <code>honesty</code> , <code>honesty.fraction</code> , <code>honesty.prune.leaves</code> , and recommendations for parameter tuning, see the <a href="#">grf</a> reference for more information (initial source) <a href="#">algorithm reference</a> .

<code>honesty.fraction</code>	The fraction of data that will be used for determining splits if <code>honesty = TRUE</code> . Corresponds to set J1 in the notation of the paper. Default is 0.5 (i.e. half of the data is used for determining splits).
<code>honesty.prune.leaves</code>	If <code>TRUE</code> , prunes the estimation sample tree such that no leaves are empty. If <code>FALSE</code> , keep the same tree as determined in the splits sample (if an empty leave is encountered, that tree is skipped and does not contribute to the estimate). Setting this to <code>FALSE</code> may improve performance on small/marginally powered data, but requires more trees (note: tuning does not adjust the number of trees). Only applies if <code>honesty</code> is enabled. Default is <code>TRUE</code> .
<code>alpha</code>	A tuning parameter that controls the maximum imbalance of a split. Default is 0.05.
<code>imbalance.penalty</code>	A tuning parameter that controls how harshly imbalanced splits are penalized. Default is 0.
<code>ci.group.size</code>	The forest will grow <code>ci.group.size</code> trees on each subsample. In order to provide confidence intervals, <code>ci.group.size</code> must be at least 2. Default is 2.
<code>compute.oob.predictions</code>	Whether OOB predictions on training set should be precomputed. Default is <code>TRUE</code> .
<code>num.threads</code>	Number of threads used in training. By default, the number of threads is set to the maximum hardware concurrency.
<code>seed</code>	The seed of the C++ random number generator.
<code>compute.variable.importance</code>	boolean, should the variable importance be computed in the object.

## Value

A trained distributional random forest object.

## Examples

```
# Train a distributional random forest with CART splitting rule.
n <- 100
p <- 2
X <- matrix(rnorm(n * p), n, p)
Y <- X + matrix(rnorm(n * p), ncol=p)
drf.forest <- drf(X = X, Y = Y)

# Predict conditional correlation.
X.test <- matrix(0, 101, p)
X.test[, 1] <- seq(-2, 2, length.out = 101)
cor.pred <- predict(drf.forest, X.test, functional = "cor")

# Predict on out-of-bag training samples.
cor.oob.pred <- predict(drf.forest, functional = "cor")

# Train a distributional random forest with "FourierMMD" splitting rule.
```

```

n <- 100
p <- 2
X <- matrix(rnorm(n * p), n, p)
Y <- X + matrix(rnorm(n * p), ncol=p)
drf.forest <- drf(X = X, Y = Y, splitting.rule = "FourierMMD", num.features = 10)

# Predict conditional correlation.
X.test <- matrix(0, 101, p)
X.test[, 1] <- seq(-2, 2, length.out = 101)
cor.pred <- predict(drf.forest, X.test, functional = "cor")

# Predict on out-of-bag training samples.
cor.oob.pred <- predict(drf.forest, functional = "cor")

```

---

get_sample_weights	<i>Given a trained forest and test data, compute the training sample weights for each test point.</i>
--------------------	---

---

## Description

During normal prediction, these weights are computed as an intermediate step towards producing estimates. This function allows for examining the weights directly, so they could be potentially be used as the input to a different analysis.

## Usage

```
get_sample_weights(forest, newdata = NULL, num.threads = NULL)
```

## Arguments

forest	The trained forest.
newdata	Points at which predictions should be made. If NULL, makes out-of-bag predictions on the training set instead (i.e., provides predictions at $X_i$ using only trees that did not use the $i$ -th training example).#’ @param max.depth Maximum depth of splits to consider.
num.threads	Number of threads used in training. If set to NULL, the software automatically selects an appropriate amount.

## Value

A sparse matrix where each row represents a test sample, and each column is a sample in the training data. The value at  $(i, j)$  gives the weight of training sample  $j$  for test sample  $i$ .

**Examples**

```
## Not run:
p <- 10
n <- 100
X <- matrix(2 * runif(n * p) - 1, n, p)
Y <- (X[, 1] > 0) + 2 * rnorm(n)
rrf <- drf(X, matrix(Y, ncol=1), mtry = p)
sample.weights.oob <- get_sample_weights(rrf)

n.test <- 15
X.test <- matrix(2 * runif(n.test * p) - 1, n.test, p)
sample.weights <- get_sample_weights(rrf, X.test)

## End(Not run)
```

---

get\_tree

*Retrieve a single tree from a trained forest object.*


---

**Description**

Retrieve a single tree from a trained forest object.

**Usage**

```
get_tree(forest, index)
```

**Arguments**

forest	The trained forest.
index	The index of the tree to retrieve.

**Value**

A DRF tree object containing the below attributes. `drawn_samples`: a list of examples that were used in training the tree. This includes examples that were used in choosing splits, as well as the examples that populate the leaf nodes. Put another way, if `honesty` is enabled, this list includes both subsamples from the split (`J1` and `J2` in the notation of the paper). `num_samples`: the number of examples used in training the tree. `nodes`: a list of objects representing the nodes in the tree, starting with the root node. Each node will contain an `'is_leaf'` attribute, which indicates whether it is an interior or leaf node. Interior nodes contain the attributes `'left_child'` and `'right_child'`, which give the indices of their children in the list, as well as `'split_variable'`, and `'split_value'`, which describe the split that was chosen. Leaf nodes only have the attribute `'samples'`, which is a list of the training examples that the leaf contains. Note that if `honesty` is enabled, this list will only contain examples from the second subsample that was used to `'repopulate'` the tree (`J2` in the notation of the paper).

**Examples**

```
## Not run:
# Train a quantile forest.
n <- 50
p <- 10
X <- matrix(rnorm(n * p), n, p)
Y <- X[, 1] * rnorm(n)
q.forest <- quantile_forest(X, Y, quantiles = c(0.1, 0.5, 0.9))

# Examine a particular tree.
q.tree <- get_tree(q.forest, 3)
q.tree$nodes

## End(Not run)
```

---

leaf_stats.default	<i>A default leaf_stats for forests classes without a leaf_stats method that always returns NULL.</i>
--------------------	---

---

**Description**

A default leaf\_stats for forests classes without a leaf\_stats method that always returns NULL.

**Usage**

```
## Default S3 method:
leaf_stats(forest, samples, ...)
```

**Arguments**

forest	Any forest
samples	The samples to include in the calculations.
...	Additional arguments (currently ignored).

---

leaf_stats.drf	<i>Calculate summary stats given a set of samples for regression forests.</i>
----------------	---

---

**Description**

Calculate summary stats given a set of samples for regression forests.

**Usage**

```
## S3 method for class 'drf'
leaf_stats(forest, samples, ...)
```

**Arguments**

forest	The GRF forest
samples	The samples to include in the calculations.
...	Additional arguments (currently ignored).

**Value**

A named vector containing summary stats

---

medianHeuristic	<i>Compute the median heuristic for the MMD bandwidth choice</i>
-----------------	--

---

**Description**

Compute the median heuristic for the MMD bandwidth choice

**Usage**

```
medianHeuristic(Y)
```

**Arguments**

Y	the response matrix
---	---------------------

**Value**

the median heuristic

---

plot.drf_tree	<i>Plot a DRF tree object.</i>
---------------	--------------------------------

---

**Description**

Plot a DRF tree object.

**Usage**

```
## S3 method for class 'drf_tree'
plot(x, ...)
```

**Arguments**

x	The tree to plot
...	Additional arguments (currently ignored).



---

predict.drf	<i>Predict with a drf forest</i>
-------------	----------------------------------

---

**Description**

Predict with a drf forest

**Usage**

```
## S3 method for class 'drf'
predict(
  object,
  newdata = NULL,
  transformation = NULL,
  functional = NULL,
  num.threads = NULL,
  custom.functional = function(y, w) apply(y, 2, sum(y * w)),
  ...
)
```

**Arguments**

object	The trained drf forest.
newdata	Points at which predictions should be made. If NULL, makes out-of-bag predictions on the training set instead (i.e., provides predictions at $X_i$ using only trees that did not use the $i$ -th training example). Note that this matrix (or vector) should have the number of columns as the training matrix, and that the columns must appear in the same order.
transformation	a function giving a transformation of the responses, by default if NULL, the identity function( $y$ ) $y$ is used.
functional	<p>which type of statistical functional. One option between:</p> <ul style="list-style-type: none"> <li>• "mean"the conditional mean, the returned value is a list containing a matrix mean of size <math>n \times f</math>, where <math>n</math> denotes the number of observation in newdata and <math>f</math> the dimension of the transformation.</li> <li>• "sd"the conditional standard deviation, the returned value is a list containing a matrix sd of size <math>n \times f</math>, where <math>n</math> denotes the number of observation in newdata and <math>f</math> the dimension of the transformation.</li> <li>• "quantile"the conditional quantiles, the returned value is a list containing an array quantile of size <math>n \times f \times q</math>, where <math>n</math> denotes the number of observation in newdata, <math>f</math> the dimension of the transformation and <math>q</math> the number of desired quantiles.</li> <li>• "cor"the conditional correlation, the returned value is a list containing an array cor of size <math>n \times f \times f</math>, where <math>n</math> denotes the number of observation in newdata, <math>f</math> the dimension of the transformation.</li> </ul>

- "cov"the conditional covariance, the returned value is a list containing an array cor of size  $n \times f \times f$ , where  $n$  denotes the number of observation in newdata,  $f$  the dimension of the transformation.
  - "normalPredictionScore"a prediction score based on an asymptotic normality assumption, the returned value is a list containing a list of functions normalPredictionScore of size  $n$ , where  $n$  denotes the number of observation in newdata. Here the transformation should be uni-dimensional.
  - "custom"a custom function provided by the user, the returned value is a list containing a matrix custom of size  $n \times f$ , where  $n$  denotes the number of observation in newdata and  $f$  the dimension of the output of the function custom.functional.
  - "MQ"multivariate quantiles, return a list containing a matrix of the inputed ranks  $u$  (that should be provided as an argument of the predict function) along with a list of the different corresponding MQ (same size as  $u$ ).
- num.threads      Number of threads used in training. If set to NULL, the software automatically selects an appropriate amount.
- custom.functional  
a function giving the custom functional when functional is set to "custom". This should be a function  $f(y,w)$  using the training response matrix  $y$  and the weights  $w$  at a single testing point.
- ...                additional parameters.

### Value

a list containing an entry with the same name as the functional selected.

### Examples

```
# Train a distributional random forest with CART splitting rule.
n <- 100
p <- 2
X <- matrix(rnorm(n * p), n, p)
Y <- X + matrix(rnorm(n * p), ncol=p)
drf.forest <- drf(X = X, Y = Y)

# Predict conditional correlation.
X.test <- matrix(0, 101, p)
X.test[, 1] <- seq(-2, 2, length.out = 101)
cor.pred <- predict(drf.forest, X.test, functional = "cor")

# Predict on out-of-bag training samples.
cor.oob.pred <- predict(drf.forest, functional = "cor")

# Train a distributional random forest with "FourierMMD" splitting rule.
n <- 100
p <- 2
X <- matrix(rnorm(n * p), n, p)
Y <- X + matrix(rnorm(n * p), ncol=p)
drf.forest <- drf(X = X, Y = Y, splitting.rule = "FourierMMD", num.features = 10)
```

```
# Predict conditional correlation.
X.test <- matrix(0, 101, p)
X.test[, 1] <- seq(-2, 2, length.out = 101)
cor.pred <- predict(drf.forest, X.test, functional = "cor")

# Predict on out-of-bag training samples.
cor.oob.pred <- predict(drf.forest, functional = "cor")
```

---

```
print.drf          Print a DRF forest object.
```

---

### Description

Print a DRF forest object.

### Usage

```
## S3 method for class 'drf'
print(x, decay.exponent = 2, max.depth = 4, ...)
```

### Arguments

x	The tree to print.
decay.exponent	A tuning parameter that controls the importance of split depth.
max.depth	The maximum depth of splits to consider.
...	Additional arguments (currently ignored).

---

```
print.drf_tree     Print a DRF tree object.
```

---

### Description

Print a DRF tree object.

### Usage

```
## S3 method for class 'drf_tree'
print(x, ...)
```

### Arguments

x	The tree to print.
...	Additional arguments (currently ignored).

---

split_frequencies	<i>Calculate which features the forest split on at each depth.</i>
-------------------	--

---

**Description**

Calculate which features the forest split on at each depth.

**Usage**

```
split_frequencies(forest, max.depth = 4)
```

**Arguments**

forest	The trained forest.
max.depth	Maximum depth of splits to consider.

**Value**

A matrix of split depth by feature index, where each value is the number of times the feature was split on at that depth.

**Examples**

```
## Not run:
# Train a quantile forest.
n <- 50
p <- 10
X <- matrix(rnorm(n * p), n, p)
Y <- X[, 1] * rnorm(n)
q.forest <- quantile_forest(X, Y, quantiles = c(0.1, 0.5, 0.9))

# Calculate the split frequencies for this forest.
split_frequencies(q.forest)

## End(Not run)
```

---

variableImportance	<i>Variable importance based on MMD</i>
--------------------	---

---

**Description**

compute an mmd-based variable importance for the drf fit.

**Usage**

```
variableImportance(
  object,
  h = NULL,
  response.scaling = TRUE,
  type = "difference"
)
```

**Arguments**

object	an S3 object of class drf.
h	the bandwidth parameter, default to NULL using then the median heuristic.
response.scaling	a boolean value indicating if the responses should be scaled globally beforehand.
type	the type of importance, could be either "raw", the plain MMD values, "relative", the ratios to the observed MMD or "difference", the excess to the observed MMD

**Value**

a vector of variable importance values.

---

variable\_importance     *Calculate a simple measure of 'importance' for each feature.*

---

**Description**

A simple weighted sum of how many times feature *i* was split on at each depth in the forest.

**Usage**

```
variable_importance(forest, decay.exponent = 2, max.depth = 4)
```

**Arguments**

forest	The trained forest.
decay.exponent	A tuning parameter that controls the importance of split depth.
max.depth	Maximum depth of splits to consider.

**Value**

A list specifying an 'importance value' for each feature.

**Examples**

```
## Not run:
# Train a quantile forest.
n <- 50
p <- 10
X <- matrix(rnorm(n * p), n, p)
Y <- X[, 1] * rnorm(n)
q.forest <- quantile_forest(X, Y, quantiles = c(0.1, 0.5, 0.9))

# Calculate the 'importance' of each feature.
variable_importance(q.forest)

## End(Not run)
```

---

weighted.quantile      *Weighted quantiles*

---

**Description**

Weighted quantiles

**Usage**

```
weighted.quantile(x, w, probs = seq(0, 1, 0.25), na.rm = TRUE)
```

**Arguments**

x	a vector of observations
w	a vector of weights
probs	the given probabilities for which we want to get quantiles
na.rm	should we remove missing values.

# Index

`drf`, [2](#)

`get_sample_weights`, [5](#)

`get_tree`, [6](#)

`leaf_stats.default`, [7](#)

`leaf_stats.drf`, [7](#)

`medianHeuristic`, [8](#)

`plot.drf_tree`, [8](#)

`predict.drf`, [9](#)

`print.drf`, [11](#)

`print.drf_tree`, [11](#)

`split_frequencies`, [12](#)

`variable_importance`, [13](#)

`variableImportance`, [12](#)

`weighted.quantile`, [14](#)