Package ‘arf’

February 24, 2025

Title Adversarial Random Forests

Version 0.2.4

Date 2025-02-24

Maintainer Marvin N. Wright <cran@wrig.de>

Description Adversarial random forests (ARFs) recursively partition data into fully factorized leaves, where features are jointly independent. The procedure is iterative, with alternating rounds of generation and discrimination. Data becomes increasingly realistic at each round, until original and synthetic samples can no longer be reliably distinguished. This is useful for several unsupervised learning tasks, such as density estimation and data synthesis. Methods for both are implemented in this package. ARFs naturally handle unstructured data with mixed continuous and categorical covariates. They inherit many of the benefits of random forests, including speed, flexibility, and solid performance with default parameters. For details, see Watson et al. (2023) <https://proceedings.mlr.press/v206/watson23a.html>.

License GPL (>= 3)

URL https://github.com/bips-hb/arf, https://bips-hb.github.io/arf/

BugReports https://github.com/bips-hb/arf/issues

Imports data.table, ranger, foreach, stringr, truncnorm

Encoding UTF-8

RoxygenNote 7.3.2

Suggests ggplot2, doParallel, doFuture, mlbench, knitr, rmarkdown, tibble, palmerpenguins, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Marvin N. Wright [aut, cre] (<https://orcid.org/0000-0002-8542-6291>),
David S. Watson [aut] (<https://orcid.org/0000-0001-9632-2159>),
Kristin Blesch [aut] (<https://orcid.org/0000-0001-6241-3079>),
Jan Kapar [aut] (<https://orcid.org/0009-0000-6408-2840>)

Repository CRAN

Date/Publication 2025-02-24 21:50:02 UTC

Contents

arf-package 2
adversarial_rf 3
darf 6
earf 7
expct 8
forde 10
forge 12
impute 15
lik 17
rarf 19

Index **21**

`arf-package` *arf: Adversarial Random Forests*

Description

Adversarial random forests (ARFs) recursively partition data into fully factorized leaves, where features are jointly independent. The procedure is iterative, with alternating rounds of generation and discrimination. Data becomes increasingly realistic at each round, until original and synthetic samples can no longer be reliably distinguished. This is useful for several unsupervised learning tasks, such as density estimation and data synthesis. Methods for both are implemented in this package. ARFs naturally handle unstructured data with mixed continuous and categorical covariates. They inherit many of the benefits of random forests, including speed, flexibility, and solid performance with default parameters. For details, see Watson et al. (2023) `https://proceedings.mlr.press/v206/watson23a.html`.

Author(s)

Maintainer: Marvin N. Wright `<cran@wrig.de>` (ORCID)

Authors:

- David S. Watson `<david.s.watson11@gmail.com>` (ORCID)
- Kristin Blesch (ORCID)
- Jan Kapar (ORCID)

See Also

adversarial_rf, forde, forge, expct, lik

Useful links:

- https://github.com/bips-hb/arf
- https://bips-hb.github.io/arf/
- Report bugs at https://github.com/bips-hb/arf/issues

Examples

```
# Train ARF and estimate leaf parameters
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)

# Generate 100 synthetic samples from the iris dataset
x_synth <- forge(psi, n_synth = 100)

# Condition on Species = "setosa" and Sepal.Length > 6
evi <- data.frame(Species = "setosa",
                  Sepal.Length = "(6, Inf)")
x_synth <- forge(psi, n_synth = 100, evidence = evi)

# Estimate average log-likelihood
ll <- lik(psi, iris, arf = arf, log = TRUE)
mean(ll)

# Expectation of Sepal.Length for class setosa
evi <- data.frame(Species = "setosa")
expct(psi, query = "Sepal.Length", evidence = evi)

## Not run:
# Parallelization with doParallel
doParallel::registerDoParallel(cores = 4)

# ... or with doFuture
doFuture::registerDoFuture()
future::plan("multisession", workers = 4)

## End(Not run)
```

---

adversarial_rf *Adversarial Random Forests*

Description

Implements an adversarial random forest to learn independence-inducing splits.

Usage

```
adversarial_rf(
  x,
  num_trees = 10L,
  min_node_size = 2L,
  delta = 0,
  max_iters = 10L,
  early_stop = TRUE,
  prune = TRUE,
  verbose = TRUE,
  parallel = TRUE,
  ...
)
```

Arguments

`x`	Input data. Integer variables are recoded as ordered factors with a warning. See Details.
`num_trees`	Number of trees to grow in each forest. The default works well for most generative modeling tasks, but should be increased for likelihood estimation. See Details.
`min_node_size`	Minimal number of real data samples in leaf nodes.
`delta`	Tolerance parameter. Algorithm converges when OOB accuracy is < 0.5 + `delta`.
`max_iters`	Maximum iterations for the adversarial loop.
`early_stop`	Terminate loop if performance fails to improve from one round to the next?
`prune`	Impose `min_node_size` by pruning?
`verbose`	Print discriminator accuracy after each round? Will also show additional warnings.
`parallel`	Compute in parallel? Must register backend beforehand, e.g. via `doParallel` or `doFuture`; see examples.
`...`	Extra parameters to be passed to `ranger`.

Details

The adversarial random forest (ARF) algorithm partitions data into fully factorized leaves where features are jointly independent. ARFs are trained iteratively, with alternating rounds of generation and discrimination. In the first instance, synthetic data is generated via independent bootstraps of each feature, and a RF classifier is trained to distinguish between real and fake samples. In subsequent rounds, synthetic data is generated separately in each leaf, using splits from the previous forest. This creates increasingly realistic data that satisfies local independence by construction. The algorithm converges when a RF cannot reliably distinguish between the two classes, i.e. when OOB accuracy falls below 0.5 + `delta`.

ARFs are useful for several unsupervised learning tasks, such as density estimation (see `forde`) and data synthesis (see `forge`). For the former, we recommend increasing the number of trees for improved performance (typically on the order of 100-1000 depending on sample size).

Integer variables are recoded with a warning (set `verbose = FALSE` to silence these). Default behavior is to convert integer variables with six or more unique values to numeric, while those with up to five unique values are treated as ordered factors. To override this behavior, explicitly recode integer variables to the target type prior to training.

Note: convergence is not guaranteed in finite samples. The `max_iters` argument sets an upper bound on the number of training rounds. Similar results may be attained by increasing `delta`. Even a single round can often give good performance, but data with strong or complex dependencies may require more iterations. With the default `early_stop = TRUE`, the adversarial loop terminates if performance does not improve from one round to the next, in which case further training may be pointless.

Value

A random forest object of class `ranger`.

References

Watson, D., Blesch, K., Kapar, J., & Wright, M. (2023). Adversarial random forests for density estimation and generative modeling. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pp. 5357-5375.

See Also

`arf`, `forde`, `forge`, `expct`, `lik`

Examples

```
# Train ARF and estimate leaf parameters
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)

# Generate 100 synthetic samples from the iris dataset
x_synth <- forge(psi, n_synth = 100)

# Condition on Species = "setosa" and Sepal.Length > 6
evi <- data.frame(Species = "setosa",
                  Sepal.Length = "(6, Inf)")
x_synth <- forge(psi, n_synth = 100, evidence = evi)

# Estimate average log-likelihood
ll <- lik(psi, iris, arf = arf, log = TRUE)
mean(ll)

# Expectation of Sepal.Length for class setosa
evi <- data.frame(Species = "setosa")
expct(psi, query = "Sepal.Length", evidence = evi)

## Not run:
# Parallelization with doParallel
doParallel::registerDoParallel(cores = 4)
```

```
# ... or with doFuture
doFuture::registerDoFuture()
future::plan("multisession", workers = 4)

## End(Not run)
```

---

darf *Shortcut likelihood function*

Description

Calls adversarial_rf, forde and lik. For repeated application, it is faster to save outputs of adversarial_rf and forde and pass them via ... or directly use lik.

Usage

```
darf(x, query = NULL, ...)
```

Arguments

x	Input data. Integer variables are recoded as ordered factors with a warning. See Details.
query	Data frame of samples, optionally comprising just a subset of training features. See Details of lik. Is set to x if zero.
...	Extra parameters to be passed to adversarial_rf, forde and lik.

Value

A vector of likelihoods, optionally on the log scale. A dataset of n_synth synthetic samples or of nrow(x) synthetic samples if n_synth is undefined.

References

Watson, D., Blesch, K., Kapar, J., & Wright, M. (2023). Adversarial random forests for density estimation and generative modeling. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pp. 5357-5375.

See Also

arf, adversarial_rf, forde, forge

Examples

```
# Estimate log-likelihoods
ll <- darf(iris)

# Partial evidence query
ll <- darf(iris, query = iris[1, 1:3])

# Condition on Species = "setosa"
ll <- darf(iris, query = iris[1, 1:3], evidence = data.frame(Species = "setosa"))
```

---

earf *Shortcut expectation function*

Description

Calls `adversarial_rf`, `forde` and `expct`. For repeated application, it is faster to save outputs of `adversarial_rf` and `forde` and pass them via `...` or directly use `expct`.

Usage

```
earf(x, ...)
```

Arguments

`x`	Input data. Integer variables are recoded as ordered factors with a warning. See Details.
`...`	Extra parameters to be passed to `adversarial_rf`, `forde` and `expct`.

Value

A one row data frame with values for all query variables.

References

Watson, D., Blesch, K., Kapar, J., & Wright, M. (2023). Adversarial random forests for density estimation and generative modeling. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pp. 5357-5375.

See Also

`arf`, `adversarial_rf`, `forde`, `expct`

Examples

```
# What is the expected values of each feature?
earf(iris)

#' # What is the expected values of Sepal.Length?
earf(iris, query = "Sepal.Length")

# What if we condition on Species = "setosa"?
earf(iris, query = "Sepal.Length", evidence = data.frame(Species = "setosa"))
```

---

expct *Expected Value*

Description

Compute the expectation of some query variable(s), optionally conditioned on some event(s).

Usage

```
expct(
  params,
  query = NULL,
  evidence = NULL,
  evidence_row_mode = c("separate", "or"),
  round = FALSE,
  nomatch = c("force", "na"),
  verbose = TRUE,
  stepsize = 0,
  parallel = TRUE
)
```

Arguments

`params`	Circuit parameters learned via `forde`.
`query`	Optional character vector of variable names. Estimates will be computed for each. If `NULL`, all variables other than those in `evidence` will be estimated. If `evidence` contains `NA`s, those values will be imputed and a full dataset is returned.
`evidence`	Optional set of conditioning events. This can take one of three forms: (1) a partial sample, i.e. a single row of data with some but not all columns; (2) a data frame of conditioning events, which allows for inequalities and intervals; or (3) a posterior distribution over leaves. See Details and Examples.
`evidence_row_mode`	Interpretation of rows in multi-row evidence. If `"separate"`, each row in `evidence` is a unique conditioning event for which `n_synth` synthetic samples are generated. If `"or"`, the rows are combined with a logical OR. See Examples.

`round`	Round continuous variables to their respective maximum precision in the real data set?
`nomatch`	What to do if no leaf matches a condition in `evidence`? Options are to force sampling from a random leaf (`"force"`) or return `NA` (`"na"`). The default is `"force"`.
`verbose`	Show warnings, e.g. when no leaf matches a condition?
`stepsize`	How many rows of evidence should be handled at each step? Defaults to `nrow(evidence) / num_registered_workers` for `parallel == TRUE`.
`parallel`	Compute in parallel? Must register backend beforehand, e.g. via `doParallel` or `doFuture`; see Examples.

Details

This function computes expected values for any subset of features, optionally conditioned on some event(s).

There are three methods for (optionally) encoding conditioning events via the `evidence` argument. The first is to provide a partial sample, where some columns from the training data are missing or set to `NA`. The second is to provide a data frame with condition events. This supports inequalities and intervals. Alternatively, users may directly input a pre-calculated posterior distribution over leaves, with columns `f_idx` and `wt`. This may be preferable for complex constraints. See Examples.

Please note that results for continuous features which are both included in `query` and in `evidence` with an interval condition are currently inconsistent.

Value

A one row data frame with values for all query variables.

References

Watson, D., Blesch, K., Kapar, J., & Wright, M. (2023). Adversarial random forests for density estimation and generative modeling. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pp. 5357-5375.

See Also

`arf`, `adversarial_rf`, `forde`, `forge`, `lik`

Examples

```
# Train ARF and estimate leaf parameters
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)

# What is the expected value of Sepal.Length?
expct(psi, query = "Sepal.Length")

# What if we condition on Species = "setosa"?
evi <- data.frame(Species = "setosa")
```

```
expct(psi, query = "Sepal.Length", evidence = evi)

# Compute expectations for all features other than Species
expct(psi, evidence = evi)

# Condition on Species = "setosa" and Petal.Width > 0.3
evi <- data.frame(Species = "setosa",
                  Petal.Width = ">0.3")
expct(psi, evidence = evi)

# Condition on first two rows with some missing values
evi <- iris[1:2,]
evi[1, 1] <- NA_real_
evi[1, 5] <- NA_character_
evi[2, 2] <- NA_real_
x_synth <- expct(psi, evidence = evi)

## Not run:
# Parallelization with doParallel
doParallel::registerDoParallel(cores = 4)

# ... or with doFuture
doFuture::registerDoFuture()
future::plan("multisession", workers = 4)

## End(Not run)
```

forde *Forests for Density Estimation*

Description

Uses a pre-trained ARF model to estimate leaf and distribution parameters.

Usage

```
forde(
  arf,
  x,
  oob = FALSE,
  family = "truncnorm",
  finite_bounds = c("no", "local", "global"),
  alpha = 0,
  epsilon = 0,
  parallel = TRUE
)
```

Arguments

arf	Pre-trained adversarial_rf. Alternatively, any object of class ranger.
x	Training data for estimating parameters.
oob	Only use out-of-bag samples for parameter estimation? If TRUE, x must be the same dataset used to train arf. Set to "inbag" to only use in-bag samples. Default is FALSE, i.e. use all observations.
family	Distribution to use for density estimation of continuous features. Current options include truncated normal (the default family = "truncnorm") and uniform (family = "unif"). See Details.
finite_bounds	Impose finite bounds on all continuous variables? If "local", infinite bounds are set to empirical extrema within leaves. If "global", infinite bounds are set to global empirical extrema. if "no" (the default), infinite bounds are left unchanged.
alpha	Optional pseudocount for Laplace smoothing of categorical features. This avoids zero-mass points when test data fall outside the support of training data. Effectively parameterizes a flat Dirichlet prior on multinomial likelihoods.
epsilon	Optional slack parameter on empirical bounds when finite_bounds != "no". This avoids zero-density points when test data fall outside the support of training data. The gap between lower and upper bounds is expanded by a factor of 1 + epsilon.
parallel	Compute in parallel? Must register backend beforehand, e.g. via doParallel or doFuture; see examples.

Details

forde extracts leaf parameters from a pretrained forest and learns distribution parameters for data within each leaf. The former includes coverage (proportion of data falling into the leaf) and split criteria. The latter includes proportions for categorical features and mean/variance for continuous features. The result is a probabilistic circuit, stored as a data.table, which can be used for various downstream inference tasks.

Currently, forde only provides support for a limited number of distributional families: truncated normal or uniform for continuous data, and multinomial for discrete data.

Though forde was designed to take an adversarial random forest as input, the function's first argument can in principle be any object of class ranger. This allows users to test performance with alternative pipelines (e.g., with supervised forest input). There is also no requirement that x be the data used to fit arf, unless oob = TRUE. In fact, using another dataset here may protect against overfitting. This connects with Wager & Athey's (2018) notion of "honest trees".

Value

A list with 5 elements: (1) parameters for continuous data; (2) parameters for discrete data; (3) leaf indices and coverage; (4) metadata on variables; and (5) the data input class. This list is used for estimating likelihoods with lik and generating data with forge.

References

Watson, D., Blesch, K., Kapar, J., & Wright, M. (2023). Adversarial random forests for density estimation and generative modeling. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pp. 5357-5375.

Wager, S. & Athey, S. (2018). Estimation and inference of heterogeneous treatment effects using random forests. *J. Am. Stat. Assoc.*, *113*(523): 1228-1242.

See Also

arf, adversarial_rf, forge, expct, lik

Examples

```
# Train ARF and estimate leaf parameters
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)

# Generate 100 synthetic samples from the iris dataset
x_synth <- forge(psi, n_synth = 100)

# Condition on Species = "setosa" and Sepal.Length > 6
evi <- data.frame(Species = "setosa",
                  Sepal.Length = "(6, Inf)")
x_synth <- forge(psi, n_synth = 100, evidence = evi)

# Estimate average log-likelihood
ll <- lik(psi, iris, arf = arf, log = TRUE)
mean(ll)

# Expectation of Sepal.Length for class setosa
evi <- data.frame(Species = "setosa")
expct(psi, query = "Sepal.Length", evidence = evi)

## Not run:
# Parallelization with doParallel
doParallel::registerDoParallel(cores = 4)

# ... or with doFuture
doFuture::registerDoFuture()
future::plan("multisession", workers = 4)

## End(Not run)
```

---

forge *Forests for Generative Modeling*

Description

Uses pre-trained FORDE model to simulate synthetic data.

Usage

```
forge(
  params,
  n_synth,
  evidence = NULL,
  evidence_row_mode = c("separate", "or"),
  round = TRUE,
  sample_NAs = FALSE,
  nomatch = c("force", "na"),
  verbose = TRUE,
  stepsize = 0,
  parallel = TRUE
)
```

Arguments

params	Circuit parameters learned via forde.
n_synth	Number of synthetic samples to generate.
evidence	Optional set of conditioning events. This can take one of three forms: (1) a partial sample, i.e. a single row of data with some but not all columns; (2) a data frame of conditioning events, which allows for inequalities; or (3) a posterior distribution over leaves. See Details.
evidence_row_mode	Interpretation of rows in multi-row evidence. If "separate", each row in evidence is a unique conditioning event for which n_synth synthetic samples are generated. If "or", the rows are combined with a logical OR. See Examples.
round	Round continuous variables to their respective maximum precision in the real data set?
sample_NAs	Sample NAs respecting the probability for missing values in the original data?
nomatch	What to do if no leaf matches a condition in evidence? Options are to force sampling from a random leaf ("force") or return NA ("na"). The default is "force".
verbose	Show warnings, e.g. when no leaf matches a condition?
stepsize	How many rows of evidence should be handled at each step? Defaults to nrow(evidence) / num_registered_workers for parallel == TRUE.
parallel	Compute in parallel? Must register backend beforehand, e.g. via doParallel or doFuture; see examples.

Details

forge simulates a synthetic dataset of n_synth samples. First, leaves are sampled in proportion to either their coverage (if evidence = NULL) or their posterior probability. Then, each feature is sampled independently within each leaf according to the probability mass or density function learned by `forde`. This will create realistic data so long as the adversarial RF used in the previous step satisfies the local independence criterion. See Watson et al. (2023).

There are three methods for (optionally) encoding conditioning events via the `evidence` argument. The first is to provide a partial sample, where some columns from the training data are missing or set to `NA`. The second is to provide a data frame with condition events. This supports inequalities and intervals. Alternatively, users may directly input a pre-calculated posterior distribution over leaves, with columns `f_idx` and `wt`. This may be preferable for complex constraints. See Examples.

Value

A dataset of `n_synth` synthetic samples.

References

Watson, D., Blesch, K., Kapar, J., & Wright, M. (2023). Adversarial random forests for density estimation and generative modeling. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pp. 5357-5375.

See Also

`arf`, `adversarial_rf`, `forde`, `expct`, `lik`

Examples

```
# Train ARF and estimate leaf parameters
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)

# Generate 100 synthetic samples from the iris dataset
x_synth <- forge(psi, n_synth = 100)

# Condition on Species = "setosa"
evi <- data.frame(Species = "setosa")
x_synth <- forge(psi, n_synth = 100, evidence = evi)

# Condition on Species = "setosa" and Sepal.Length > 6
evi <- data.frame(Species = "setosa",
                  Sepal.Length = "(6, Inf)")
x_synth <- forge(psi, n_synth = 100, evidence = evi)

# Alternative syntax for </> conditions
evi <- data.frame(Sepal.Length = ">6")
x_synth <- forge(psi, n_synth = 100, evidence = evi)

# Negation condition, i.e. all classes except "setosa"
evi <- data.frame(Species = "!setosa")
x_synth <- forge(psi, n_synth = 100, evidence = evi)

# Condition on first two data rows with some missing values
evi <- iris[1:2,]
```

```
evi[1, 1] <- NA_real_
evi[1, 5] <- NA_character_
evi[2, 2] <- NA_real_
x_synth <- forge(psi, n_synth = 1, evidence = evi)

# Or just input some distribution on leaves
# (Weights that do not sum to unity are automatically scaled)
n_leaves <- nrow(psi$forest)
evi <- data.frame(f_idx = psi$forest$f_idx, wt = rexp(n_leaves))
x_synth <- forge(psi, n_synth = 100, evidence = evi)

## Not run:
# Parallelization with doParallel
doParallel::registerDoParallel(cores = 4)

# ... or with doFuture
doFuture::registerDoFuture()
future::plan("multisession", workers = 4)

## End(Not run)
```

---

impute *Missing value imputation with ARF*

Description

Perform single or multiple imputation with ARFs. Calls `adversarial_rf`, `forde` and `expct`/`forge`.

Usage

```
impute(
  x,
  m = 1,
  expectation = ifelse(m == 1, TRUE, FALSE),
  num_trees = 100L,
  min_node_size = 10L,
  round = TRUE,
  finite_bounds = "local",
  epsilon = 1e-14,
  verbose = FALSE,
  ...
)
```

Arguments

`x`	Input data.
`m`	Number of imputed datasets to generate. The default is single imputation (`m = 1`).

expectation	Return expected value instead of multiple imputations. By default, for single imputation (m = 1), the expected value is returned.
num_trees	Number of trees to grow in the ARF.
min_node_size	Minimal number of real data samples in leaf nodes.
round	Round continuous variables to their respective maximum precision in the real data set?
finite_bounds	Impose finite bounds on all continuous variables? See forde.
epsilon	Slack parameter on empirical bounds; see forde.
verbose	Print progress for adversarial_rf?
...	Extra parameters to be passed to adversarial_rf, forde and expct/forge.

Value

Imputed data. A single dataset is returned for m = 1, a list of datasets for m > 1.

See Also

arf, forde, forge, expct, lik

Examples

```
# Generate some missings
iris_na <- iris
for (j in 1:ncol(iris)) {
  iris_na[sample(1:nrow(iris), 5), j] <- NA
}

# Single imputation
iris_imputed <- arf::impute(iris_na, num_trees = 10, m = 1)

# Multiple imputation
iris_imputed <- arf::impute(iris_na, num_trees = 10, m = 10)

## Not run:
# Parallelization with doParallel
doParallel::registerDoParallel(cores = 4)

# ... or with doFuture
doFuture::registerDoFuture()
future::plan("multisession", workers = 4)

## End(Not run)
```

lik *Likelihood Estimation*

Description

Compute the likelihood of input data, optionally conditioned on some event(s).

Usage

```
lik(
  params,
  query,
  evidence = NULL,
  arf = NULL,
  oob = FALSE,
  log = TRUE,
  batch = NULL,
  parallel = TRUE
)
```

Arguments

`params`	Circuit parameters learned via `forde`.
`query`	Data frame of samples, optionally comprising just a subset of training features. Likelihoods will be computed for each sample. Missing features will be marginalized out. See Details.
`evidence`	Optional set of conditioning events. This can take one of three forms: (1) a partial sample, i.e. a single row of data with some but not all columns; (2) a data frame of conditioning events, which allows for inequalities; or (3) a posterior distribution over leaves. See Details.
`arf`	Pre-trained `adversarial_rf` or other object of class `ranger`. This is not required but speeds up computation considerably for total evidence queries. (Ignored for partial evidence queries.)
`oob`	Only use out-of-bag leaves for likelihood estimation? If `TRUE`, `x` must be the same dataset used to train `arf`. Only applicable for total evidence queries.
`log`	Return likelihoods on log scale? Recommended to prevent underflow.
`batch`	Batch size. The default is to compute densities for all of queries in one round, which is always the fastest option if memory allows. However, with large samples or many trees, it can be more memory efficient to split the data into batches. This has no impact on results.
`parallel`	Compute in parallel? Must register backend beforehand, e.g. via `doParallel` or `doFuture`; see examples.

Details

This function computes the likelihood of input data, optionally conditioned on some event(s). Queries may be partial, i.e. covering some but not all features, in which case excluded variables will be marginalized out.

There are three methods for (optionally) encoding conditioning events via the `evidence` argument. The first is to provide a partial sample, where some but not all columns from the training data are present. The second is to provide a data frame with three columns: `variable`, `relation`, and `value`. This supports inequalities via `relation`. Alternatively, users may directly input a pre-calculated posterior distribution over leaves, with columns `f_idx` and `wt`. This may be preferable for complex constraints. See Examples.

Value

A vector of likelihoods, optionally on the log scale.

References

Watson, D., Blesch, K., Kapar, J., & Wright, M. (2023). Adversarial random forests for density estimation and generative modeling. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pp. 5357-5375.

See Also

`arf`, `adversarial_rf`, `forde`, `forge`, `expct`

Examples

```
# Train ARF and estimate leaf parameters
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)

# Estimate average log-likelihood
ll <- lik(psi, iris, arf = arf, log = TRUE)
mean(ll)

# Identical but slower
ll <- lik(psi, iris, log = TRUE)
mean(ll)

# Partial evidence query
lik(psi, query = iris[1, 1:3])

# Condition on Species = "setosa"
evi <- data.frame(Species = "setosa")
lik(psi, query = iris[1, 1:3], evidence = evi)

# Condition on Species = "setosa" and Petal.Width > 0.3
evi <- data.frame(Species = "setosa",
                  Petal.Width = ">0.3")
lik(psi, query = iris[1, 1:3], evidence = evi)
```

```
## Not run:
# Parallelization with doParallel
doParallel::registerDoParallel(cores = 4)

# ... or with doFuture
doFuture::registerDoFuture()
future::plan("multisession", workers = 4)

## End(Not run)
```

---

rarf *Shortcut sampling function*

Description

Calls `adversarial_rf`, `forde` and `forge`. For repeated application, it is faster to save outputs of `adversarial_rf` and `forde` and pass them via `...` or directly use `forge`.

Usage

```
rarf(x, n_synth = NULL, ...)
```

Arguments

`x`	Input data. Integer variables are recoded as ordered factors with a warning. See Details.
`n_synth`	Number of synthetic samples to generate for unconditional generation with no `evidence` given. Number of synthetic samples to generate per `evidence` row if `evidence` is provided. If `NULL`, defaults to `nrow(x)` if no `evidence` is provided and to 1 otherwise.
`...`	Extra parameters to be passed to `adversarial_rf`, `forde` and `forge`.

Value

A dataset of `n_synth` synthetic samples or of `nrow(x)` synthetic samples if `n_synth` is undefined.

References

Watson, D., Blesch, K., Kapar, J., & Wright, M. (2023). Adversarial random forests for density estimation and generative modeling. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pp. 5357-5375.

See Also

`arf`, `adversarial_rf`, `forde`, `forge`

Examples

```
# Generate 150 (size of original iris dataset) synthetic samples from the iris dataset
x_synth <- rarf(iris)

# Generate 100 synthetic samples from the iris dataset
x_synth <- rarf(iris, n_synth = 100)

# Condition on Species = "setosa"
x_synth <- rarf(iris, evidence = data.frame(Species = "setosa"))
```

Index

adversarial_rf, *3*, 3, *6*, *7*, *9*, *11*, *12*, *14*, *17–19*
arf, *5–7*, *9*, *12*, *14*, *16*, *18*, *19*
arf (arf-package), 2
arf-package, 2

darf, 6

earf, 7
expct, *3*, *5*, *7*, 8, *12*, *14*, *16*, *18*

forde, *3–9*, 10, *13*, *14*, *16–19*
forge, *3–6*, *9*, *11*, *12*, 12, *16*, *18*, *19*

impute, 15

lik, *3*, *5*, *9*, *11*, *12*, *14*, *16*, 17

rarf, 19