

# Package: arf (via r-universe)

October 4, 2026

**Title** Adversarial Random Forests

**Version** 0.2.5.9000

**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, foreach, ranger, stringr, truncnorm

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

**VignetteBuilder** knitr

**Config/roxygen2/version** 8.1.0

**Config/testthat/edition** 3

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**Config/pak/sysreqs** libicu-dev

**Repository** <https://bips-hb.r-universe.dev>

**Date/Publication** 2026-10-04 17:53:27 UTC

**RemoteUrl** `https://github.com/bips-hb/arf`  
**RemoteRef** `HEAD`  
**RemoteSha** `04105443073febc51e5c339e53de775fa911c6c7`

**Contents**

|                              |           |
|------------------------------|-----------|
| arf-package . . . . .        | 2         |
| adversarial_rf . . . . .     | 4         |
| arf-options . . . . .        | 6         |
| darf . . . . .               | 8         |
| earf . . . . .               | 9         |
| expct . . . . .              | 10        |
| forde . . . . .              | 12        |
| forge . . . . .              | 15        |
| impute . . . . .             | 17        |
| lik . . . . .                | 19        |
| rarf . . . . .               | 21        |
| sample_from_leaves . . . . . | 22        |
| <b>Index</b>                 | <b>24</b> |

---

|             |                                        |
|-------------|----------------------------------------|
| arf-package | <i>arf: Adversarial Random Forests</i> |
|-------------|----------------------------------------|

---

**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:
- Marvin N. Wright <cran@wrig.de> ([ORCID](#))
  - David S. Watson <david.s.watson11@gmail.com> ([ORCID](#))
  - Kristin Blesch ([ORCID](#))
  - Jan Kapar ([ORCID](#))

Other contributors:

- Lukas Burk <cran@lukasburk.de> ([ORCID](#)) [contributor]
- Martin Jullum <Martin.Jullum@nr.no> ([ORCID](#)) [contributor]

## 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)

# ... or with mirai (shares large read-only inputs across workers via mori)
mirai::daemons(4)

## End(Not run)
```

## Description

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

## Usage

```
adversarial_rf(
  x,
  num_trees = 10L,
  min_node_size = 2L,
  mtry = NULL,
  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.                                                                                                                                                                                                                                                                  |
| mtry          | Number of candidate features at each split. Default is $\max(2, \text{floor}(\sqrt{p}))$ for $p$ features, capped at $p$ . This differs from <code>ranger</code> 's $\text{floor}(\sqrt{p})$ , which yields 1 for $p < 4$ and weakens the discriminator.                                                            |
| delta         | Tolerance parameter. Algorithm converges when OOB accuracy is $< 0.5 + \text{delta}$ .                                                                                                                                                                                                                              |
| 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 <code>min_node_size</code> by pruning?                                                                                                                                                                                                                                                                       |
| verbose       | Print discriminator accuracy after each round? Will also show additional warnings.                                                                                                                                                                                                                                  |
| parallel      | Compute in parallel? Enables multithreaded <code>ranger</code> training (no backend needed) and parallelizes the pruning step, which requires a registered <code>foreach</code> backend ( <code>doParallel</code> , <code>doFuture</code> ) or active <code>mirai</code> daemons. See <a href="#">arf-options</a> . |
| ...           | Extra parameters to be passed to <code>ranger</code> .                                                                                                                                                                                                                                                              |

## 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)

# ... or with mirai (shares large read-only inputs across workers via mori)
mirai::daemons(4)

## End(Not run)

```

---

arf-options

*arf package options*


---

## Description

Options controlling the parallel backend and its messaging, set via [options](#).

## Details

**arf.backend** Parallel backend used when `parallel = TRUE`: "foreach" or "mirai". If unset, arf uses "mirai" when mirai daemons are running and "foreach" otherwise.

**arf.verbose** Report the selected backend once per backend configuration per session? Default TRUE; set FALSE to silence.

**arf.block\_rows** Cap on rows materialized per block of conditions in [expct](#). Default 5e6. Lower it to trade speed for memory on large forests with many conditions.

**arf.block\_rows** does not affect results, only peak memory and speed. For memory-constrained hardware, combine a small daemon count with a lower `arf.block_rows` and the batch/stepsize arguments of [lik](#), [forge](#) and [expct](#).

The "foreach" backend uses whatever adapter is registered (e.g. `doParallel`, `doFuture`). The "mirai" backend uses mirai daemons and shares large read-only inputs (training data, forest, learned parameters) across workers via `mori`, so workers do not each copy them. In our benchmarks ( $n = 20000$ , 200 trees, up to 16 workers) run time was similar between the backends, though on a cold daemon pool "mirai" can be noticeably slower. The memory benefit is largest for the

tree-parallel operations (`forde`, `adversarial_rf`) on large forests with many workers: at 16 workers peak memory was about 30 percent lower for `forde` and about half for `adversarial_rf`. Expect smaller gains on smaller problems, where the daemon pool's fixed overhead can outweigh the sharing, so prefer "mirai" at scale and either backend otherwise. Daemons started via `future.mirai` (e.g. `plan(future.mirai::mirai_multisession)`) are detected like any other mirai daemons, so `futureverse` users get the "mirai" backend automatically. None of these packages are installed with `arf`: install `mirai` and `mori` for the "mirai" backend, `doParallel` or `doFuture` for "foreach", and `doRNG` or `future.mirai` if you use them below.

Workers and `data.table` threads multiply. If their product exceeds the core count, cap threads per worker with `data.table::setDTthreads()` and set `OMP_WAIT_POLICY=passive` before starting R so idle OpenMP threads sleep instead of spinning.

Reproducibility of stochastic operations (`forge`, categorical `expct`) under parallel execution: `set.seed` only governs the calling process, not the workers. With the "mirai" backend, seed the daemons instead: `mirai::daemons(n, seed = 42)` gives reproducible results provided the daemon count, the seed and the sequence of calls on a fresh daemon pool are kept fixed (changing the daemon count changes how work is chunked and therefore the random stream assignment). This also applies to pools started via `future.mirai`: `future`'s own seed machinery covers only work dispatched through its API, which `arf`'s backend bypasses, and `plan()` does not accept a seed argument, so seed the pool with `mirai::daemons()` directly. For the "foreach" backend, register `doRNG` on top of the adapter (`doRNG::registerDoRNG(42)` after `registerDoParallel()` or `registerDoFuture()`): results are then reproducible, identical across adapters, and independent of the worker count, provided stepsize is set explicitly (its default depends on the worker count). Without `doRNG`, `doFuture` flags the stochastic operations with "UNRELIABLE VALUE" warnings. Sequential execution (`parallel = FALSE`) with `set.seed` is exact as always.

## Examples

```
## Not run:
arf <- adversarial_rf(iris)

# foreach backend
doParallel::registerDoParallel(cores = 4)
psi <- forde(arf, iris)

# mirai backend: start daemons, then call as usual
mirai::daemons(4)
psi <- forde(arf, iris)
mirai::daemons(0) # shut down when done

# futureverse: future.mirai daemons are detected automatically
future::plan(future.mirai::mirai_multisession, workers = 4)
psi <- forde(arf, iris)
future::plan("sequential") # shuts the daemons down

# force a backend regardless of what is registered (NULL restores auto-detection)
options(arf.backend = "mirai")
options(arf.backend = NULL)

# silence the backend message
options(arf.verbose = FALSE)
```

```

# reproducible parallel sampling with mirai: seeded daemons
# (fixed daemon count, fresh pool)
evi <- data.frame(Species = sample(levels(iris$Species), 100, replace = TRUE))
mirai::daemons(4, seed = 42)
# stepsize = evidence rows per step; 25 = 100 conditions / 4 workers,
# i.e. the default sizing, made explicit
x <- forge(psi, n_synth = 1, evidence = evi, stepsize = 25)
mirai::daemons(0)

# reproducible parallel sampling with foreach: doRNG on top of the
# adapter; set stepsize explicitly (its default depends on worker count)
doParallel::registerDoParallel(cores = 4)
doRNG::registerDoRNG(42)
x <- forge(psi, n_synth = 1, evidence = evi, stepsize = 25)

## 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

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

## 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**

|                  |                                                                                                            |
|------------------|------------------------------------------------------------------------------------------------------------|
| <code>x</code>   | Input data. Integer variables are recoded as ordered factors with a warning. See Details.                  |
| <code>...</code> | Extra parameters to be passed to <code>adversarial_rf</code> , <code>forde</code> and <code>expct</code> . |

**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 | <i>Expected Value</i> |
|-------|-----------------------|

---

## 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 <a href="#">forde</a> .                                                                                                                                                                                                                                                             |
| 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 NAs, 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 <code>n_synth</code> 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? Smaller steps lower peak memory. Defaults to <code>nrow(evidence)</code> divided by the number of workers or daemons when <code>parallel = TRUE</code> , so seeded parallel runs only reproduce across worker counts if <code>stepsize</code> is set explicitly. See <a href="#">arf-options</a> .                                                                   |
| parallel | Compute in parallel? Requires a registered foreach backend ( <code>doParallel</code> , <code>doFuture</code> ) or active mirai daemons. See <a href="#">arf-options</a> . With <code>evidence_row_mode = "or"</code> , parallelization happens inside the conditional circuit computation; in benchmarks this gave little speedup while raising peak memory, so consider <code>parallel = FALSE</code> for large "or" queries. |

## 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)

# ... or with mirai (shares large read-only inputs across workers via mori)
mirai::daemons(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

|                            |                                                                                                                                                                                                                                                                                                 |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>arf</code>           | Pre-trained <a href="#">adversarial_rf</a> . Alternatively, any object of class <code>ranger</code> .                                                                                                                                                                                           |
| <code>x</code>             | Training data for estimating parameters.                                                                                                                                                                                                                                                        |
| <code>oob</code>           | Only use out-of-bag samples for parameter estimation? If <code>TRUE</code> , <code>x</code> must be the same dataset used to train <code>arf</code> . Set to <code>"inbag"</code> to only use in-bag samples. Default is <code>FALSE</code> , i.e. use all observations.                        |
| <code>family</code>        | Distribution to use for density estimation of continuous features. Current options include truncated normal (the default <code>family = "truncnorm"</code> ) and uniform ( <code>family = "unif"</code> ). See Details.                                                                         |
| <code>finite_bounds</code> | Impose finite bounds on all continuous variables? If <code>"local"</code> , infinite bounds are set to empirical extrema within leaves. If <code>"global"</code> , infinite bounds are set to global empirical extrema. If <code>"no"</code> (the default), infinite bounds are left unchanged. |
| <code>alpha</code>         | 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.                                                         |
| <code>epsilon</code>       | Optional slack parameter on empirical bounds when <code>finite_bounds != "no"</code> . 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 $1 + \epsilon$ .                             |
| <code>parallel</code>      | Compute in parallel? Requires a registered foreach backend ( <code>doParallel</code> , <code>doFuture</code> ) or active <code>mirai</code> daemons. See <a href="#">arf-options</a> .                                                                                                          |

## 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)

# ... or with mirai (shares large read-only inputs across workers via mori)
mirai::daemons(4)

## End(Not run)
```

## 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 <a href="#">forde</a> .                                                                                                                                                                                                                                  |
| 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? Smaller steps lower peak memory. Defaults to <code>nrow(evidence)</code> divided by the number of workers or daemons when <code>parallel = TRUE</code> , so seeded parallel runs only reproduce across worker counts if <code>stepsize</code> is set explicitly. See <a href="#">arf-options</a> .                                                                                |
| parallel | Compute in parallel? Requires a registered foreach backend ( <code>doParallel</code> , <code>doFuture</code> ) or active <code>mirai</code> daemons. See <a href="#">arf-options</a> . With <code>evidence_row_mode = "or"</code> , parallelization happens inside the conditional circuit computation; in benchmarks this gave little speedup while raising peak memory, so consider <code>parallel = FALSE</code> for large "or" queries. |

## 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)

# ... or with mirai (shares large read-only inputs across workers via mori)
mirai::daemons(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 = m == 1,
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 <a href="#">forde</a> .                                                     |
| epsilon       | Slack parameter on empirical bounds; see <a href="#">forde</a> .                                                                  |
| 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)

# ... or with mirai (shares large read-only inputs across workers via mori)
mirai::daemons(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 <a href="#">forde</a> .                                                                                                                                                                                                                                  |
| 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 <a href="#">adversarial_rf</a> 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 computes densities for all queries in one round. Splitting into batches lowers peak memory with large samples or many trees, and is what parallel distributes across workers. This has no impact on results.              |
| parallel | Compute in parallel? Batches are processed in parallel, so this only takes effect when batch is smaller than nrow(query). Requires a registered foreach backend (doParallel, doFuture) or active mirai daemons. See <a href="#">arf-options</a> . |

## 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)

# ... or with mirai (shares large read-only inputs across workers via mori)
mirai::daemons(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

|                      |                                                                                                                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>       | Input data. Integer variables are recoded as ordered factors with a warning. See Details.                                                                                                                                                                                             |
| <code>n_synth</code> | 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 <code>NULL</code> , defaults to <code>nrow(x)</code> 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"))
```

---

|                    |                                                                               |
|--------------------|-------------------------------------------------------------------------------|
| sample_from_leaves | <i>Generate synthetic data by sampling from the leaves of a random forest</i> |
|--------------------|-------------------------------------------------------------------------------|

---

### Description

Draws synthetic samples by sampling, for each observation, a leaf from the forest and then drawing each feature independently (marginally) from the real observations that fall into that leaf. This is the intra-leaf sampling step used internally by [adversarial\\_rf](#) to generate synthetic data during the adversarial loop, exposed here as a stand-alone function.

### Usage

```
sample_from_leaves(
  arf,
  x_real,
  params = NULL,
  round = TRUE,
  factor_cols = NULL,
  lvls = NULL,
  prep = TRUE
)
```

**Arguments**

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| arf         | A trained ARF, as returned by <a href="#">adversarial_rf</a> (a ranger object).                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| x_real      | Data whose intra-leaf structure is used for sampling, typically the data the forest was trained on.                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| params      | Optional circuit parameters as returned by <a href="#">forde</a> . If supplied, the synthetic data is post-processed with the same routine used by <a href="#">forge</a> : variable types and factor levels are restored, continuous variables are rounded to their observed precision (see <a href="#">round</a> ), and the class of the original input is reinstated. If NULL, a minimally processed data.table is returned with factor columns encoded as character, matching the representation used internally by <a href="#">adversarial_rf</a> . |
| round       | Round continuous variables to their maximum precision in the real data? Only relevant when params is supplied.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| factor_cols | Optional logical vector flagging the factor columns of x_real. Computed from x_real if NULL. Mainly for internal use to avoid recomputation.                                                                                                                                                                                                                                                                                                                                                                                                            |
| lvls        | Optional list of factor levels for the factor columns of x_real. Computed from x_real if NULL. Mainly for internal use.                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| prep        | Prepare x_real with the internal pre-processing routine before sampling? Set to FALSE if x_real is already prepared (internal use).                                                                                                                                                                                                                                                                                                                                                                                                                     |

**Value**

A dataset of `nrow(x_real)` synthetic samples. When params is supplied, its class and column types match the original data; otherwise a data.table with factor columns encoded as character.

**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**

[adversarial\\_rf](#), [forde](#), [forge](#)

**Examples**

```
arf <- adversarial_rf(iris)

# Minimally processed output (factors as character)
x_synth <- sample_from_leaves(arf, iris)

# Fully post-processed output, consistent with forge()
psi <- forde(arf, iris)
x_synth <- sample_from_leaves(arf, iris, params = psi)
```

# Index

adversarial\_rf, [3](#), [4](#), [7](#), [9](#), [11](#), [13](#), [14](#), [16](#), [20](#),  
[22](#), [23](#)  
arf, [5](#), [9](#), [11](#), [14](#), [16](#), [18](#), [20](#), [22](#)  
arf (arf-package), [2](#)  
arf-options, [6](#)  
arf-package, [2](#)  
arf.backend (arf-options), [6](#)  
arf.block\_rows (arf-options), [6](#)  
arf.verbose (arf-options), [6](#)  
  
darf, [8](#)  
  
earf, [9](#)  
expct, [3](#), [5–7](#), [9](#), [10](#), [14](#), [16](#), [18](#), [20](#)  
  
forde, [3](#), [5](#), [7](#), [9–11](#), [12](#), [15](#), [16](#), [18–20](#), [22](#), [23](#)  
forge, [3](#), [5–7](#), [9](#), [11](#), [14](#), [15](#), [18](#), [20](#), [22](#), [23](#)  
  
impute, [17](#)  
  
lik, [3](#), [5](#), [6](#), [11](#), [14](#), [16](#), [18](#), [19](#)  
  
options, [6](#)  
  
rarf, [21](#)  
  
sample\_from\_leaves, [22](#)  
set.seed, [7](#)