Package ‘arf’
January 24, 2024

Title Adversarial Random Forests
Version 0.2.0
Date 2024-01-24
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. (2022) <arXiv:2205.09435>.
License GPL (>= 3)
BugReports https://github.com/bips-hb/arf/issues
Imports data.table, ranger, foreach, truncnorm
Encoding UTF-8
RoxygenNote 7.3.0
Suggests ggplot2, doParallel, mlbench, knitr, rmarkdown, tibble, 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 2024-01-24 14:53:15 UTC
adversarial_rf

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.
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. Default behavior is to convert those 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


See Also

forde, forge
Examples

arf <- adversarial_rf(iris)

---

**colrename**

*Adaptive column renaming*

**Description**

This function renames columns in case the input data.frame includes any colnames required by internal functions (e.g., "y").

**Usage**

```
col_rename(df, old_name)
```

**Arguments**

- `df`: Input data.frame.
- `old_name`: Name of column to be renamed.

---

**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)
```

**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.
- `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.
forde

Details

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

Value

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

References


See Also

adversarial_rf, forde, lik

Examples

# Train ARF and corresponding circuit
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)

# What is the expected value 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)

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 = FALSE,
  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.

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?

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 parametrizes a flat Dirichlet prior on multinomial likelihoods.

epsilon
Optional slack parameter on empirical bounds when family = "unif" or finite_bounds = TRUE. 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.

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. Future releases will accommodate a larger set of options.

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 \texttt{lik} and generating data with \texttt{forge}.

References


See Also

\texttt{adversarial\_rf}, \texttt{forge}, \texttt{lik}

Examples

arf <- adversarial\_rf(iris)
psi <- forde(arf, iris)
head(psi)
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 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 dataset of n_synth synthetic samples.

References

See Also
adversarial_rf, forde

Examples
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)
x_synth <- forge(psi, n_synth = 100)

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

# Condition in Species = "setosa" and Sepal.Length > 6
ev <- data.frame(variable = c("Species", "Sepal.Length"),
  relation = c("==", ">"),
  value = c("setosa", 6))
x_synth <- forge(psi, n_synth = 100, evidence = ev)

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

Compute leaf posterior

Description

This function returns a posterior distribution on leaves, conditional on some evidence.

Usage

leaf_posterior(params, evidence)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>params</td>
<td>Circuit parameters learned via forde.</td>
</tr>
<tr>
<td>evidence</td>
<td>Data frame of conditioning event(s).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>params</td>
<td>Circuit parameters learned via forde.</td>
</tr>
<tr>
<td>query</td>
<td>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.</td>
</tr>
<tr>
<td>evidence</td>
<td>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.</td>
</tr>
</tbody>
</table>
### lik

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arf</code></td>
<td>Pre-trained <code>adversarial_rf</code> or other object of class ranger. This is not required but speeds up computation considerably for total evidence queries. (Ignored for partial evidence queries.)</td>
</tr>
<tr>
<td><code>oob</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>log</code></td>
<td>Return likelihoods on log scale? Recommended to prevent underflow.</td>
</tr>
<tr>
<td><code>batch</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>parallel</code></td>
<td>Compute in parallel? Must register backend beforehand, e.g. via <code>doParallel</code>.</td>
</tr>
</tbody>
</table>

**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**


**See Also**

`adversarial_rf`, `forge`

**Examples**

```r
# Estimate average log-likelihood
arf <- adversarial_rf(iris)
psi <- forde(arf, iris)
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"
ev <- data.frame(Species = "setosa")
lik(psi, query = iris[1, 1:3], evidence = ev)

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

---

**post_x**

*Post-process data*

**Description**

This function prepares output data for forge.

**Usage**

post_x(x, params)

**Arguments**

- **x**
  - Input data.frame.

- **params**
  - Circuit parameters learned via forde.

---

**prep_evi**

*Preprocess evidence*

**Description**

This function prepares the evidence for computing leaf posteriors.

**Usage**

prep_evi(params, evidence)

**Arguments**

- **params**
  - Circuit parameters learned via forde.

- **evidence**
  - Optional set of conditioning events.
**Description**

This function prepares input data for ARFs.

**Usage**

```r
prep_x(x)
```

**Arguments**

- `x` Input data.frame.
Index

adversarial_rf, 2, 5–8, 10

col_rename, 4

expct, 4

forde, 3–5, 5, 7–9, 11
forge, 3, 7, 7, 10

leaf_posterior, 9
lik, 5, 7, 9

post_x, 11
prep_evi, 11
prep_x, 12