Package 'SoftBart'

May 11, 2025

Type Package

Title Implements the SoftBart Algorithm

Version 1.0.2

Date 2025-05-10

Encoding UTF-8

Description Implements the SoftBart model of described by Linero and Yang (2018) <doi:10.1111/rssb.12293>, with the optional use of a sparsity-inducing prior to allow for variable selection. For usability, the package maintains the same style as the 'BayesTree' package.

License GPL (>= 2)

Imports Rcpp (>= 0.12.9), glmnet (>= 4.0.0), scales (>= 1.1.1), methods, caret, truncnorm, progress, MASS

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.1

NeedsCompilation yes

Author Antonio R. Linero [aut, cre]

Maintainer Antonio R. Linero <antonio.linero@austin.utexas.edu>

Repository CRAN

Date/Publication 2025-05-10 23:10:02 UTC

Contents

`contr.ltfr` *Create a Full Set of Dummy Variables*

Description

Used with `dummyVars` in the **caret** package to create a full set of dummy variables (i.e. less than full rank parameterization).

Usage

```
contr.ltfr(...)
```

Arguments

...	A list of arguments.

Value

A matrix produced containing full sets of dummy variables.

`gsoftbart_regression` *General SoftBart Regression*

Description

Fits the general (Soft) BART (GBART) model, which combines the BART model with a linear predictor. That is, it fits the semiparametric Gaussian regression model

$$Y = r(X) + Z^\top \beta + \epsilon$$

where the function $r(x)$ is modeled using a BART ensemble.

Usage

```
gsoftbart_regression(
  formula,
  linear_formula,
  data,
  test_data,
  num_tree = 20,
  k = 2,
  hypers = NULL,
  opts = NULL,
  remove_intercept = TRUE,
  verbose = TRUE,
  warn = TRUE
)
```

Arguments

formula	A model formula with a numeric variable on the left-hand-side and non-linear predictors on the right-hand-side.
linear_formula	A model formula with the linear variables on the right-hand-side (left-hand-side is not used).
data	A data frame consisting of the training data.
test_data	A data frame consisting of the testing data.
num_tree	The number of trees used in the ensemble.
k	Determines the standard deviation of the leaf node parameters, which is given by 3 / k / sqrt(num_tree).
hypers	A list of hyperparameters constructed from the Hypers() function (num_tree, k, and sigma_mu are overridden by this function).
opts	A list of options for running the chain constructed from the Opts() function (update_sigma is overridden by this function).
remove_intercept	If TRUE then any intercept term in the linear formula will be removed, with the overall location of the outcome captured by the nonparametric function.
verbose	If TRUE, progress of the chain will be printed to the console.
warn	If TRUE, remind the user that they probably don't want the linear predictors to be included in the formula for the nonlinear part.

Value

Returns a list with the following components

- r_train: samples of the nonparametric function evaluated on the training set.
- r_test: samples of the nonparametric function evaluated on the test set.
- eta_train: samples of the linear predictor on the training set.
- eta_test: samples of the linear predictor on the test set.

- `mu_train`: samples of the prediction on the training set.
- `mu_test`: samples of the prediction on the test set.
- `beta`: samples of the regression coefficients.
- `sigma`: samples of the error standard deviation.
- `sigma_mu`: samples of the standard deviation of the leaf node parameters.
- `var_counts`: a matrix with a column for each nonparametric predictor containing the number of times that predictor is used in the ensemble at each iteration.
- `opts`: the options used when running the chain.
- `formula`: the formula specified by the user.
- `ecdfs`: empirical distribution functions, used by the predict function.
- `mu_Y, sd_Y`: used with the predict function to transform predictions.
- `forest`: a forest object for the nonlinear part; see the `MakeForest()` documentation for more details.

Examples

```
## NOTE: SET NUMBER OF BURN IN AND SAMPLE ITERATIONS HIGHER IN PRACTICE

num_burn <- 10 ## Should be ~ 5000
num_save <- 10 ## Should be ~ 5000

set.seed(1234)
f_fried <- function(x) 10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]

gen_data <- function(n_train, n_test, P, sigma) {
  X <- matrix(runif(n_train * P), nrow = n_train)
  mu <- f_fried(X)
  X_test <- matrix(runif(n_test * P), nrow = n_test)
  mu_test <- f_fried(X_test)
  Y <- mu + sigma * rnorm(n_train)
  Y_test <- mu + sigma * rnorm(n_test)

  return(list(X = X, Y = Y, mu = mu, X_test = X_test, Y_test = Y_test,
              mu_test = mu_test))
}

## Simiulate dataset
sim_data <- gen_data(250, 250, 100, 1)

df <- data.frame(X = sim_data$X, Y = sim_data$Y)
df_test <- data.frame(X = sim_data$X_test, Y = sim_data$Y_test)

## Fit the model

opts <- Opts(num_burn = num_burn, num_save = num_save)
fitted_reg <- gsoftbart_regression(Y ~ . - X.4 - X.5, ~ X.4 + X.5, df, df_test, opts = opts)
```

```
## Plot results

plot(colMeans(fitted_reg$mu_test), sim_data$mu_test)
abline(a = 0, b = 1)
plot(fitted_reg$beta[,1])
plot(fitted_reg$beta[,2])
```

Hypers *Create a list of hyperparameter values*

Description

Creates a list which holds all the hyperparameters for use with the model-fitting functions and with the `MakeForest` functionality.

Usage

```
Hypers(
  X,
  Y,
  group = NULL,
  alpha = 1,
  beta = 2,
  gamma = 0.95,
  k = 2,
  sigma_hat = NULL,
  shape = 1,
  width = 0.1,
  num_tree = 20,
  alpha_scale = NULL,
  alpha_shape_1 = 0.5,
  alpha_shape_2 = 1,
  tau_rate = 10,
  num_tree_prob = NULL,
  temperature = 1,
  weights = NULL,
  normalize_Y = TRUE
)
```

Arguments

`X`	A matrix of training data covariates.
`Y`	A vector of training data responses.
`group`	Allows for grouping of covariates with shared splitting proportions, which is useful for categorical dummy variables. For each column of `X`, `group` gives the associated group.
`alpha`	Positive constant controlling the sparsity level.

beta	Parameter penalizing tree depth in the branching process prior.
gamma	Parameter penalizing new nodes in the branching process prior.
k	Related to the signal-to-noise ratio, `sigma_mu = 0.5 / (sqrt(num_tree) * k)`. BART defaults to `k = 2` after applying the max/min normalization to the outcome.
sigma_hat	A prior guess at the conditional variance of `Y` given `X`. If not provided, this is estimated empirically by linear regression.
shape	Shape parameter for gating probabilities.
width	Bandwidth of gating probabilities.
num_tree	Number of trees in the ensemble.
alpha_scale	Scale of the prior for `alpha`; if not provided, defaults to the number of predictors.
alpha_shape_1	Shape parameter for prior on `alpha`; if not provided, defaults to 0.5.
alpha_shape_2	Shape parameter for prior on `alpha`; if not provided, defaults to 1.0.
tau_rate	Rate parameter for the bandwidths of the trees with an exponential prior; defaults to 10.
num_tree_prob	Parameter for geometric prior on number of tree.
temperature	The temperature applied to the posterior distribution; set to 1 unless you know what you are doing.
weights	Only used by the function `softbart`, this is a vector of weights to be used in heteroskedastic regression models, with the variance of an observation given by `sigma_sq / weight`.
normalize_Y	Do you want to compute `sigma_hat` after applying the standard BART max/min normalization to $(-0.5, 0.5)$ for the outcome? If `FALSE`, no normalization is applied. This might be useful for fitting custom models where the outcome is normalized by hand.

Value

Returns a list containing the function arguments.

`MakeForest` *Create an Rcpp_Forest Object*

Description

Make an object of type `Rcpp_Forest`, which can be used to embed a soft BART model into other models. Some examples are given in the package vignette.

Usage

```
MakeForest(hypers, opts, warn = TRUE)
```

Arguments

hypers	A list of hyperparameter values obtained from Hypers() function
opts	A list of MCMC chain settings obtained from Opts() function
warn	If TRUE, reminds the user to normalize their design matrix when interacting with a forest object.

Value

Returns an object of type Rcpp_Forest. If forest is an Rcpp_Forest object then it has the following methods.

- forest$do_gibbs(X, Y, X_test, i) runs i iterations of the Bayesian backfitting algorithm and predicts on the test set X_test. The state of forest is also updated.
- forest$do_gibbs_weighted(X, Y, weights X_test, i) runs i iterations of the Bayesian backfitting algorithm and predicts on the test set X_test; assumes that Y is heteroskedastic with known weights. The state of forest is also updated.
- forest$do_predict(X) returns the predictions from a matrix X of predictors.
- forest$get_counts() returns the number of times each variable has been used in a splitting rule at the current state of forest.
- forest$get_s() returns the splitting probabilities of the forest.
- forest$get_sigma() returns the error standard deviation of the forest.
- forest$get_sigma_mu() returns the standard deviation of the leaf node parameters.
- forest$get_tree_counts() returns a matrix with a row for each group of predictors and a column for each tree that counts the number of times each group of predictors is used in each tree at the current state of forest.
- forest$predict_iteration(X, i) returns the predictions from a matrix X of predictors at iteration i. Requires that opts$cache_trees = TRUE in MakeForest(hypers, opts).
- forest$set_s(s) sets the splitting probabilities of the forest to s.
- forest$set_sigma(x) sets the error standard deviation of the forest to x.
- forest$num_gibbs returns the number of iterations in total that the Gibbs sampler has been run.

Examples

```
X <- matrix(runif(100 * 10), nrow = 100, ncol = 10)
Y <- rowSums(X) + rnorm(100)
my_forest <- MakeForest(Hypers(X,Y), Opts())
mu_hat <- my_forest$do_gibbs(X,Y,X,200)
```

`Opts` *MCMC options for SoftBart*

Description

Creates a list that provides the parameters for running the Markov chain.

Usage

```
Opts(
  num_burn = 2500,
  num_thin = 1,
  num_save = 2500,
  num_print = 100,
  update_sigma_mu = TRUE,
  update_s = TRUE,
  update_alpha = TRUE,
  update_beta = FALSE,
  update_gamma = FALSE,
  update_tau = TRUE,
  update_tau_mean = FALSE,
  update_sigma = TRUE,
  cache_trees = TRUE
)
```

Arguments

`num_burn`	Number of warmup iterations for the chain.
`num_thin`	Thinning interval for the chain.
`num_save`	The number of samples to collect; in total, `num_burn + num_save * num_thin` iterations are run.
`num_print`	Interval for how often to print the chain's progress.
`update_sigma_mu`	If `TRUE`, `sigma_mu` is updated, with a half-Cauchy prior on `sigma_mu` centered at the initial guess.
`update_s`	If `TRUE`, `s` is updated using the Dirichlet prior $s \sim D(\alpha/P, \ldots, \alpha/P)$ where P is the number of covariates.
`update_alpha`	If `TRUE`, `alpha` is updated using a scaled beta prime prior.
`update_beta`	If `TRUE`, `beta` is updated using a normal prior with mean 0 and variance 4.
`update_gamma`	If `TRUE`, gamma is updated using a Uniform(0.5, 1) prior.
`update_tau`	If `TRUE`, the bandwidth `tau` is updated for each tree
`update_tau_mean`	If `TRUE`, the mean of `tau` is updated

update_sigma	If TRUE, sigma is updated, with a half-Cauchy prior on sigma centered at the initial guess.
cache_trees	If TRUE, we save the trees for each MCMC iteration when using the MakeForest interface

Value

Returns a list containing the function arguments.

`partial_dependence_probit` *Partial Dependence Function for SoftBART Probit Regression*

Description

Computes the partial dependence function for a given covariate at a given set of covariate values for the probit model.

Usage

```
partial_dependence_probit(fit, test_data, var_str, grid)
```

Arguments

fit	A fitted model of type softbart_probit.
test_data	A data set used to form the baseline distribution of covariates for the partial dependence function.
var_str	A string giving the variable name of the predictor to compute the partial dependence function for.
grid	The values of the predictor to compute the partial dependence function at.

Value

Returns a list with the following components:

- pred_df: a data frame containing columns for a MCMC iteration ID (sample), the value on the grid, and the partial dependence function value.
- p: a matrix containing the same information as pred_df, with the rows corresponding to iterations and columns corresponding to grid values.
- grid: the grid used as input.

partial_dependence_regression *Partial Dependence Function for SoftBART Regression*

Description

Computes the partial dependence function for a given covariate at a given set of covariate values.

Usage

```
partial_dependence_regression(fit, test_data, var_str, grid)
```

Arguments

fit	A fitted model of type softbart_regression.
test_data	A data set used to form the baseline distribution of covariates for the partial dependence function.
var_str	A string giving the variable name of the predictor to compute the partial dependence function for.
grid	The values of the predictor to compute the partial dependence function at.

Value

Returns a list with the following components:

- pred_df: a data.frame containing columns for a MCMC iteration ID (sample), the value on the grid, and the partial dependence function value.
- mu: a matrix containing the same information as pred_df, with the rows corresponding to iterations and columns corresponding to grid values.
- grid: the grid used as input.

Examples

```
## NOTE: SET NUMBER OF BURN IN AND SAMPLE ITERATIONS HIGHER IN PRACTICE

num_burn <- 10 ## Should be ~ 5000
num_save <- 10 ## Should be ~ 5000

set.seed(1234)
f_fried <- function(x) 10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]

gen_data <- function(n_train, n_test, P, sigma) {
  X <- matrix(runif(n_train * P), nrow = n_train)
  mu <- f_fried(X)
  X_test <- matrix(runif(n_test * P), nrow = n_test)
  mu_test <- f_fried(X_test)
```

```
  Y <- mu + sigma * rnorm(n_train)
  Y_test <- mu + sigma * rnorm(n_test)

  return(list(X = X, Y = Y, mu = mu, X_test = X_test, Y_test = Y_test,
              mu_test = mu_test))
}

## Simiulate dataset
sim_data <- gen_data(250, 250, 10, 1)

df <- data.frame(X = sim_data$X, Y = sim_data$Y)
df_test <- data.frame(X = sim_data$X_test, Y = sim_data$Y_test)

## Fit the model

opts <- Opts(num_burn = num_burn, num_save = num_save)
fitted_reg <- softbart_regression(Y ~ ., df, df_test, opts = opts)

## Compute PDP and plot

grid <- seq(from = 0, to = 1, length = 10)
pdp_x4 <- partial_dependence_regression(fitted_reg, df_test, "X.4", grid)
plot(pdp_x4$grid, colMeans(pdp_x4$mu))
```

`pdsoftbart` *Partial dependence plots for SoftBart*

Description

Modified version of the `pdbart` function from the `BayesTree` package; largely supplanted by the `softbart_regression` and `partial_dependence_regression` functions. Runs `softbart` at test observations constructed so that a plot can be created displaying the effect of a single variable or pair of variables.

Usage

```
pdsoftbart(
  X,
  Y,
  xind = NULL,
  levs = NULL,
  levquants = c(0.05, (1:9)/10, 0.95),
  pl = FALSE,
  plquants = c(0.05, 0.95),
  ...
)
```

Arguments

X	Training data covariates.
Y	Training data response.
xind	Variables to create the partial dependence plots for.
levs	List of levels of the covariates to evaluate at.
levquants	Used if levs is not supplied; takes levs to be quantiles of associated predictors.
pl	Create a plot?
plquants	Quantiles for the partial dependence plot.
...	Additional arguments passed to softbart or plot.

Value

Returns a list with components given below.

- fd: A matrix whose (i,j)th value is the ith draw of the partial dependence function for the jth level.
- levs: The list of levels used, each component corresponding to a variable. If the argument levs was supplied it is unchanged. Otherwise, the levels in levs are constructed using the argument levquants.

---

posterior_probs *BART Posterior Inclusion Probabilities*

Description

Computes the posterior inclusion probabilities (PIPs) for the fitted SoftBART model, as well as variable importances and the median probability model (MPM).

Usage

```
posterior_probs(fit)
```

Arguments

fit	An object of class softbart, softbart_regression, or softbart_probit.

Value

A list containing the following:

- varimp: a vector containing the average number of times a predictor was used in a splitting rule.
- post_probs: the posterior inclusion probabilities for each predictor.
- median_probability_model: a vector containing the indicies of the variables included in at least 50 percent of the samples.

Examples

```
## NOTE: SET NUMBER OF BURN IN AND SAMPLE ITERATIONS HIGHER IN PRACTICE

num_burn <- 10 ## Should be ~ 5000
num_save <- 10 ## Should be ~ 5000

set.seed(1234)
f_fried <- function(x) 10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]

gen_data <- function(n_train, n_test, P, sigma) {
  X <- matrix(runif(n_train * P), nrow = n_train)
  mu <- f_fried(X)
  X_test <- matrix(runif(n_test * P), nrow = n_test)
  mu_test <- f_fried(X_test)
  Y <- mu + sigma * rnorm(n_train)
  Y_test <- mu_test + sigma * rnorm(n_test)

  return(list(X = X, Y = Y, mu = mu, X_test = X_test, Y_test = Y_test, mu_test = mu_test))
}

## Simiulate dataset
sim_data <- gen_data(250, 100, 1000, 1)

## Fit the model
fit <- softbart(X = sim_data$X, Y = sim_data$Y, X_test = sim_data$X_test,
                hypers = Hypers(sim_data$X, sim_data$Y, num_tree = 50, temperature = 1),
                opts = Opts(num_burn = num_burn, num_save = num_save, update_tau = TRUE))

## Variable selection

post_probs <- posterior_probs(fit)
plot(post_probs$post_probs)
print(post_probs$median_probability_model)
```

`predict.softbart_probit` *Predict for SoftBart Probit Regression*

Description

Computes predictions from a `softbart_probit` object for new data.

Usage

```
## S3 method for class 'softbart_probit'
predict(object, newdata, iterations = NULL, ...)
```

Arguments

object	A softbart_probit object obtained as output of the softbart_probit function.
newdata	A dataset to construct predictions on.
iterations	The iterations get predictions on; includes all of iterations including burn-in and thinning iterations. Defaults to the saved iterations, running from (num_burn + num_thin):(num_burn + num_thin * num_save).
...	Other arguments passed to predict.

Value

A list containing

- mu: samples of the nonparametric function for each observation, where pnorm(mu) is the success probability.
- mu_mean: posterior mean of mu.
- p: samples of the success probability pnorm(mu) for each observation.
- p_mean: posterior mean of p.

---

predict.softbart_regression

Predict for SoftBart Regression

Description

Computes predictions from a softbart_regression object on new data.

Usage

```
## S3 method for class 'softbart_regression'
predict(object, newdata, iterations = NULL, ...)
```

Arguments

object	A softbart_regression object obtained as output of the softbart_regression() function.
newdata	A dataset to construct predictions on.
iterations	The iterations to get predictions on; includes all of iterations including burn-in and thinning iterations. Defaults to the saved iterations, running from (num_burn + num_thin):(num_burn + num_thin * num_save).
...	Other arguments passed to predict.

Value

A list containing

- mu: samples of the predicted value for each observation and iteration.
- mu_mean: posterior predicted values for each observation.

---

preprocess_df *Preprocess a dataset for use with SoftBart*

Description

Preprocesses a data frame for use with softbart; not needed with other model fitting functions, but may also be useful when designing custom methods with MakeForest. Returns a data matrix X that will work with categorical predictors, and a vector of group indicators; this is required to get sensible variable selection for categorical variables, and should be passed in as the group argument to Hypers.

Usage

```
preprocess_df(X)
```

Arguments

X	A data frame, possibly containing categorical variables stored as factors.

Value

A list containing two elements.

- X: a matrix consisting of the columns of the input data frame, with separate columns for the different levels of categorical variables.
- group: a vector of group memberships of the predictors in X, to be passed as an argument to Hypers.

Examples

```
data(iris)
preprocess_df(iris)
```

quantile_normalize_bart *Quantile normalization for predictors*

Description

Performs a quantile normalization to each column of the matrix `X`.

Usage

```
quantile_normalize_bart(X)
```

Arguments

`X`	A design matrix, should not include a column for the intercept.

Value

A matrix `X_norm` such that each column gives the associated empirical quantile of each observation for each predictor.

Examples

```
X <- matrix(rgamma(100 * 10, shape = 2), nrow = 100)
X <- quantile_normalize_bart(X)
summary(X)
```

rmse *Root mean squared error*

Description

Computes the root mean-squared error between `y` and `yhat`, given by `sqrt(mean((y - yhat)^2))`.

Usage

```
rmse(y, yhat)
```

Arguments

`y`	the realized outcomes
`yhat`	the predicted outcomes

Value

Returns the root mean-squared error.

Examples

```
rmse(c(1,1,1), c(1,0,2))
```

`softbart` *Fits the SoftBart model*

Description

Runs the Markov chain for the semiparametric Gaussian model

$$Y = r(X) + \epsilon$$

and collects the output, where $r(x)$ is modeled using a soft BART model.

Usage

```
softbart(X, Y, X_test, hypers = NULL, opts = Opts(), verbose = TRUE)
```

Arguments

`X`	A matrix of training data covariates.
`Y`	A vector of training data responses.
`X_test`	A matrix of test data covariates
`hypers`	A ;ist of hyperparameter values obtained from `Hypers` function
`opts`	A list of MCMC chain settings obtained from `Opts` function
`verbose`	If `TRUE`, progress of the chain will be printed to the console.

Value

Returns a list with the following components:

- `y_hat_train`: predicted values for the training data for each iteration of the chain.
- `y_hat_test`: predicted values for the test data for each iteration of the chain.
- `y_hat_train_mean`: predicted values for the training data, averaged over iterations.
- `y_hat_test_mean`: predicted values for the test data, averaged over iterations.
- `sigma`: posterior samples of the error standard deviations.
- `sigma_mu`: posterior samples of `sigma_mu`, the standard deviation of the leaf node parameters.
- `s`: posterior samples of `s`.
- `alpha`: posterior samples of `alpha`.
- `beta`: posterior samples of `beta`.
- `gamma`: posterior samples of `gamma`.
- `k`: posterior samples of `k = 0.5 / (sqrt(num_tree) * sigma_mu)`
- `num_leaves_final`: the number of leaves for each tree at the final iteration.

Examples

```
## NOTE: SET NUMBER OF BURN IN AND SAMPLE ITERATIONS HIGHER IN PRACTICE

num_burn <- 10 ## Should be ~ 5000
num_save <- 10 ## Should be ~ 5000

set.seed(1234)
f_fried <- function(x) 10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]

gen_data <- function(n_train, n_test, P, sigma) {
  X <- matrix(runif(n_train * P), nrow = n_train)
  mu <- f_fried(X)
  X_test <- matrix(runif(n_test * P), nrow = n_test)
  mu_test <- f_fried(X_test)
  Y <- mu + sigma * rnorm(n_train)
  Y_test <- mu_test + sigma * rnorm(n_test)

  return(list(X = X, Y = Y, mu = mu, X_test = X_test, Y_test = Y_test, mu_test = mu_test))
}

## Simiulate dataset
sim_data <- gen_data(250, 100, 1000, 1)

## Fit the model
fit <- softbart(X = sim_data$X, Y = sim_data$Y, X_test = sim_data$X_test,
                hypers = Hypers(sim_data$X, sim_data$Y, num_tree = 50, temperature = 1),
                opts = Opts(num_burn = num_burn, num_save = num_save, update_tau = TRUE))

## Plot the fit (note: interval estimates are not prediction intervals,
## so they do not cover the predictions at the nominal rate)
plot(fit)

## Look at posterior model inclusion probabilities for each predictor.

plot(posterior_probs(fit)[["post_probs"]],
     col = ifelse(posterior_probs(fit)[["post_probs"]] > 0.5, scales::muted("blue"),
                  scales::muted("green")),
     pch = 20)


rmse(fit$y_hat_test_mean, sim_data$mu_test)
rmse(fit$y_hat_train_mean, sim_data$mu)
```

`softbart_probit` *SoftBart Probit Regression*

Description

Fits a nonparametric probit regression model with the nonparametric function modeled using a SoftBart model. Specifically, the model takes $\Pr(Y = 1 \mid X = x) = \Phi\{a + r(x)\}$ where a is an offset and $r(x)$ is a Soft BART ensemble.

Usage

```
softbart_probit(
  formula,
  data,
  test_data,
  num_tree = 20,
  k = 1,
  hypers = NULL,
  opts = NULL,
  verbose = TRUE
)
```

Arguments

formula	A model formula with a binary factor on the left-hand-side and predictors on the right-hand-side.
data	A data frame consisting of the training data.
test_data	A data frame consisting of the testing data.
num_tree	The number of trees in the ensemble to use.
k	Determines the standard deviation of the leaf node parameters, which is given by 3 / k / sqrt(num_tree).
hypers	A list of hyperparameters constructed from the Hypers() function (num_tree, k, and sigma_mu are overridden by this function).
opts	A list of options for running the chain constructed from the Opts() function (update_sigma is overridden by this function).
verbose	If TRUE, progress of the chain will be printed to the console.

Value

Returns a list with the following components:

- sigma_mu: samples of the standard deviation of the leaf node parameters
- var_counts: a matrix with a column for each predictor group containing the number of times each predictor is used in the ensemble at each iteration.
- mu_train: samples of the nonparametric function evaluated on the training set; pnorm(mu_train) gives the success probabilities.
- mu_test: samples of the nonparametric function evaluated on the test set; pnorm(mu_train) gives the success probabilities .
- p_train: samples of probabilities on training set.

- p_test: samples of probabilities on test set.
- mu_train_mean: posterior mean of mu_train.
- mu_test_mean: posterior mean of mu_test.
- p_train_mean: posterior mean of p_train.
- p_test_mean: posterior mean of p_test.
- offset: we fit model of the form (offset + BART), with the offset estimated empirically prior to running the chain.
- pnorm_offset: the pnorm of the offset, which is chosen to match the probability of the second factor level.
- formula: the formula specified by the user.
- ecdfs: empirical distribution functions, used by the predict function.
- opts: the options used when running the chain.
- forest: a forest object; see the MakeForest documentation for more details.

Examples

```
## NOTE: SET NUMBER OF BURN IN AND SAMPLE ITERATIONS HIGHER IN PRACTICE

num_burn <- 10 ## Should be ~ 5000
num_save <- 10 ## Should be ~ 5000

set.seed(1234)
f_fried <- function(x) 10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]

gen_data <- function(n_train, n_test, P, sigma) {
  X <- matrix(runif(n_train * P), nrow = n_train)
  mu <- (f_fried(X) - 14) / 5
  X_test <- matrix(runif(n_test * P), nrow = n_test)
  mu_test <- (f_fried(X_test) - 14) / 5
  Y <- factor(rbinom(n_train, 1, pnorm(mu)), levels = c(0,1))
  Y_test <- factor(rbinom(n_test, 1, pnorm(mu_test)), levels = c(0,1))

  return(list(X = X, Y = Y, mu = mu, X_test = X_test, Y_test = Y_test,
              mu_test = mu_test))
}

## Simiulate dataset
sim_data <- gen_data(250, 250, 100, 1)

df <- data.frame(X = sim_data$X, Y = sim_data$Y)
df_test <- data.frame(X = sim_data$X_test, Y = sim_data$Y_test)

## Fit the model

opts <- Opts(num_burn = num_burn, num_save = num_save)
fitted_probit <- softbart_probit(Y ~ ., df, df_test, opts = opts)
```

```
## Plot results

plot(fitted_probit$mu_test_mean, sim_data$mu_test)
abline(a = 0, b = 1)
```

`softbart_regression` *SoftBart Regression*

Description

Fits a semiparametric regression model with the nonparametric function modeled using a SoftBart model.

Usage

```
softbart_regression(
  formula,
  data,
  test_data,
  num_tree = 20,
  k = 2,
  hypers = NULL,
  opts = NULL,
  verbose = TRUE
)
```

Arguments

`formula`	A model formula with a numeric variable on the left-hand-side and predictors on the right-hand-side.
`data`	A data frame consisting of the training data.
`test_data`	A data frame consisting of the testing data.
`num_tree`	The number of trees in the ensemble to use.
`k`	Determines the standard deviation of the leaf node parameters, which is given by `3 / k / sqrt(num_tree)`.
`hypers`	A list of hyperparameters constructed from the `Hypers()` function (`num_tree`, `k`, and `sigma_mu` are overridden by this function).
`opts`	A list of options for running the chain constructed from the `Opts()` function (`update_sigma` is overridden by this function).
`verbose`	If `TRUE`, progress of the chain will be printed to the console.

Value

Returns a list with the following components:

- `sigma_mu`: samples of the standard deviation of the leaf node parameters.
- `sigma`: samples of the error standard deviation.
- `var_counts`: a matrix with a column for each predictor group containing the number of times each predictor is used in the ensemble at each iteration.
- `mu_train`: samples of the nonparametric function evaluated on the training set.
- `mu_test`: samples of the nonparametric function evaluated on the test set.
- `mu_train_mean`: posterior mean of `mu_train`.
- `mu_test_mean`: posterior mean of `mu_test`.
- `formula`: the formula specified by the user.
- `ecdfs`: empirical distribution functions, used by the `predict` function.
- `opts`: the options used when running the chain.
- `mu_Y, sd_Y`: used with the predict function to transform predictions.
- `forest`: a forest object; see the `MakeForest` documentation for more details.

Examples

```
## NOTE: SET NUMBER OF BURN IN AND SAMPLE ITERATIONS HIGHER IN PRACTICE

num_burn <- 10 ## Should be ~ 5000
num_save <- 10 ## Should be ~ 5000

set.seed(1234)
f_fried <- function(x) 10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]

gen_data <- function(n_train, n_test, P, sigma) {
  X <- matrix(runif(n_train * P), nrow = n_train)
  mu <- f_fried(X)
  X_test <- matrix(runif(n_test * P), nrow = n_test)
  mu_test <- f_fried(X_test)
  Y <- mu + sigma * rnorm(n_train)
  Y_test <- mu + sigma * rnorm(n_test)

  return(list(X = X, Y = Y, mu = mu, X_test = X_test, Y_test = Y_test,
              mu_test = mu_test))
}

## Simiulate dataset
sim_data <- gen_data(250, 250, 100, 1)

df <- data.frame(X = sim_data$X, Y = sim_data$Y)
df_test <- data.frame(X = sim_data$X_test, Y = sim_data$Y_test)

## Fit the model
```

```
opts <- Opts(num_burn = num_burn, num_save = num_save)
fitted_reg <- softbart_regression(Y ~ ., df, df_test, opts = opts)

## Plot results

plot(colMeans(fitted_reg$mu_test), sim_data$mu_test)
abline(a = 0, b = 1)
```

---

vc_softbart_regression *SoftBart Varying Coefficient Regression*

Description

Fits a semiparametric varying coefficient regression model with the nonparametric slope and intercept

$$Y = \alpha(X) + Z\beta(X) + \epsilon$$

using a soft BART model.

Usage

```
vc_softbart_regression(
  formula,
  linear_var_name,
  data,
  test_data,
  num_tree = 20,
  k = 2,
  hypers_intercept = NULL,
  hypers_slope = NULL,
  opts = NULL,
  verbose = TRUE,
  warn = TRUE
)
```

Arguments

formula	A model formula with a numeric variable on the left-hand-side and non-linear predictors on the right-hand-side.
linear_var_name	A string containing the variable in the data that is to be treated linearly.
data	A data frame consisting of the training data.
test_data	A data frame consisting of the testing data.
num_tree	The number of trees in the ensemble to use.

k	Determines the standard deviation of the leaf node parameters, which is given by 3 / k / sqrt(num_tree) (intercept) and defaults to 1/k/sqrt(num_tree) (slope). This can be modified for the slope by specifying your own hyperparameter.
hypers_intercept	A list of hyperparameters constructed from the Hypers() function (num_tree, k, and sigma_mu are overridden by this function).
hypers_slope	A list of hyperparameters constructed from the Hypers() function (num_tree is overridden by this function).
opts	A list of options for running the chain constructed from the Opts() function (update_sigma is overridden by this function).
verbose	If TRUE, progress of the chain will be printed to the console.
warn	If TRUE, remind the user that they probably don't want the linear term to be included in the formula for the nonlinear part.

Value

Returns a list with the following components

- sigma_mu_alpha: samples of the standard deviation of the leaf node parameters for the intercept.
- sigma_mu_beta: samples of the standard deviation of the leaf node parameters for the slope.
- sigma: samples of the error standard deviation.
- var_counts_alpha: a matrix with a column for each predictor group containing the number of times each predictor is used in the ensemble at each iteration for the intercept.
- var_counts_beta: a matrix with a column for each predictor group containing the number of times each predictor is used in the ensemble at each iteration for the slope.
- alpha_train: samples of the nonparametric intercept evaluated on the training set.
- alpha_test: samples of the nonparametric intercept evaluated on the test set.
- beta_train: samples of the nonparametric slope evaluated on the training set.
- beta_test: samples of the nonparametric slope evaluated on the test set.
- mu_train: samples of the predictions evaluated on the training set.
- mu_test: samples of the predictions evaluated on the test set.
- formula: the formula specified by the user.
- ecdfs: empirical distribution functions, used by the predict function.
- opts: the options used when running the chain.
- mu_Y, sd_Y: used with the predict function to transform predictions.
- alpha_forest: a forest object for the intercept; see the MakeForest documentation for more details.
- beta_forest: a forest object for the slope; see the MakeForest documentation for more details.

Examples

```
## NOTE: SET NUMBER OF BURN IN AND SAMPLE ITERATIONS HIGHER IN PRACTICE

num_burn <- 10 ## Should be ~ 5000
num_save <- 10 ## Should be ~ 5000

set.seed(1234)
f_fried <- function(x) 10 * sin(pi * x[,1] * x[,2]) + 20 * (x[,3] - 0.5)^2 +
  10 * x[,4] + 5 * x[,5]

gen_data <- function(n_train, n_test, P, sigma) {
  X <- matrix(runif(n_train * P), nrow = n_train)
  Z <- rnorm(n_train)
  r <- f_fried(X)
  mu <- Z * r
  X_test <- matrix(runif(n_test * P), nrow = n_test)
  Z_test <- rnorm(n_test)
  r_test <- f_fried(X_test)
  mu_test <- Z_test * r_test
  Y <- mu + sigma * rnorm(n_train)
  Y_test <- mu + sigma * rnorm(n_test)

  return(list(X = X, Y = Y, Z = Z, r = r, mu = mu, X_test = X_test, Y_test =
              Y_test, Z_test = Z_test, r_test = r_test, mu_test = mu_test))
}

## Simiulate dataset
sim_data <- gen_data(250, 250, 100, 1)

df <- data.frame(X = sim_data$X, Y = sim_data$Y, Z = sim_data$Z)
df_test <- data.frame(X = sim_data$X_test, Y = sim_data$Y_test, Z = sim_data$Z_test)

## Fit the model

opts <- Opts(num_burn = num_burn, num_save = num_save)
fitted_vc <- vc_softbart_regression(Y ~ . -Z, "Z", df, df_test, opts = opts)

## Plot results

plot(colMeans(fitted_vc$mu_test), sim_data$mu_test)
abline(a = 0, b = 1)
```

Index