Package: sparseVCBART (via r-universe)

November 2, 2026

Type Package

Title Sparse Varying Coefficient BART with Global-Local Priors"

Version 1.0.0

Date 2026-05-13

Description Fits sparse linear varying coefficient models (VCMs), which assert a linear relationship between an outcome and several covariates that is allowed to change as functions of additional variables known as effect modifiers. Designed for high-dimensional settings where the number of covariates (i.e., number of slopes) is comparable to or larger than the number of observations. Approximates the coefficient functions using a version of Bayesian Additive Regression Trees that can perform global-local shrinkage. For more details see Ghosh, Bhogale, and Deshpande (2026+) <doi:10.48550/arXiv.2510.08204>.

URL https://github.com/ghoshstats/sparseVCBART

License GPL (>= 3)

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, MASS

NeedsCompilation yes

Author Soham Ghosh [aut], Sameer K. Deshpande [cre, aut] (ORCID: <https://orcid.org/0000-0003-4116-5533>)

Maintainer Sameer K. Deshpande <sameer.deshpande@wisc.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-05-19 08:42:26 UTC

RemoteUrl https://github.com/cran/sparseVCBART

RemoteRef HEAD

RemoteSha 7b13f1bb4cf5d4a766dd0bf5d515584c9675faac

Contents

predict_betas 2
sparseVCBART_cs 3
sparseVCBART_ind 6
summarize_beta 13

Index **14**

`predict_betas` *Compute posterior predictive evaluates of covariate effect functions.*

Description

Given an object returned by `VCBART_ind` or `VCBART_cs` and matrices of continuous and categorical modifiers, returns MCMC samples of the coefficient functions evaluated the provided points.

Usage

```
predict_betas(fit,
              Z_cont = matrix(0, nrow = 1, ncol = 1),
              Z_cat =  matrix(0, nrow = 1, ncol = 1),
              verbose = TRUE)
```

Arguments

`fit`	A list returned by `VCBART_ind` or `VCBART_cs`
`Z_cont`	Matrix of continuous modifiers at which you wish to evaluate the covariate effect functions. Default is a 1x1 matrix, which signals that no continuous modifiers are required for these evaluations.
`Z_cat`	Integer matrix of categorical modifiers at which you wish to evaluate the covariate effect functions. Default is a 1x1 matrix, which signals that no continuous modifiers are required for these evaluations.
`verbose`	Boolean indicating whether the code should print its progress (`TRUE`). Default is `TRUE`.

Value

An array of size nd x N x (p+1) where nd is the total number of MCMC draws, N is the total number of points at which you are evaluating the covariate effect functions (i.e. `nrow(Z_cont)` or `nrow(Z_cat)`), and p is the number of covariates. Note that the intercept function is included as the first slice in the third dimension.

sparseVCBART_cs *Fit a sparse VCBART model with compound symmetry error structure*

Description

Fit a sparse varying coefficient model by approximating each coefficient function with an ensemble of regression trees. Global-local priors on the regression tree outputs facilitate adaptive shrinkage of the coefficient functions. Assumes a compound symmetry error structure in which the residual errors for a given subject are equally correlated. This is equivalent to assuming that there is a normally distributed random effect per subject.

Usage

```
sparseVCBART_cs(Y_train,subj_id_train, ni_train,X_train,
          Z_cont_train = matrix(0, nrow = 1, ncol = 1),
          Z_cat_train = matrix(0L, nrow = 1, ncol = 1),
          X_test = matrix(0, nrow = 1, ncol = 1),
          Z_cont_test = matrix(0, nrow = 1, ncol = 1),
          Z_cat_test = matrix(0, nrow = 1, ncol = 1),
          unif_cuts = rep(TRUE, times = ncol(Z_cont_train)),
          cutpoints_list = NULL,
          cat_levels_list = NULL,
          edge_mat_list = NULL,
          graph_split = rep(FALSE, times = ncol(Z_cat_train)),
          sparse = TRUE,
          rho = 0.9,
          M = 50,
          mu0 = NULL, tau = NULL, nu = NULL, lambda = NULL,
          nd = 1000, burn = 1000, thin = 1,
          save_samples = TRUE, save_trees = TRUE,
          verbose = TRUE, print_every = floor( (nd*thin + burn)/10))
```

Arguments

Y_train	Vector of continous responses for training data
ni_train	Vector containing the number of observations per subject in the training data.
subj_id_train	Vector of length length(Y_train) that records which subject contributed each observation. Subjects should be numbered sequentially from 1 to length(ni_train).
X_train	Matrix of covariates for training observations. Do not include intercept as the first column.
Z_cont_train	Matrix of continuous modifiers for training data. Note, modifiers must be rescaled to lie in the interval [-1,1]. Default is a 1x1 matrix, which signals that there are no continuous modifiers in the training data.
Z_cat_train	Integer matrix of categorical modifiers for training data. Note categorical levels should be 0-indexed. That is, if a categorical modifier has 10 levels, the values

should run from 0 to 9. Default is a 1x1 matrix, which signals that there are no categorical modifiers in the training data.

`X_test` Matrix of covariate for testing observations. Default is a 1x1 matrix, which signals that testing data is not provided.

`Z_cont_test` Matrix of continuous modifiers for testing data. Default is a 1x1 matrix, which signals that testing data is not provided.

`Z_cat_test` Integer matrix of categorical modifiers for testing data. Default is a 1x1 matrix, which signals that testing data is not provided.

`unif_cuts` Vector of logical values indicating whether cutpoints for each continuous modifier should be drawn from a continuous uniform distribution (`TRUE`) or a discrete set (`FALSE`) specified in `cutpoints_list`. Default is `TRUE` for each variable in `Z_cont_train`

`cutpoints_list` List of length `ncol(Z_cont_train)` containing a vector of cutpoints for each continuous modifier. By default, this is set to `NULL` so that cutpoints are drawn uniformly from a continuous distribution.

`cat_levels_list`

List of length `ncol(Z_cat_train)` containing a vector of levels for each categorical modifier. If the j-th categorical modifier contains L levels, `cat_levels_list[[j]]` should be the vector `0:(L-1)`. Default is `NULL`, which corresponds to the case that no categorical modifiers are available.

`edge_mat_list` List of adjacency matrices if any of the categorical modifiers are network-structured. Default is `NULL`, which corresponds to the case that there are no network-structured categorical modifiers.

`graph_split` Vector of logicals indicating whether each categorical modifier is network-structured. Default is `rep(FALSE, times = ncol(Z_cat_train))`.

`sparse` Logical, indicating whether or not to perform variable selection in each tree ensemble based on a sparse Dirichlet prior rather than uniform prior; see Linero 2018. Default is `TRUE`

`rho` Initial auto-correlation parameter for compound symmetry error structure. Must be between 0 and 1. Default is 0.9.

`M` Number of trees in each ensemble. Default is 50.

`mu0` Prior mean for jumps/leaf parameters. Default is 0 for each beta function. If supplied, must be a vector of length `1 + ncol(X_train)`.

`tau` Prior standard deviation for jumps/leaf parameters. Default is `1/sqrt(M)` for each beta function. If supplied, must be a vector of length `1 + ncol(X_train)`.

`nu` Degrees of freedom for scaled-inverse chi-square prior on sigma^2. Default is 3.

`lambda` Scale hyperparameter for scaled-inverse chi-square prior on sigma^2. Default places 90% prior probability that sigma is less than `sd(Y_train)`.

`nd` Number of posterior draws to return. Default is 1000.

`burn` Number of MCMC iterations to be treated as "warmup" or "burn-in". Default is 1000.

`thin` Number of post-warmup MCMC iteration by which to thin. Default is 1.

save_samples	Logical, indicating whether to return all posterior samples. Default is TRUE. If FALSE, only posterior mean is returned.
save_trees	Logical, indicating whether or not to save a text-based representation of the tree samples. This representation can be passed to predict_flexBART to make predictions at a later time. Default is FALSE.
verbose	Logical, inciating whether to print progress to R console. Default is TRUE.
print_every	As the MCMC runs, a message is printed every print_every iterations. Default is floor((nd*thin + burn)/10) so that only 10 messages are printed.

Details

Given p covariates $X_1, \ldots, X_p$ and r effect modifiers $Z_1, \ldots, Z_r$, the varying coefficient model asserts that

$E[Y|X = x, Z =] = \beta_0(z) + \beta_1(z) * x_1 + ...\beta_p(z) * X_p.$

That is, for any r-vector Z, the relationships between X and Y is linear. However, the specific relationship is allowed to vary with respect tp Z. This function approximates the functions $\beta_j(z)$ using ensembles of regression trees and specifies global-local priors on the regression tree outputs. Thanks to the ensuing adaptive shrinkage, this function can estimate *sparse* varying coefficient models, which are useful in settings where p is comparable to or greater than n. This function assumes that the within-subject errors are equi-correlated (i.e. a compound symmetry error structure).

Value

A list containing

y_mean	Mean of the training observations (needed by predict_VCBART)
y_sd	Standard deviation of the training observations (needed by predict_VCBART)
x_mean	Vector of means of columns of X_train, including the intercept (needed by predict_VCBART).
x_sd	Vector of standard deviations of X_trian, including the intercept (needed by predict_VCBART).
yhat.train.mean	Vector containing posterior mean of evaluations of regression function E[y|x,z] on training data.
betahat.train.mean	Matrix with length(Y_train) rows and ncol(X_train)+1 columns containing the posterior mean of evaluations of each coefficient function evaluated on the training data. Each row corresponds to a training set observation and each colunn corresponds to a coefficient function. Note the first column is for the intercept function.
yhat.train	Matrix with nd rows and length(Y_train) columns. Each row corresponds to a posterior sample of the regression function E[y|x,z] and each column corresponds to a training set observation. Only returned if save_samples == TRUE.

betahat.train Array of dimension with nd x length(Y_train) x ncol(X_train)+1 containing posterior samples of evaluations of the coefficient functions. The first dimension corresponds to posterior samples/MCMC iterations, the second dimension corresponds to individual training set observations, and the third dimension corresponds to coefficient functions. Only returned if save_samples == TRUE.

yhat.test.mean Vector containing posterior mean of evaluations of regression function E[y|x,z] on testing data.

betahat.test.mean Matrix with nrow(X_test) rows and ncol(X_testn)+1 columns containing the posterior mean of evaluations of each coefficient function evaluated on the training data. Each row corresponds to a training set observation and each colunn corresponds to a coefficient function. Note the first column is for the intercept function.

yhat.test Matrix with nd rows and nrow(X_test) columns. Each row corresponds to a posterior sample of the regression function E[y|x,z] and each column corresponds to a testing set observation. Only returned if save_samples == TRUE.

betahat.test Array of size nd x nrow(X_test) x ncol(X_test)+1 containing posterior samples of evaluations of the coefficient functions. The first dimension corresponds to posterior samples/MCMC iterations, the second dimension corresponds to individual training set observations, and the third dimension corresponds to coefficient functions. Only returned if save_samples == TRUE.

sigma Vector containing ALL samples of the residual standard deviation, including warmup.

rho Vector containing ALL samples of the auto-correlation parameter rho, including warmup.

varcounts Array of size nd x R x ncol(X)+1 that counts the number of times a variable was used in a decision rule in each posterior sample of each ensemble. Here R is the total number of potential modifiers (i.e. R = ncol(Z_cont_train) + ncol(Z_cat_train)).

theta If sparse=TRUE, an array of size nd x R ncol(X)+1 containing samples of the variable splitting probabilities.

trees A list (of length nd) of lists (of length ncol(X_train)+1) of character vectors (of length M) containing textual representations of the regression trees. The string for the s-th sample of the m-th tree in the j-th ensemble is contaiend in trees[[s]][[j]][m]. These strings are parsed by predict_VCBART to reconstruct the C++ representations of the sampled trees.

---

sparseVCBART_ind *Fit a sparse VCBART model with independent error structure*

Description

Fit a sparse varying coefficient model by approximating each coefficient function with an ensemble of regression trees. Global-local priors on the regression tree outputs facilitate adaptive shrinkage of the coefficient functions. Assumes residual errors are independent within and between subjects.

Usage

```
sparseVCBART_ind(Y_train,subj_id_train, ni_train,X_train,
           Z_cont_train = matrix(0, nrow = 1, ncol = 1),
           Z_cat_train = matrix(0L, nrow = 1, ncol = 1),
           X_test = matrix(0, nrow = 1, ncol = 1),
           Z_cont_test = matrix(0, nrow = 1, ncol = 1),
           Z_cat_test = matrix(0, nrow = 1, ncol = 1),
           unif_cuts = rep(TRUE, times = ncol(Z_cont_train)),
           cutpoints_list = NULL,
           cat_levels_list = NULL,
           edge_mat_list = NULL,
           graph_split = rep(FALSE, times = ncol(Z_cat_train)),
           sparse = TRUE,
           M = 50,
           mu0 = NULL, tau = NULL, nu = NULL, lambda = NULL,
           nd = 1000, burn = 1000, thin = 1,
           save_samples = TRUE, save_trees = TRUE,
           verbose = TRUE, print_every = floor( (nd*thin + burn)/10))
```

Arguments

`Y_train`	Vector of continous responses for training data
`ni_train`	Vector containing the number of observations per subject in the training data.
`subj_id_train`	Vector of length `length(Y_train)` that records which subject contributed each observation. Subjects should be numbered sequentially from 1 to `length(ni_train)`.
`X_train`	Matrix of covariates for training observations. Do not include intercept as the first column.
`Z_cont_train`	Matrix of continuous modifiers for training data. Note, modifiers must be rescaled to lie in the interval [-1,1]. Default is a 1x1 matrix, which signals that there are no continuous modifiers in the training data.
`Z_cat_train`	Integer matrix of categorical modifiers for training data. Note categorical levels should be 0-indexed. That is, if a categorical modifier has 10 levels, the values should run from 0 to 9. Default is a 1x1 matrix, which signals that there are no categorical modifiers in the training data.
`X_test`	Matrix of covariate for testing observations. Default is a 1x1 matrix, which signals that testing data is not provided.
`Z_cont_test`	Matrix of continuous modifiers for testing data. Default is a 1x1 matrix, which signals that testing data is not provided.
`Z_cat_test`	Integer matrix of categorical modifiers for testing data. Default is a 1x1 matrix, which signals that testing data is not provided.
`unif_cuts`	Vector of logical values indicating whether cutpoints for each continuous modifier should be drawn from a continuous uniform distribution (`TRUE`) or a discrete set (`FALSE`) specified in `cutpoints_list`. Default is `TRUE` for each variable in `Z_cont_train`

`cutpoints_list` List of length `ncol(Z_cont_train)` containing a vector of cutpoints for each continuous modifier. By default, this is set to `NULL` so that cutpoints are drawn uniformly from a continuous distribution.

`cat_levels_list`
List of length `ncol(Z_cat_train)` containing a vector of levels for each categorical modifier. If the j-th categorical modifier contains L levels, `cat_levels_list[[j]]` should be the vector `0:(L-1)`. Default is `NULL`, which corresponds to the case that no categorical modifiers are available.

`edge_mat_list` List of adjacency matrices if any of the categorical modifiers are network-structured. Default is `NULL`, which corresponds to the case that there are no network-structured categorical modifiers.

`graph_split` Vector of logicals indicating whether each categorical modifier is network-structured. Default is `rep(FALSE, times = ncol(Z_cat_train))`.

`sparse` Logical, indicating whether or not to perform variable selection in each tree ensemble based on a sparse Dirichlet prior rather than uniform prior; see Linero 2018. Default is `TRUE`

`M` Number of trees in each ensemble. Default is 50.

`mu0` Prior mean for jumps/leaf parameters. Default is 0 for each beta function. If supplied, must be a vector of length `1 + ncol(X_train)`.

`tau` Prior standard deviation for jumps/leaf parameters. Default is `1/sqrt(M)` for each beta function. If supplied, must be a vector of length `1 + ncol(X_train)`.

`nu` Degrees of freedom for scaled-inverse chi-square prior on sigma^2. Default is 3.

`lambda` Scale hyperparameter for scaled-inverse chi-square prior on sigma^2. Default places 90% prior probability that sigma is less than `sd(Y_train)`.

`nd` Number of posterior draws to return. Default is 1000.

`burn` Number of MCMC iterations to be treated as "warmup" or "burn-in". Default is 1000.

`thin` Number of post-warmup MCMC iteration by which to thin. Default is 1.

`save_samples` Logical, indicating whether to return all posterior samples. Default is `TRUE`. If `FALSE`, only posterior mean is returned.

`save_trees` Logical, indicating whether or not to save a text-based representation of the tree samples. This representation can be passed to `predict_flexBART` to make predictions at a later time. Default is `FALSE`.

`verbose` Logical, inciating whether to print progress to R console. Default is `TRUE`.

`print_every` As the MCMC runs, a message is printed every `print_every` iterations. Default is `floor( (nd*thin + burn)/10)` so that only 10 messages are printed.

Details

Given p covariates $X_1, \ldots, X_p$ and r effect modifiers $Z_1, \ldots, Z_r$, the varying coefficient model asserts that

$E[Y|X = x, Z =] = \beta_0(z) + \beta_1(z) * x_1 + ...\beta_p(z) * X_p.$

That is, for any r-vector Z, the relationships between X and Y is linear. However, the specific relationship is allowed to vary with respect tp Z. This function approximates the functions $\beta_j(z)$ using ensembles of regression trees and specifies global-local priors on the regression tree outputs. Thanks to the ensuing adaptive shrinkage, this function can estimate *sparse* varying coefficient models, which are useful in settings where p is comparable to or greater than n. This function assumes that the within-subject errors are independent.

Value

A list containing

`y_mean` Mean of the training observations (needed by `predict_VCBART`)

`y_sd` Standard deviation of the training observations (needed by `predict_VCBART`)

`x_mean` Vector of means of columns of `X_train`, including the intercept (needed by `predict_VCBART`).

`x_sd` Vector of standard deviations of `X_trian`, including the intercept (needed by `predict_VCBART`).

`yhat.train.mean`
Vector containing posterior mean of evaluations of regression function E[y|x,z] on training data.

`betahat.train.mean`
Matrix with `length(Y_train)` rows and `ncol(X_train)+1` columns containing the posterior mean of evaluations of each coefficient function evaluated on the training data. Each row corresponds to a training set observation and each column corresponds to a coefficient function. Note the first column is for the intercept function.

`yhat.train` Matrix with `nd` rows and `length(Y_train)` columns. Each row corresponds to a posterior sample of the regression function E[y|x,z] and each column corresponds to a training set observation. Only returned if `save_samples == TRUE`.

`betahat.train` Array of dimension with `nd` x `length(Y_train)` x `ncol(X_train)+1` containing posterior samples of evaluations of the coefficient functions. The first dimension corresponds to posterior samples/MCMC iterations, the second dimension corresponds to individual training set observations, and the third dimension corresponds to coefficient functions. Only returned if `save_samples == TRUE`.

`yhat.test.mean` Vector containing posterior mean of evaluations of regression function E[y|x,z] on testing data.

`betahat.test.mean`
Matrix with `nrow(X_test)` rows and `ncol(X_testn)+1` columns containing the posterior mean of evaluations of each coefficient function evaluated on the training data. Each row corresponds to a training set observation and each column corresponds to a coefficient function. Note the first column is for the intercept function.

`yhat.test` Matrix with `nd` rows and `nrow(X_test)` columns. Each row corresponds to a posterior sample of the regression function E[y|x,z] and each column corresponds to a testing set observation. Only returned if `save_samples == TRUE`.

`betahat.test` Array of size nd x `nrow(X_test)` x `ncol(X_test)+1` containing posterior samples of evaluations of the coefficient functions. The first dimension corresponds to posterior samples/MCMC iterations, the second dimension corresponds to individual training set observations, and the third dimension corresponds to coefficient functions. Only returned if `save_samples == TRUE`.

`sigma` Vector containing ALL samples of the residual standard deviation, including warmup.

`varcounts` Array of size nd x R x `ncol(X)+1` that counts the number of times a variable was used in a decision rule in each posterior sample of each ensemble. Here R is the total number of potential modifiers (i.e. `R = ncol(Z_cont_train) + ncol(Z_cat_train)`).

`theta` If `sparse=TRUE`, an array of size nd x R `ncol(X)+1` containing samples of the variable splitting probabilities.

`trees` A list (of length nd) of lists (of length `ncol(X_train)+1`) of character vectors (of length `M`) containing textual representations of the regression trees. The string for the s-th sample of the m-th tree in the j-th ensemble is contaiend in `trees[[s]][[j]][m]`. These strings are parsed by `predict_VCBART` to reconstruct the C++ representations of the sampled trees.

References

Ghosh, S., Bhogale, S., and Deshpande, S.K (2026+). Fitting sparse high-dimensional varying-coefficient models with Bayesian regression tree ensembles. *arXiv preprint* arXiv:2510.08204. doi:10.48550/arXiv.2510.08204

Examples

```
############
# True beta functions
beta0_true <- function(Z){
  tmp_Z <- (Z+1)/2
  return( 3 * tmp_Z[,1] +
            (2 - 5 * (tmp_Z[,2] > 0.5)) * sin(pi * tmp_Z[,1]) -
            2 * (tmp_Z[,2] > 0.5))
}
beta1_true <- function(Z){
  tmp_Z <- (Z+1)/2
  return(sin(2*tmp_Z[,1] + 0.5)/(4*tmp_Z[,1] + 1) + (2*tmp_Z[,1] - 0.5)^3)
}
beta2_true <- function(Z){
  tmp_Z <- (Z+1)/2
  return( (3 - 3*cos(6*pi*tmp_Z[,1]) * tmp_Z[,1]^2) * (tmp_Z[,1] > 0.6) -
            (10 * sqrt(tmp_Z[,1])) * (tmp_Z[,1] < 0.25) )
}
beta3_true <- function(Z){
  return(rep(1, times = nrow(Z)))
}
beta4_true <- function(Z){
  tmp_Z <- (Z+1)/2
  return(10 * sin(pi * tmp_Z[,1] * tmp_Z[,2]) +
```

```
           20 * (tmp_Z[,3] - 0.5)^2 +
           10 * tmp_Z[,4] + 5 * tmp_Z[,5])
}
beta5_true <- function(Z){
  tmp_Z <- (Z+1)/2
  return(exp(sin((0.9 * (tmp_Z[,1] + 0.48))^10)) +
           tmp_Z[,2] * tmp_Z[,3] + tmp_Z[,4])
}
beta6_true <- function(Z){
  return(rep(0, times = nrow(Z)))
}
beta7_true <- function(Z){
  return(rep(0, times = nrow(Z)))
}
beta8_true <- function(Z){
  return(rep(0, times = nrow(Z)))
}
beta9_true <- function(Z){
  return(rep(0, times = nrow(Z)))
}


## Set problem dimensions
n_train <- 250
n_test <- 25
sigma <- 1

set.seed(417)
n_all <- n_train + n_test
ni_all <- rep(4, times = n_all) # 4 observations per subject
subj_id_all <- rep(1:n_all, each = 4) # give every subject an id number
N_all <- sum(ni_all) # total number of observations

p <- 9 # number of covariates
R_cont <- 20 # number of continuous modifiers
R_cat <- 0 # number of categorical modifiers
R <- R_cont + R_cat

## Generate data
Sigma_X <-  (0.5)^(abs(outer(1:p, 1:p, FUN = "-"))) # covariates are all correlated
X_all <- MASS::mvrnorm(N_all, mu = rep(0, times = p), Sigma = Sigma_X)
Z_cont_all <- matrix(runif(N_all * R_cont, min = -1, max = 1), nrow = N_all, ncol = R_cont)
beta0_all <- beta0_true(Z_cont_all)
beta1_all <- beta1_true(Z_cont_all)
beta2_all <- beta2_true(Z_cont_all)
beta3_all <- beta3_true(Z_cont_all)
beta4_all <- beta4_true(Z_cont_all)
beta5_all <- beta5_true(Z_cont_all)
beta6_all <- beta6_true(Z_cont_all)
beta7_all <- beta7_true(Z_cont_all)
beta8_all <- beta8_true(Z_cont_all)
beta9_all <- beta9_true(Z_cont_all)
```

```
beta_all <-
  cbind(
    beta0_all, beta1_all, beta2_all, beta3_all, beta4_all,
    beta5_all,beta6_all, beta7_all, beta8_all, beta9_all)

mu_all <-
  beta0_all + X_all[,1] * beta1_all + X_all[,2] * beta2_all + X_all[,3] * beta3_all +
  X_all[,4] * beta4_all + X_all[,5] * beta5_all + X_all[,6] * beta6_all +
  X_all[,7] * beta7_all + X_all[,8] * beta8_all + X_all[,9] * beta9_all

Y_all <- mu_all + sigma * rnorm(n = N_all, mean = 0, sd = 1)

## Token run to ensure installation works
fit <-
  sparseVCBART_ind(
    Y_train = Y_all,
    subj_id_train = subj_id_all,
    ni_train = ni_all,
    X_train = X_all,
    Z_cont_train = Z_cont_all,
    nd = 5, burn = 5,
    verbose = FALSE)

##############
# Create a training/testing split
# since we randomly generated X and Z,
# we can just use first n_train subjects as train
############
train_subjects <- 1:n_train
test_subjects <- (n_train+1):(n_train + n_test)

test_index <- which(subj_id_all %in% test_subjects)
train_index <- which(subj_id_all %in% train_subjects)

ni_train <- ni_all[train_subjects]
ni_test <- ni_all[test_subjects]

N_train <- sum(ni_train)
N_test <- sum(ni_test)

X_train <- X_all[train_index,]
X_test <- X_all[test_index,]

Z_cont_train <- Z_cont_all[train_index,]
Z_cont_test <- Z_cont_all[test_index,]
subj_id_train <- rep(1:n_train, times = ni_train)

Y_train <- Y_all[train_index]
Y_test <- Y_all[test_index]

beta_train <- beta_all[train_index,]
beta_test <- beta_all[test_index,]
```

```
fit <-
  sparseVCBART_ind(
    Y_train = Y_train,
    subj_id_train = subj_id_train,
    ni_train = ni_train,
    X_train = X_train,
    Z_cont_train = Z_cont_train,
    X_test = X_test,
    Z_cont_test = Z_cont_test,
    verbose = FALSE)

plot(c(beta_train, beta_test),
     c(fit$betahat.train.mean, fit$betahat.test.mean),
     xlab = "Actual", ylab = "Fitted")
abline(a = 0, b = 1, col = 'blue')
```

`summarize_beta` *Compute posterior mean and 95% credible interval for evaluations of each coefficient function.*

Description

Given an array of posterior samples of coefficient function evaluations, returns the posterior mean and 95% credible interval for each evaluation.

Usage

```
summarize_beta(beta_samples,
               level = 0.95,
               include_median = FALSE,
               na.rm = FALSE)
```

Arguments

`beta_samples`	An array, returned by `VCBART_ind`, `VCBART_cs`, or `predict_betas` of posterior samples of coefficient function evaluations
`level`	Default is `0.95`.
`include_median`	Default is `FALSE`.
`na.rm`	Default is `FALSE`.

Value

An array of size N x 3 x p where N is the number of inputs at which the coefficient functions are evaluated (i.e. N = `dim(beta_samples)[2]`) and p is the total number of coefficient functions including the intercept (i.e. p = `dim(beta_samples)[3]`). The j-th slice is an N x 3 matrix where the columns correspond to the posterior mean, 2.5% quantile, and 97.5% quantile of each evaluation of the (j-1)-th coefficient function. Note the effect of predictor X_j is the (j+1)-st coefficient function.

Index

predict_betas, 2

sparseVCBART_cs, 3
sparseVCBART_ind, 6
summarize_beta, 13