Package ‘bartMachine’

August 13, 2020

Type Package

Title Bayesian Additive Regression Trees

Version 1.2.5.1

Date 2020-08-05

Author Adam Kapelner and Justin Bleich (R package)

Maintainer Adam Kapelner <kapelner@qc.cuny.edu>

Description An advanced implementation of Bayesian Additive Regression Trees with expanded features for data analysis and visualization.

License GPL-3

Depends R (>= 2.14.0), rJava (>= 0.9-8), bartMachineJARs (>= 1.0), randomForest, missForest

Imports graphics, grDevices, stats

SystemRequirements Java (>= 7.0)

NeedsCompilation no

Repository CRAN

Date/Publication 2020-08-13 05:20:09 UTC

R topics documented:

`automobile` *Data concerning automobile prices.*

Description

The `automobile` data frame has 201 rows and 25 columns and concerns automobiles in the 1985 Auto Imports Database. The response variable, `price`, is the log selling price of the automobile. There are 7 categorical predictors and 17 continuous / integer predictors which are features of the automobiles. 41 automobiles have missing data in one or more of the feature entries. This dataset is true to the original except with a few of the predictors dropped.

Usage

```
data(automobile)
```

Source

K Bache and M Lichman. UCI machine learning repository, 2013. http://archive.ics.uci.edu/ml/datasets/Automobile bartMachine *Build a BART Model*

Description

Builds a BART model for regression or classification.

Usage

```
bartMachine(X = NULL, y = NULL, Xy = NULL,
num_trees = 50,
num_burn_in = 250,
num_iterations_after_burn_in = 1000,
alpha = 0.95, beta = 2, k = 2, q = 0.9, nu = 3,
prob_rule_class = 0.5,
mh_prob_steps = c(2.5, 2.5, 4)/9,
debug_log = FALSE,
run_in_sample = TRUE,
s_sq_y = "mse",
sig_sq_est = NULL,
cov_prior_vec = NULL,
use_missing_data = FALSE,
covariates_to_permute = NULL,
num_rand_samps_in_library = 10000,
use_missing_data_dummies_as_covars = FALSE,
replace_missing_data_with_x_j_bar = FALSE,
impute_missingness_with_rf_impute = FALSE,
impute_missingness_with_x_j_bar_for_lm = TRUE,
mem_cache_for_speed = TRUE,
flush_indices_to_save_RAM = TRUE,
serialize = FALSE,
seed = NULL,
verbose = TRUE)

build_bart_machine(X = NULL, y = NULL, Xy = NULL,
num_trees = 50,
num_burn_in = 250,
num_iterations_after_burn_in = 1000,
alpha = 0.95, beta = 2, k = 2, q = 0.9, nu = 3,
prob_rule_class = 0.5,
mh_prob_steps = c(2.5, 2.5, 4)/9,
debug_log = FALSE,
run_in_sample = TRUE,
s_sq_y = "mse",
sig_sq_est = NULL,
cov_prior_vec = NULL,
use_missing_data = FALSE,
```

```
covariates_to_permute = NULL,
num_rand_samps_in_library = 10000,
use_missing_data_dummies_as_covars = FALSE,
replace_missing_data_with_x_j_bar = FALSE,
impute_missingness_with_rf_impute = FALSE,
impute_missingness_with_x_j_bar_for_lm = TRUE,
mem_cache_for_speed = TRUE,
flush_indices_to_save_RAM = TRUE,
serialize = FALSE,
seed = NULL,
verbose = TRUE)
```

Arguments

`X` Data frame of predictors. Factors are automatically converted to dummies internally.

`y` Vector of response variable. If `y` is `numeric` or `integer`, a BART model for regression is built. If `y` is a factor with two levels, a BART model for classification is built.

`Xy` A data frame of predictors and the response. The response column must be named “y”.

`num_trees` The number of trees to be grown in the sum-of-trees model.

`num_burn_in` Number of MCMC samples to be discarded as “burn-in”.

`num_iterations_after_burn_in` Number of MCMC samples to draw from the posterior distribution of $\hat{f}(x)$.

`alpha` Base hyperparameter in tree prior for whether a node is nonterminal or not.

`beta` Power hyperparameter in tree prior for whether a node is nonterminal or not.

`k` For regression, `k` determines the prior probability that $E(Y|X)$ is contained in the interval (y_{min}, y_{max}), based on a normal distribution. For example, when $k = 2$, the prior probability is 95%. For classification, `k` determines the prior probability that $E(Y|X)$ is between $(-3, 3)$. Note that a larger value of `k` results in more shrinkage and a more conservative fit.

`q` Quantile of the prior on the error variance at which the data-based estimate is placed. Note that the larger the value of `q`, the more aggressive the fit as you are placing more prior weight on values lower than the data-based estimate. Not used for classification.

`nu` Degrees of freedom for the inverse χ^2 prior. Not used for classification.

`prob_rule_class` Threshold for classification. Any observation with a conditional probability greater than `prob_class_rule` is assigned the “positive” outcome. Note that the first level of the response is treated as the “negative” outcome and the second is treated as the “positive” outcome.

`mh_prob_steps` Vector of prior probabilities for proposing changes to the tree structures: (GROW, PRUNE, CHANGE)

`debug_log` If TRUE, additional information about the model construction are printed to a file in the working directory.

run_in_sample
: If TRUE, in-sample statistics such as $\hat{f}(x)$, Pseudo-R^2, and RMSE are computed. Setting this to FALSE when not needed can decrease computation time.

s_sq_y
: If “mse”, a data-based estimated of the error variance is computed as the MSE from ordinary least squares regression. If “var”., the data-based estimate is computed as the variance of the response. Not used in classification.

sig_sq_est
: Pass in an estimate of the maximum sig_sq of the model. This is useful to cache somewhere and then pass in during cross-validation since the default method of estimation is a linear model. In large dimensions, linear model estimation is slow.

cov_prior_vec
: Vector assigning relative weights to how often a particular variable should be proposed as a candidate for a split. The vector is internally normalized so that the weights sum to 1. Note that the length of this vector must equal the length of the design matrix after dummification and augmentation of indicators of missingness (if used). To see what the dummified matrix looks like, use `dummify_data`. See Bleich et al. (2013) for more details on when this feature is most appropriate.

use_missing_data
: If TRUE, the missing data feature is used to automatically handle missing data without imputation. See Kapelner and Bleich (2013) for details.

covariates_to_permute
: Private argument for `cov_importance_test`. Not needed by user.

num_rand_samps_in_library
: Before building a BART model, samples from the Standard Normal and $\chi^2(\nu)$ are drawn to be used in the MCMC steps. This parameter determines the number of samples to be taken.

use_missing_data_dummies_as_covars
: If TRUE, additional indicator variables for whether or not an observation in a particular column is missing are included. See Kapelner and Bleich (2013) for details.

replace_missing_data_with_x_j_bar
: If TRUE ,missing entries in `X` are imputed with average value or modal category.

impute_missingness_with_rf_impute
: If TRUE, missing entries are filled in using the rf.impute() function from the `randomForest` library.

impute_missingness_with_x_j_bar_for_lm
: If TRUE, when computing the data-based estimate of σ^2, missing entries are imputed with average value or modal category.

mem_cache_for_speed
: Speed enhancement that caches the predictors and the split values that are available at each node for selecting new rules. If the number of predictors is large, the memory requirements become large. We recommend keeping this on (default) and turning it off if you experience out-of-memory errors.

flush_indices_to_save_RAM
: Setting this flag to `TRUE` saves memory with the downside that you cannot use the functions `node_prediction_training_data_indices` nor `get_projection_weights`.

serialize
: Setting this option to TRUE will allow serialization of bartMachine objects which allows for persistence between R sessions if the object is saved and reloaded. Note that serialized objects can take up a large amount of memory. Thus, the default is FALSE.

seed
: Optional: sets the seed in both R and Java. Default is NULL which does not set the seed in R nor Java.

verbose
: Prints information about progress of the algorithm to the screen.

Value

Returns an object of class "bartMachine". The "bartMachine" object contains a list of the following components:

java_bart_machine
: A pointer to the BART Java object.

train_data_features
: The names of the variables used in the training data.

training_data_features_with_missing_features.
: The names of the variables used in the training data. If use_missing_data_dummies_as_covars = TRUE, this also includes dummies for any predictors that contain at least one missing entry (named "M_<feature>").

y
: The values of the response for the training data.

y_levels
: The levels of the response (for classification only).

pred_type
: Whether the model was build for regression of classification.

model_matrix_training_data
: The training data with factors converted to dummies.

num_cores
: The number of cores used to build the BART model.

sig_sq_est
: The data-based estimate of σ^2 used to create the prior on the error variance for the BART model.

time_to_build
: Total time to build the BART model.

y_hat_train
: The posterior means of $\hat{f}(x)$ for each observation. Only returned if run_in_sample = TRUE.

residuals
: The model residuals given by y - y_hat_train. Only returned if run_in_sample = TRUE.

L1_err_train
: L1 error on the training set. Only returned if run_in_sample = TRUE.

L2_err_train
: L2 error on the training set. Only returned if run_in_sample = TRUE.

PseudoRsq
: Calculated as 1 - SSE / SST where SSE is the sum of square errors in the training data and SST is the sample variance of the response times $n - 1$. Only returned if run_in_sample = TRUE.

rmse_train
: Root mean square error on the training set. Only returned if run_in_sample = TRUE.

Additionally, the parameters passed to the function bartMachine are also components of the list.

Note

This function is parallelized by the number of cores set by set_bart_machine_num_cores. Each core will create an independent MCMC chain of size
num_burn_in + num_iterations_after_burn_in / bart_machine_num_cores.

Author(s)

Adam Kapelner and Justin Bleich

References

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

HA Chipman, EI George, and RE McCulloch. BART: Bayesian Additive Regressive Trees. The Annals of Applied Statistics, 4(1): 266–298, 2010.

A Kapelner and J Bleich. Prediction with Missing Data via Bayesian Additive Regression Trees. Canadian Journal of Statistics, 43(2): 224-239, 2015

J Bleich, A Kapelner, ST Jensen, and EI George. Variable Selection Inference for Bayesian Additive Regression Trees. ArXiv e-prints, 2013.

See Also

bartMachineCV

Examples

```
##regression example

##generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)
summary(bart_machine)

## Not run:
##Build another BART regression model
bart_machine = bartMachine(X,y, num_trees = 200, num_burn_in = 500,
num_iterations_after_burn_in = 1000)

##Classification example

#get data and only use 2 factors
data(iris)
iris2 = iris[51:150,]
iris2$Species = factor(iris2$Species)
```

```
#build BART classification model
bart_machine = build_bart_machine(iris2[ ,1:4], iris2$Species)

##get estimated probabilities
phat = bart_machine$p_hat_train
##look at in-sample confusion matrix
bart_machine$confusion_matrix

## End(Not run)
```

`bartMachineArr` *Create an array of BART models for the same data.*

Description

If BART creates models that are variable, running many on the same dataset and averaging is a good strategy. This function is a convenience method for this procedure.

Usage

```
bartMachineArr(bart_machine, R = 10)
```

Arguments

`bart_machine`	An object of class "bartMachine".
`R`	The number of replicated BART models in the array.

Value

A `bartMachineArr` object which is just a list of the `R` bartMachine models.

Author(s)

Adam Kapelner

Examples

```
#Regression example
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
```

```
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)
bart_machine_arr = bartMachineArr(bart_machine)

#Classification example
data(iris)
iris2 = iris[51 : 150, ] #do not include the third type of flower for this example
iris2$Species = factor(iris2$Species)
bart_machine = bartMachine(iris2[ ,1:4], iris2$Species)
bart_machine_arr = bartMachineArr(bart_machine)

## End(Not run)
```

`bartMachineCV` *Build BART-CV*

Description

Builds a BART-CV model by cross-validating over a grid of hyperparameter choices.

Usage

```
bartMachineCV(X = NULL, y = NULL, Xy = NULL,
num_tree_cvs = c(50, 200), k_cvs = c(2, 3, 5),
nu_q_cvs = NULL, k_folds = 5, verbose = FALSE, ...)

build_bart_machine_cv(X = NULL, y = NULL, Xy = NULL,
num_tree_cvs = c(50, 200), k_cvs = c(2, 3, 5),
nu_q_cvs = NULL, k_folds = 5, verbose = FALSE, ...)
```

Arguments

`X`	Data frame of predictors. Factors are automatically converted to dummies interally.
`y`	Vector of response variable. If `y` is `numeric` or `integer`, a BART model for regression is built. If `y` is a factor with two levels, a BART model for classification is built.
`Xy`	A data frame of predictors and the response. The response column must be named “y”.
`num_tree_cvs`	Vector of sizes for the sum-of-trees models to cross-validate over.
`k_cvs`	Vector of choices for the hyperparameter `k` to cross-validate over.
`nu_q_cvs`	Only for regression. List of vectors containing (`nu`, `q`) ordered pair choices to cross-validate over. If `NULL`, then it defaults to the three values `list(c(3,0.9),c(3,0.99),c(10,0.75))`

k_folds	Number of folds for cross-validation
verbose	Prints information about progress of the algorithm to the screen.
...	Additional arguments to be passed to bartMachine.

Value

Returns an object of class “bartMachine” with the set of hyperparameters chosen via cross-validation. We also return a matrix “cv_stats” which contains the out-of-sample RMSE for each hyperparameter set tried and “folds” which gives the fold in which each observation fell across the k-folds.

Note

This function may require significant run-time. This function is parallelized by the number of cores set in set_bart_machine_num_cores via calling bartMachine.

Author(s)

Adam Kapelner and Justin Bleich

References

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

See Also

bartMachine

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine_cv = bartMachineCV(X, y)

#information about cross-validated model
summary(bart_machine_cv)

## End(Not run)
```

`bart_machine_get_posterior` *Get Full Posterior Distribution*

Description

Generates draws from posterior distribution of $\hat{f}(x)$ for a specified set of observations.

Usage

```
bart_machine_get_posterior(bart_machine, new_data)
```

Arguments

`bart_machine`	An object of class "bartMachine".
`new_data`	A data frame containing observations at which draws from posterior distribution of $\hat{f}(x)$ are to be obtained.

Value

Returns a list with the following components:

`y_hat`	Posterior mean estimates. For regression, the estimates have the same units as the response. For classification, the estimates are probabilities.
`new_data`	The data frame with rows at which the posterior draws are to be generated. Column names should match that of the training data.
`y_hat_posterior_samples`	The full set of posterior samples of size `num_iterations_after_burn_in` for each observation. For regression, the estimates have the same units as the response. For classification, the estimates are probabilities.

Note

This function is parallelized by the number of cores set in `set_bart_machine_num_cores`.

Author(s)

Adam Kapelner and Justin Bleich

See Also

`calc_credible_intervals`, `calc_prediction_intervals`

Examples

```
## Not run:
#Regression example

#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

#get posterior distribution
posterior = bart_machine_get_posterior(bart_machine, X)
print(posterior$y_hat)


#Classification example

#get data and only use 2 factors
data(iris)
iris2 = iris[51:150,]
iris2$Species = factor(iris2$Species)

#build BART classification model
bart_machine = bartMachine(iris2[ ,1 : 4], iris2$Species)

#get posterior distribution
posterior = bart_machine_get_posterior(bart_machine, iris2[ ,1 : 4])
print(posterior$y_hat)

## End(Not run)
```

`bart_machine_num_cores` *Get Number of Cores Used by BART*

Description

Returns number of cores used by BART

Usage

```
bart_machine_num_cores()
```

Details

Returns the number of cores currently being used by parallelized BART functions

Value

Number of cores currently being used by parallelized BART functions.

Author(s)

Adam Kapelner and Justin Bleich

See Also

`set_bart_machine_num_cores`

Examples

```
## Not run:
bart_machine_num_cores()

## End(Not run)
```

`bart_predict_for_test_data` *Predict for Test Data with Known Outcomes*

Description

Utility wrapper function for computing out-of-sample metrics for a BART model when the test set outcomes are known.

Usage

```
bart_predict_for_test_data(bart_machine, Xtest, ytest, prob_rule_class = NULL)
```

Arguments

`bart_machine`	An object of class “bartMachine”.
`Xtest`	Data frame for test data containing rows at which predictions are to be made. Colnames should match that of the training data.
`ytest`	Actual outcomes for test data.
`prob_rule_class`	Threshold for classification.

Value

For regression models, a list with the following components is returned:

- `y_hat` Predictions (as posterior means) for the test observations.
- `L1_err` L1 error for predictions.
- `L2_err` L2 error for predictions.
- `rmse` RMSE for predictions.

For classification models, a list with the following components is returned:

- `y_hat` Class predictions for the test observations.
- `p_hat` Probability estimates for the test observations.
- `confusion_matrix` A confusion matrix for the test observations.

Author(s)

Adam Kapelner and Justin Bleich

See Also

predict

Examples

```
#generate Friedman data
set.seed(11)
n  = 250
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##split into train and test
train_X = X[1 : 200, ]
test_X = X[201 : 250, ]
train_y = y[1 : 200]
test_y = y[201 : 250]

##build BART regression model
bart_machine = bartMachine(train_X, train_y)

#explore performance on test data
oos_perf = bart_predict_for_test_data(bart_machine, test_X, test_y)
print(oos_perf$rmse)
```

benchmark_datasets *benchmark_datasets*

Description

Nine diverse datasets which were used for benchmarking bartMachine's out of sample performance in the vignette for this package.

Usage

```
data(benchmark_datasets)
```

Source

See vignette for details.

calc_credible_intervals *Calculate Credible Intervals*

Description

Generates credible intervals for $\hat{f}(x)$ for a specified set of observations.

Usage

```
calc_credible_intervals(bart_machine, new_data,
ci_conf = 0.95)
```

Arguments

bart_machine	An object of class "bartMachine".
new_data	A data frame containing observations at which credible intervals for $\hat{f}(x)$ are to be computed.
ci_conf	Confidence level for the credible intervals. The default is 95%.

Details

This interval is the appropriate quantiles based on the confidence level, `ci_conf`, of the predictions for each of the Gibbs samples post-burn in.

Value

Returns a matrix of the lower and upper bounds of the credible intervals for each observation in `new_data`.

Note

This function is parallelized by the number of cores set in set_bart_machine_num_cores.

Author(s)

Adam Kapelner and Justin Bleich

See Also

calc_prediction_intervals, bart_machine_get_posterior

Examples

```
#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

#get credible interval
cred_int = calc_credible_intervals(bart_machine, X)
print(head(cred_int))
```

---

calc_prediction_intervals

Calculate Prediction Intervals

Description

Generates prediction intervals for $\hat{y}$ for a specified set of observations.

Usage

```
calc_prediction_intervals(bart_machine, new_data,
pi_conf = 0.95, num_samples_per_data_point = 1000)
```

Arguments

bart_machine	An object of class "bartMachine".
new_data	A data frame containing observations at which prediction intervals for $\hat{y}$ are to be computed.
pi_conf	Confidence level for the prediction intervals. The default is 95%.
num_samples_per_data_point	The number of samples taken from the predictive distribution. The default is 1000.

Details

Credible intervals (see `calc_credible_intervals`) are the appropriate quantiles of the prediction for each of the Gibbs samples post-burn in. Prediction intervals also make use of the noise estimate at each Gibbs sample and hence are wider. For each Gibbs sample, we record the $\hat{y}$ estimate of the response and the $\hat{\sigma^2}$ estimate of the noise variance. We then sample `normal_samples_per_gibbs_sample` times from a $N(\hat{y}, \hat{\sigma^2})$ random variable to simulate many possible disturbances for that Gibbs sample. Then, all `normal_samples_per_gibbs_sample` times the number of Gibbs sample post burn-in are collected and the appropriate quantiles are taken based on the confidence level, `pi_conf`.

Value

Returns a matrix of the lower and upper bounds of the prediction intervals for each observation in `new_data`.

Note

This function is parallelized by the number of cores set in `set_bart_machine_num_cores`.

Author(s)

Adam Kapelner and Justin Bleich

References

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

See Also

`calc_credible_intervals`, `bart_machine_get_posterior`

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

#get prediction interval
pred_int = calc_prediction_intervals(bart_machine, X)
print(head(pred_int))

## End(Not run)
```

check_bart_error_assumptions
Check BART Error Assumptions

Description

Diagnostic tools to assess whether the errors of the BART model for regression are normally distributed and homoskedastic, as assumed by the model. This function generates a normal quantile plot of the residuals with a Shapiro-Wilks p-value as well as a residual plot.

Usage

```
check_bart_error_assumptions(bart_machine, hetero_plot = "yhats")
```

Arguments

bart_machine	An object of class "bartMachine".
hetero_plot	If "yhats", the residuals are plotted against the fitted values of the response. If "ys", the residuals are plotted against the actual values of the response.

Value

None.

Author(s)

Adam Kapelner and Justin Bleich

See Also

plot_convergence_diagnostics

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 300
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

#check error diagnostics
check_bart_error_assumptions(bart_machine)

## End(Not run)
```

cov_importance_test *Importance Test for Covariate(s) of Interest*

Description

This function tests the null hypothesis H_0: These covariates of interest do not affect the response under the assumptions of the BART model.

Usage

```
cov_importance_test(bart_machine, covariates = NULL,
num_permutation_samples = 100, plot = TRUE)
```

Arguments

`bart_machine`	An object of class "bart_machine".
`covariates`	A vector of names of covariates of interest to be tested for having an effect on the response. A value of NULL indicates an omnibus test for all covariates having an effect on the response. If the name of a covariate is a factor, the entire factor will be permuted. We do not recommend entering the names of factor covariate dummies.
`num_permutation_samples`	The number of times to permute the covariates of interest and create a corresponding new BART model (see details).
`plot`	If `TRUE`, this produces a histogram of the Pseudo-Rsq's / total misclassifcation error rates from the `num_permutations` BART models created with the `covariates` permuted. The plot also illustrates the observed Pseudo-Rsq's / total misclassifcation error rate from the original training data and indicates the test's p-value.

Details

To test the importance of a covariate or a set of covariates of interest on the response, this function generates `num_permutations` BART models with the covariate(s) of interest permuted (differently each time). On each run, a measure of fit is recorded. For regression, the metric is Pseudo-Rsq; for classification, it is total misclassification error.
A p-value can then be generated as follows. For regression, the p-value is the number of permutation-sampled Pseudo-Rsq's greater than the observed Pseudo-Rsq divided by `num_permutations + 1`. For classification, the p-value is the number of permutation-sampled total misclassification errors less than the observed total misclassification error divided by `num_permutations + 1`.

Value

`permutation_samples_of_error`	A vector which records the error metric of the BART models with the covariates permuted (see details).

observed_error_estimate
: For regression, this is the Pseudo-Rsq on the original training data set. For classification, this is the observed total misclassification error on the original training data set.

pval
: The approximate p-value for this test (see details).

Note

This function is parallelized by the number of cores set in set_bart_machine_num_cores.

Author(s)

Adam Kapelner and Justin Bleich

References

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

Examples

```
## Not run:
##regression example

##generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

##now test if X[, 1] affects Y nonparametrically under the BART model assumptions
cov_importance_test(bart_machine, covariates = c(1))
## note the plot and the printed p-value


## End(Not run)
```

---

destroy_bart_machine *Destroy BART Model (deprecated — do not use!)*

Description

A deprecated function that previously was responsible for cleaning up the RAM associated with a BART model. This is now handled natively by R's garbage collection.

Usage

```
destroy_bart_machine(bart_machine)
```

Arguments

bart_machine	deprecated — do not use!

Details

Removing a "bart_machine" object from R previously did not free heap space from Java. Since BART objects can consume a large amount of RAM, it is important to remove these objects by calling this function if they are no longer needed or many BART objects are being created. This operation is now taken care of by R's garbage collection. This function is deprecated and should not be used. However, running it is harmless.

Value

None.

Author(s)

Adam Kapelner and Justin Bleich

Examples

```
##None
```

---

dummify_data *Dummify Design Matrix*

Description

Create a data frame with factors converted to dummies.

Usage

```
dummify_data(data)
```

Arguments

data	Data frame to be dummified.

Details

The column names of the dummy variables are given by the "FactorName_LevelName" and are augmented to the end of the design matrix. See the example below.

Value

Returns a data frame with factors converted to dummy indicator variables.

Note

BART handles dummification internally. This function is provided as a utility function.

Author(s)

Adam Kapelner and Justin Bleich

Examples

```
#generate data
set.seed(11)
x1 = rnorm(20)
x2 = as.factor(ifelse(x1 > 0, "A", "B"))
x3 = runif(20)
X = data.frame(x1,x2,x3)
#dummify data
X_dummified = dummify_data(X)
print(X_dummified)
```

---

get_projection_weights

Gets Training Sample Projection / Weights

Description

Returns the matrix H where yhat is approximately equal to H y where yhat is the predicted values for `new_data`. If `new_data` is unspecified, yhat will be the in-sample fits. If BART was the same as OLS, H would be an orthogonal projection matrix. Here it is a projection matrix, but clearly non-orthogonal. Unfortunately, I cannot get this function to work correctly because of three possible reasons (1) BART does not work by averaging tree predictions: it is a sum of trees model where each tree sees the residuals via backfitting (2) the prediction in each node is a bayesian posterior draw which is close to ybar of the observations contained in the node if noise is gauged to be small and (3) there are transformations of the original y variable. I believe I got close and I think I'm off by a constant multiple which is a function of the number of trees. I can use regression to estimate the constant multiple and correct for it. Turn `regression_kludge` to `TRUE` for this. Note that the weights do not add up to one here. The intuition is because due to the backfitting there is multiple counting. But I'm not entirely sure.

Usage

```
get_projection_weights(bart_machine, new_data = NULL, regression_kludge = FALSE)
```

Arguments

- `bart_machine` An object of class "bartMachine".
- `new_data` Data that you wish to investigate the training sample projection / weights. If `NULL`, the original training data is used.
- `regression_kludge` See explanation in the description. Default is `FALSE`.

Value

Returns a matrix of proportions with number of rows equal to the number of rows of `new_data` and number of columns equal to the number of rows of the original training data, n.

Examples

```
## Not run:
options(java.parameters = "-Xmx10g")
pacman::p_load(bartMachine, tidyverse)

seed = 1984
set.seed(seed)
n = 100
x = rnorm(n, 0, 1)
sigma = 0.1
y = x + rnorm(n, 0, sigma)

num_trees = 200
num_iterations_after_burn_in = 1000
bart_mod = bartMachine(data.frame(x = x), y,
flush_indices_to_save_RAM = FALSE,
num_trees = num_trees,
num_iterations_after_burn_in = num_iterations_after_burn_in,
seed = seed)
bart_mod

n_star = 100
x_star = rnorm(n_star)
y_star = as.numeric(x_star + rnorm(n_star, 0, sigma))
yhat_star_bart = predict(bart_mod, data.frame(x = x_star))

Hstar = get_projection_weights(bart_mod, data.frame(x = x_star))
rowSums(Hstar)
yhat_star_projection = as.numeric(Hstar

ggplot(data.frame(
yhat_star = yhat_star_bart,
yhat_star_projection = yhat_star_projection,
y_star = y_star)) +
  geom_point(aes(x = yhat_star_bart, y = yhat_star_projection), col = "green") +
  geom_abline(slope = 1, intercept = 0)

Hstar = get_projection_weights(bart_mod, data.frame(x = x_star), regression_kludge = TRUE)
```

```
rowSums(Hstar)
yhat_star_projection = as.numeric(Hstar

ggplot(data.frame(
yhat_star = yhat_star_bart,
yhat_star_projection = yhat_star_projection,
y_star = y_star)) +
  geom_point(aes(x = yhat_star_bart, y = yhat_star_projection), col = "green") +
  geom_abline(slope = 1, intercept = 0)


## End(Not run)
```

`get_sigsqs` *Get Posterior Error Variance Estimates*

Description

Returns the posterior estimates of the error variance from the Gibbs samples with an option to create a histogram of the posterior estimates of the error variance with a credible interval overlaid.

Usage

```
get_sigsqs(bart_machine, after_burn_in = T,
plot_hist = F, plot_CI = .95, plot_sigma = F)
```

Arguments

`bart_machine`	An object of class “bartMachine”.
`after_burn_in`	If TRUE, only the σ^2 draws after the burn-in period are returned.
`plot_hist`	If TRUE, a histogram of the posterior σ^2 draws is generated.
`plot_CI`	Confidence level for credible interval on histogram.
`plot_sigma`	If TRUE, plots σ instead of σ^2.

Value

Returns a vector of posterior σ^2 draws (with or without the burn-in samples).

Author(s)

Adam Kapelner and Justin Bleich

See Also

`get_sigsqs`

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 300
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

#get posterior sigma^2's after burn-in and plot
sigsqs = get_sigsqs(bart_machine, plot_hist = TRUE)

## End(Not run)
```

---

get_var_counts_over_chain
Get the Variable Inclusion Counts

Description

Computes the variable inclusion counts for a BART model.

Usage

```
get_var_counts_over_chain(bart_machine, type = "splits")
```

Arguments

bart_machine	An object of class “bartMachine”.
type	If “splits”, then the number of times each variable is chosen for a splitting rule is computed. If “trees”, then the number of times each variable appears in a tree is computed.

Value

Returns a matrix of counts of each predictor across all trees by Gibbs sample. Thus, the dimension is `num_interations_after_burn_in` by `p` (where `p` is the number of predictors after dummifying factors and adding missingness dummies if specified by `use_missing_data_dummies_as_covars`).

Author(s)

Adam Kapelner and Justin Bleich

See Also

get_var_props_over_chain

Examples

```
## Not run:

#generate Friedman data
set.seed(11)
n  = 200
p = 10
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y, num_trees = 20)

#get variable inclusion counts
var_counts = get_var_counts_over_chain(bart_machine)
print(var_counts)

## End(Not run)
```

---

get_var_props_over_chain *Get the Variable Inclusion Proportions*

Description

Computes the variable inclusion proportions for a BART model.

Usage

```
get_var_props_over_chain(bart_machine, type = "splits")
```

Arguments

bart_machine	An object of class “bartMachine”.
type	If “splits”, then the proportion of times each variable is chosen for a splitting rule versus all splitting rules is computed. If “trees”, then the proportion of times each variable appears in a tree versus all appearances of variables in trees is computed.

Value

Returns a vector of the variable inclusion proportions.

Author(s)

Adam Kapelner and Justin Bleich

See Also

get_var_counts_over_chain

Examples

```
#generate Friedman data
set.seed(11)
n  = 200
p = 10
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y, num_trees = 20)

#Get variable inclusion proportions
var_props = get_var_props_over_chain(bart_machine)
print(var_props)
```

---

interaction_investigator *Explore Pairwise Interactions in BART Model*

Description

Explore the pairwise interaction counts for a BART model to learn about interactions fit by the model. This function includes an option to generate a plot of the pairwise interaction counts.

Usage

```
interaction_investigator(bart_machine, plot = TRUE,
num_replicates_for_avg = 5, num_trees_bottleneck = 20,
num_var_plot = 50, cut_bottom = NULL, bottom_margin = 10)
```

Arguments

bart_machine	An object of class "bartMachine".
plot	If TRUE, a plot of the pairwise interaction counts is generated.
num_replicates_for_avg	The number of replicates of BART to be used to generate pairwise interaction inclusion counts. Averaging across multiple BART models improves stability of the estimates.

`num_trees_bottleneck`
: Number of trees to be used in the sum-of-trees model for computing pairwise interactions counts. A small number of trees should be used to force the variables to compete for entry into the model.

`num_var_plot`
: Number of variables to be shown on the plot. If "Inf," all variables are plotted (not recommended if the number of predictors is large). Default is 50.

`cut_bottom`
: A display parameter between 0 and 1 that controls where the y-axis is plotted. A value of 0 would begin the y-axis at 0; a value of 1 begins the y-axis at the minimum of the average pairwise interaction inclusion count (the smallest bar in the bar plot). Values between 0 and 1 begin the y-axis as a percentage of that minimum.

`bottom_margin`
: A display parameter that adjusts the bottom margin of the graph if labels are clipped. The scale of this parameter is the same as set with `par(mar = c(....))` in R. Higher values allow for more space if the crossed covariate names are long. Note that making this parameter too large will prevent plotting and the plot function in R will throw an error.

Details

An interaction between two variables is considered to occur whenever a path from any node of a tree to any of its terminal node contains splits using those two variables. See Kapelner and Bleich, 2013, Section 4.11.

Value

`interaction_counts_avg`
: For each of the $p \times p$ interactions, what is the average count across all `num_replicates_for_avg` BART model replicates' post burn-in Gibbs samples in all trees.

`interaction_counts_sd`
: For each of the $p \times p$ interactions, what is the average sd of the interaction counts across the `num_replicates_for_avg` BART models replicates.

Note

In the plot, the red bars correspond to the standard error of the variable inclusion proportion estimates (since multiple replicates were used).

Author(s)

Adam Kapelner and Justin Bleich

References

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

See Also

`investigate_var_importance`

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 10
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y, num_trees = 20)

#investigate interactions
interaction_investigator(bart_machine)

## End(Not run)
```

`investigate_var_importance` *Explore Variable Inclusion Proportions in BART Model*

Description

Explore the variable inclusion proportions for a BART model to learn about the relative influence of the different covariates. This function includes an option to generate a plot of the variable inclusion proportions.

Usage

```
investigate_var_importance(bart_machine, type = "splits",
plot = TRUE, num_replicates_for_avg = 5, num_trees_bottleneck = 20,
num_var_plot = Inf, bottom_margin = 10)
```

Arguments

`bart_machine` An object of class “bartMachine”.

`type` If “splits”, then the proportion of times each variable is chosen for a splitting rule is computed. If “trees”, then the proportion of times each variable appears in a tree is computed.

`plot` If TRUE, a plot of the variable inclusion proportions is generated.

`num_replicates_for_avg`
The number of replicates of BART to be used to generate variable inclusion proportions. Averaging across multiple BART models improves stability of the estimates. See Bleich et al. (2013) for more details.

num_trees_bottleneck
: Number of trees to be used in the sum-of-trees for computing the variable inclusion proportions. A small number of trees should be used to force the variables to compete for entry into the model. Chipman et al. (2010) recommend 20. See this reference for more details.

num_var_plot
: Number of variables to be shown on the plot. If "Inf", all variables are plotted.

bottom_margin
: A display parameter that adjusts the bottom margin of the graph if labels are clipped. The scale of this parameter is the same as set with `par(mar = c(....))` in R. Higher values allow for more space if the covariate names are long. Note that making this parameter too large will prevent plotting and the plot function in R will throw an error.

Details

In the plot, the red bars correspond to the standard error of the variable inclusion proportion estimates.

Value

Invisibly, returns a list with the following components:

avg_var_props
: The average variable inclusion proportions for each variable (across `num_replicates_for_avg`)

sd_var_props
: The standard deviation of the variable inclusion proportions for each variable (across `num_replicates_for_avg`)

Note

This function is parallelized by the number of cores set in `set_bart_machine_num_cores`.

Author(s)

Adam Kapelner and Justin Bleich

References

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

J Bleich, A Kapelner, ST Jensen, and EI George. Variable Selection Inference for Bayesian Additive Regression Trees. ArXiv e-prints, 2013.

HA Chipman, EI George, and RE McCulloch. BART: Bayesian Additive Regressive Trees. The Annals of Applied Statistics, 4(1): 266–298, 2010.

See Also

`interaction_investigator`

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 10
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y, num_trees = 20)

#investigate variable inclusion proportions
investigate_var_importance(bart_machine)

## End(Not run)
```

`k_fold_cv` *Estimate Out-of-sample Error with K-fold Cross validation*

Description

Builds a BART model using a specified set of arguments to `build_bart_machine` and estimates the out-of-sample performance by using k-fold cross validation.

Usage

```
k_fold_cv(X, y, k_folds = 5, folds_vec = NULL, verbose = FALSE, ...)
```

Arguments

`X`	Data frame of predictors. Factors are automatically converted to dummies interally.
`y`	Vector of response variable. If `y` is `numeric` or `integer`, a BART model for regression is built. If `y` is a factor with two levels, a BART model for classification is built.
`k_folds`	Number of folds to cross-validate over. This argument is ignored if `folds_vec` is non-null.
`folds_vec`	An integer vector of indices specifying which fold each observation belongs to.
`verbose`	Prints information about progress of the algorithm to the screen.
`...`	Additional arguments to be passed to `build_bart_machine`.

Details

For each fold, a new BART model is trained (using the same set of arguments) and its performance is evaluated on the holdout piece of that fold.

Value

For regression models, a list with the following components is returned:

- `y_hat` — Predictions for the observations computed on the fold for which the observation was omitted from the training set.
- `L1_err` — Aggregate L1 error across the folds.
- `L2_err` — Aggregate L1 error across the folds.
- `rmse` — Aggregate RMSE across the folds.
- `folds` — Vector of indices specifying which fold each observation belonged to.

For classification models, a list with the following components is returned:

- `y_hat` — Class predictions for the observations computed on the fold for which the observation was omitted from the training set.
- `p_hat` — Probability estimates for the observations computed on the fold for which the observation was omitted from the training set.
- `confusion_matrix` — Aggregate confusion matrix across the folds.
- `misclassification_error` — Total misclassification error across the folds.
- `folds` — Vector of indices specifying which fold each observation belonged to.

Note

This function is parallelized by the number of cores set in `set_bart_machine_num_cores`.

Author(s)

Adam Kapelner and Justin Bleich

See Also

`bartMachine`

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

#evaluate default BART on 5 folds
k_fold_val = k_fold_cv(X, y)
print(k_fold_val$rmse)

## End(Not run)
```

linearity_test *Test of Linearity*

Description

Test to investigate H_0 : the functional relationship between the response and the regressors is linear. We fit a linear model and then test if the residuals are a function of the regressors using the

Usage

```
linearity_test(lin_mod = NULL, X = NULL, y = NULL,
num_permutation_samples = 100, plot = TRUE, ...)
```

Arguments

lin_mod	A linear model you can pass in if you do not want to use the default which is lm(y ~ X). Default is NULL which should be used if you pass in X and y.
X	Data frame of predictors. Factors are automatically converted to dummies internally. Default is NULL which should be used if you pass in lin_mode.
y	Vector of response variable. If y is numeric or integer, a BART model for regression is built. If y is a factor with two levels, a BART model for classification is built. Default is NULL which should be used if you pass in lin_mode.
num_permutation_samples	This function relies on cov_importance_test (see documentation there for details).
plot	This function relies on cov_importance_test (see documentation there for details).
...	Additional parameters to be passed to bartMachine, the model constructed on the residuals of the linear model.

Value

permutation_samples_of_error	This function relies on cov_importance_test (see documentation there for details).
observed_error_estimate	This function relies on cov_importance_test (see documentation there for details).
pval	The approximate p-value for this test. See the documentation at cov_importance_test.

Author(s)

Adam Kapelner

See Also

cov_importance_test

Examples

```
## Not run:
##regression example

##generate Friedman data i.e. a nonlinear response model
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##now test if there is a nonlinear relationship between X1, ..., X5 and y.
linearity_test(X = X, y = y)
## note the plot and the printed p-value.. should be approx 0

#generate a linear response model
y = 1 * X[ ,1] + 3 * X[,2] + 5 * X[,3] + 7 * X[ ,4] + 9 * X[,5] + rnorm(n)
linearity_test(X = X, y = y)
## note the plot and the printed p-value.. should be > 0.05


## End(Not run)
```

---

node_prediction_training_data_indices

Gets node predictions indices of the training data for new data.

Description

This returns a binary tensor for all gibbs samples after burn-in for all trees and for all training observations.

Usage

```
node_prediction_training_data_indices(bart_machine, new_data = NULL)
```

Arguments

`bart_machine`	An object of class “bartMachine”.
`new_data`	Data that you wish to investigate the training sample weights. If `NULL`, the original training data is used.

Value

Returns a binary tensor indicating whether the prediction node contained a training datum or not. For each observation in new data, the size of this tensor is number of gibbs sample after burn-in times the number of trees times the number of training data observations. This the size of the full tensor is the number of observations in the new data times the three dimensional object just explained.

`pd_plot` *Partial Dependence Plot*

Description

Creates a partial dependence plot for a BART model for regression or classification.

Usage

```
pd_plot(bart_machine, j,
levs = c(0.05, seq(from = 0.1, to = 0.9, by = 0.1), 0.95),
lower_ci = 0.025, upper_ci = 0.975, prop_data = 1)
```

Arguments

`bart_machine`	An object of class "bartMachine".
`j`	The number or name of the column in the design matrix for which the partial dependence plot is to be created.
`levs`	Quantiles at which the partial dependence function should be evaluated. Linear extrapolation is performed between these points.
`lower_ci`	Lower limit for credible interval
`upper_ci`	Upper limit for credible interval
`prop_data`	The proportion of the training data to use. Default is 1. Use a lower proportion for speedier pd_plots. The closer to 1, the more resolution the PD plot will have; the closer to 0, the lower but faster.

Details

For regression models, the units on the y-axis are the same as the units of the response. For classification models, the units on the y-axis are probits.

Value

Invisibly, returns a list with the following components:

`x_j_quants`	Quantiles at which the partial dependence function is evaluated.
`bart_avg_predictions_by_quantile`	Posterior means for $\hat{f}(x)$ at `x_j_quants`.

Note

This function is parallelized by the number of cores set in `set_bart_machine_num_cores`.

Author(s)

Adam Kapelner and Justin Bleich

References

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

HA Chipman, EI George, and RE McCulloch. BART: Bayesian Additive Regressive Trees. The Annals of Applied Statistics, 4(1): 266–298, 2010.

Examples

```
## Not run:
#Regression example

#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

#partial dependence plot for quadratic term
pd_plot(bart_machine, "X3")


#Classification example

#get data and only use 2 factors
data(iris)
iris2 = iris[51:150,]
iris2$Species = factor(iris2$Species)

#build BART classification model
bart_machine = bartMachine(iris2[ ,1:4], iris2$Species)

#partial dependence plot
pd_plot(bart_machine, "Petal.Width")

## End(Not run)
```

`plot_convergence_diagnostics` *Plot Convergence Diagnostics*

Description

A suite of plots to assess convergence diagonstics and features of the BART model.

Usage

```
plot_convergence_diagnostics(bart_machine,
plots = c("sigsqs", "mh_acceptance", "num_nodes", "tree_depths"))
```

Arguments

`bart_machine`	An object of class “bartMachine”.
`plots`	The list of plots to be displayed. The four options are: "sigsqs", "mh_acceptance", "num_nodes", "tree_depths".

Details

The “sigsqs” option plots the posterior error variance estimates by the Gibbs sample number. This is a standard tool to assess convergence of MCMC algorithms. This option is not applicable to classification BART models.
The “mh_acceptance” option plots the proportion of Metropolis-Hastings steps accepted for each Gibbs sample (number accepted divided by number of trees).
The “num_nodes” option plots the average number of nodes across each tree in the sum-of-trees model by the Gibbs sample number (for post burn-in only). The blue line is the average number of nodes over all trees.
The “tree_depths” option plots the average tree depth across each tree in the sum-of-trees model by the Gibbs sample number (for post burn-in only). The blue line is the average number of nodes over all trees.

Value

None.

Note

The “sigsqs” plot separates the burn-in σ^2’s for the first core by post burn-in σ^2’s estimates for all cores by grey vertical lines. The “mh_acceptance” plot separates burn-in from post-burn in by a grey vertical line. Post burn-in, the different core proportions plot in different colors. The “num_nodes” plot separates different core estimates by vertical lines (post burn-in only). The ‘tree_depths” plot separates different core estimates by vertical lines (post burn-in only).

Author(s)

Adam Kapelner and Justin Bleich

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

#plot convergence diagnostics
plot_convergence_diagnostics(bart_machine)

## End(Not run)
```

---

plot_y_vs_yhat *Plot the fitted Versus Actual Response*

Description

Generates a plot actual versus fitted values and corresponding credible intervals or prediction intervals for the fitted values.

Usage

```
plot_y_vs_yhat(bart_machine, Xtest = NULL, ytest = NULL,
credible_intervals = FALSE, prediction_intervals = FALSE,
interval_confidence_level = 0.95)
```

Arguments

`bart_machine` An object of class "bartMachine".

`Xtest` Optional argument for test data. If included, BART computes fitted values at the rows of `Xtest`. Else, the fitted values from the training data are used.

`ytest` Optional argument for test data. Vector of observed values corresponding to the rows of `Xtest` to be plotted against the predictions for the rows of `Xtest`.

`credible_intervals`
If TRUE, Bayesian credible intervals are computed using the quantiles of the posterior distribution of $\hat{f}(x)$. See `calc_credible_intervals` for details.

`prediction_intervals`
If TRUE, Bayesian predictive intervals are computed using the a draw of from $\hat{f}(x)$. See `calc_prediction_intervals` for details.

`interval_confidence_level`
Desired level of confidence for credible or prediction intervals.

Value

None.

Note

This function is parallelized by the number of cores set in set_bart_machine_num_cores.

Author(s)

Adam Kapelner and Justin Bleich

See Also

bart_machine_get_posterior, calc_credible_intervals, calc_prediction_intervals

Examples

```
## Not run:
#generate linear data
set.seed(11)
n  = 500
p = 3
X = data.frame(matrix(runif(n * p), ncol = p))
y = 3*X[ ,1] + 2*X[ ,2] +X[ ,3] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

##generate plot
plot_y_vs_yhat(bart_machine)

#generate plot with prediction bands
plot_y_vs_yhat(bart_machine, prediction_intervals = TRUE)

## End(Not run)
```

---

predict.bartMachine *Make a prediction on data using a BART object*

Description

Makes a prediction on new data given a fitted BART model for regression or classification.

Usage

```
## S3 method for class 'bartMachine'
predict(object, new_data, type = "prob", prob_rule_class = NULL, verbose = TRUE, ...)
```

Arguments

- `object` An object of class “bartMachine”.
- `new_data` A data frame where each row is an observation to predict. The column names should be the same as the column names of the training data.
- `type` Only relevant if the bartMachine model is classification. The type can be “prob” which will return the estimate of $P(Y = 1)$(the “positive” class) or “class” which will return the best guess as to the class of the object, in the original label, based on if the probability estimate is greater than `prob_rule_class`. Default is “prob.”
- `prob_rule_class` The rule to determine when the class estimate is $Y = 1$ (the “positive” class) based on the probability estimate. This defaults to what was originally specified in the `bart_machine` object.
- `verbose` Prints out prediction-related messages. Currently in use only for probability predictions to let the user know which class is being predicted. Default is `TRUE`.
- `...` Parameters that are ignored.

Value

If regression, a numeric vector of `y_hat`, the best guess as to the response. If classification and `type = ``prob''`, a numeric vector of `p_hat`, the best guess as to the probability of the response class being the ”positive” class. If classification and `type = ''class''`, a character vector of the best guess of the response’s class labels.

Author(s)

Adam Kapelner and Justin Bleich

See Also

`bart_predict_for_test_data`

Examples

```
#Regression example
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

##make predictions on the training data
y_hat = predict(bart_machine, X)
```

```
#Classification example
data(iris)
iris2 = iris[51 : 150, ] #do not include the third type of flower for this example
iris2$Species = factor(iris2$Species)
bart_machine = bartMachine(iris2[ ,1:4], iris2$Species)

##make probability predictions on the training data
p_hat = predict(bart_machine, X)

##make class predictions on test data
y_hat_class = predict(bart_machine, X, type = "class")

##make class predictions on test data conservatively for ''versicolor''
y_hat_class_conservative = predict(bart_machine, X, type = "class", prob_rule_class = 0.9)

## End(Not run)
```

---

predict_bartMachineArr   *Make a prediction on data using a BART array object*

Description

Makes a prediction on new data given an array of fitted BART model for regression or classification. If BART creates models that are variable, running many and averaging is a good strategy. It is well known that the Gibbs sampler gets locked into local modes at times. This is a way to average over many chains.

Usage

```
predict_bartMachineArr(object, new_data, ...)
```

Arguments

`object`	An object of class "bartMachineArr".
`new_data`	A data frame where each row is an observation to predict. The column names should be the same as the column names of the training data.
`...`	Not supported. Note that parameters `type` and `prob_rule_class` for `predict.bartMachine` are not supported.

Value

If regression, a numeric vector of `y_hat`, the best guess as to the response. If classification and `type = ``prob''`, a numeric vector of `p_hat`, the best guess as to the probability of the response class being the "positive" class. If classification and `type = ''class''`, a character vector of the best guess of the response's class labels.

Author(s)

Adam Kapelner

See Also

predict.bartMachine

Examples

```
#Regression example
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)
bart_machine_arr = bartMachineArr(bart_machine)

##make predictions on the training data
y_hat = predict(bart_machine_arr, X)

#Classification example
data(iris)
iris2 = iris[51 : 150, ] #do not include the third type of flower for this example
iris2$Species = factor(iris2$Species)
bart_machine = bartMachine(iris2[ ,1:4], iris2$Species)
bart_machine_arr = bartMachineArr(bart_machine)

##make probability predictions on the training data
p_hat = predict_bartMachineArr(bart_machine_arr, iris2[ ,1:4])

## End(Not run)
```

---

print.bartMachine   *Summarizes information about a* bartMachine *object.*

Description

This is an alias for the summary function. See description in that section.

Usage

```
## S3 method for class 'bartMachine'
print(x, ...)
```

Arguments

x	An object of class "bartMachine".
...	Parameters that are ignored.

Value

None.

Author(s)

Adam Kapelner and Justin Bleich

Examples

```
#Regression example

#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

##print out details
print(bart_machine)

##Also, the default print works too
bart_machine
```

---

rmse_by_num_trees *Assess the Out-of-sample RMSE by Number of Trees*

Description

Assess out-of-sample RMSE of a BART model for varying numbers of trees in the sum-of-trees model.

Usage

```
rmse_by_num_trees(bart_machine, tree_list = c(5, seq(10, 50, 10), 100, 150, 200),
in_sample = FALSE, plot = TRUE, holdout_pctg = 0.3, num_replicates = 4, ...)
```

Arguments

bart_machine	An object of class "bartMachine".
tree_list	List of sizes for the sum-of-trees models.
in_sample	If TRUE, the RMSE is computed on in-sample data rather than an out-of-sample holdout.
plot	If TRUE, a plot of the RMSE by the number of trees in the ensemble is created.
holdout_pctg	Percentage of the data to be treated as an out-of-sample holdout.
num_replicates	Number of replicates to average the results over. Each replicate uses a randomly sampled holdout of the data, (which could have overlap).
...	Other arguments to be passed to the plot function.

Value

Invisibly, returns the out-of-sample average RMSEs for each tree size.

Note

Since using a large number of trees can substantially increase computation time, this plot can help assess whether a smaller ensemble size is sufficient to obtain desirable predictive performance. This function is parallelized by the number of cores set in set_bart_machine_num_cores.

Author(s)

Adam Kapelner and Justin Bleich

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 200
p = 10
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y, num_trees = 20)

#explore RMSE by number of trees
rmse_by_num_trees(bart_machine)

## End(Not run)
```

set_bart_machine_num_cores *Set the Number of Cores for BART*

Description

Sets the number of cores to be used for all parallelized BART functions.

Usage

```
set_bart_machine_num_cores(num_cores)
```

Arguments

num_cores	Number of cores to use

Value

None.

Author(s)

Adam Kapelner and Justin Bleich

See Also

bart_machine_num_cores

Examples

```
## Not run:
## set all parallelized functions to use 4 cores
## set_bart_machine_num_cores(4)

## End(Not run)
```

summary.bartMachine *Summarizes information about a* bartMachine *object.*

Description

Provides a quick summary of the BART model.

Usage

```
## S3 method for class 'bartMachine'
summary(object, ...)
```

Arguments

`object`	An object of class "bartMachine".
`...`	Parameters that are ignored.

Details

Gives the version number of the `bartMachine` package used to build this `bartMachine` object and if the object models either "regression" or "classification." Gives the amount of training data and the dimension of feature space. Prints the amount of time it took to build the model, how many processor cores were used to during its construction, as well as the number of burn-in and posterior Gibbs samples were used.

If the model is for regression, it prints the estimate of σ^2 before the model was constructed as well as after so the user can inspect how much variance was explained.

If the model was built using the `run_in_sample = TRUE` parameter in `build_bart_machine` and is for regression, the summary L1, L2, rmse, Pseudo-R^2 are printed as well as the p-value for the tests of normality and zero-mean noise. If the model is for classification, a confusion matrix is printed.

Value

None.

Author(s)

Adam Kapelner and Justin Bleich

Examples

```
#Regression example

#generate Friedman data
set.seed(11)
n  = 200
p = 5
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model
bart_machine = bartMachine(X, y)

##print out details
summary(bart_machine)

##Also, the default print works too
bart_machine
```

---

var_selection_by_permute
Perform Variable Selection using Three Threshold-based Procedures

---

Description

Performs variable selection using the three thresholding methods introduced in Bleich et al. (2013).

Usage

```
var_selection_by_permute(bart_machine,
num_reps_for_avg = 10, num_permute_samples = 100,
num_trees_for_permute = 20, alpha = 0.05,
plot = TRUE, num_var_plot = Inf, bottom_margin = 10)
```

Arguments

- `bart_machine` An object of class “bartMachine”.
- `num_reps_for_avg` Number of replicates to over over to for the BART model’s variable inclusion proportions.
- `num_permute_samples` Number of permutations of the response to be made to generate the “null” permutation distribution.
- `num_trees_for_permute` Number of trees to use in the variable selection procedure. As with `investigate_var_importance`, a small number of trees should be used to force variables to compete for entry into the model. Note that this number is used to estimate both the “true” and “null” variable inclusion proportions.
- `alpha` Cut-off level for the thresholds.
- `plot` If TRUE, a plot showing which variables are selected by each of the procedures is generated.
- `num_var_plot` Number of variables (in order of decreasing variable inclusion proportion) to be plotted.
- `bottom_margin` A display parameter that adjusts the bottom margin of the graph if labels are clipped. The scale of this parameter is the same as set with `par(mar = c(....))` in R. Higher values allow for more space if the crossed covariate names are long. Note that making this parameter too large will prevent plotting and the plot function in R will throw an error.

Details

See Bleich et al. (2013) for a complete description of the procedures outlined above as well as the corresponding vignette for a brief summary with examples.

Value

Invisibly, returns a list with the following components:

`important_vars_local_names`
: Names of the variables chosen by the Local procedure.

`important_vars_global_max_names`
: Names of the variables chosen by the Global Max procedure.

`important_vars_global_se_names`
: Names of the variables chosen by the Global SE procedure.

`important_vars_local_col_nums`
: Column numbers of the variables chosen by the Local procedure.

`important_vars_global_max_col_nums`
: Column numbers of the variables chosen by the Global Max procedure.

`important_vars_global_se_col_nums`
: Column numbers of the variables chosen by the Global SE procedure.

`var_true_props_avg`
: The variable inclusion proportions for the actual data.

`permute_mat`
: The permutation distribution generated by permuting the response vector.

Note

Although the reference only explores regression settings, this procedure is applicable to both regression and classification problems. This function is parallelized by the number of cores set in `set_bart_machine_num_cores`.

Author(s)

Adam Kapelner and Justin Bleich

References

J Bleich, A Kapelner, ST Jensen, and EI George. Variable Selection Inference for Bayesian Additive Regression Trees. ArXiv e-prints, 2013.

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

See Also

`var_selection_by_permute`, `investigate_var_importance`

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 300
p = 20 ##15 useless predictors
X = data.frame(matrix(runif(n * p), ncol = p))
```

```
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)

##build BART regression model (not actuall used in variable selection)
bart_machine = bartMachine(X, y)

#variable selection
var_sel = var_selection_by_permute(bart_machine)
print(var_sel$important_vars_local_names)
print(var_sel$important_vars_global_max_names)

## End(Not run)
```

`var_selection_by_permute_cv`
Perform Variable Selection Using Cross-validation Procedure

Description

Performs variable selection by cross-validating over the three threshold-based procedures outlined in Bleich et al. (2013) and selecting the single procedure that returns the lowest cross-validation RMSE.

Usage

```
var_selection_by_permute_cv(bart_machine, k_folds = 5,
num_reps_for_avg = 5, num_permute_samples = 100,
num_trees_for_permute = 20, alpha = 0.05, num_trees_pred_cv = 50)
```

Arguments

`bart_machine` An object of class "bartMachine".

`k_folds` Number of folds to be used in cross-validation.

`num_reps_for_avg`
Number of replicates to over over to for the BART model's variable inclusion proportions.

`num_permute_samples`
Number of permutations of the response to be made to generate the "null" permutation distribution.

`num_trees_for_permute`
Number of trees to use in the variable selection procedure. As with `investigate_var_importance`, a small number of trees should be used to force variables to compete for entry into the model. Note that this number is used to estimate both the "true" and "null" variable inclusion proportions.

`alpha` Cut-off level for the thresholds.

num_trees_pred_cv
: Number of trees to use for prediction on the hold-out portion of each fold. Once variables have been selected using the training portion of each fold, a new model is built using only those variables with num_trees_pred_cv trees in the sum-of-trees model. Forecasts for the holdout sample are made using this model. A larger number of trees is recommended to exploit the full forecasting power of BART.

Details

See Bleich et al. (2013) for a complete description of the procedures outlined above as well as the corresponding vignette for a brief summary with examples.

Value

Returns a list with the following components:

best_method
: The name of the best variable selection procedure, as chosen via cross-validation.

important_vars_cv
: The variables chosen by the best_method above.

Note

This function can have substantial run-time. This function is parallelized by the number of cores set in set_bart_machine_num_cores.

Author(s)

Adam Kapelner and Justin Bleich

References

J Bleich, A Kapelner, ST Jensen, and EI George. Variable Selection Inference for Bayesian Additive Regression Trees. ArXiv e-prints, 2013.

Adam Kapelner, Justin Bleich (2016). bartMachine: Machine Learning with Bayesian Additive Regression Trees. Journal of Statistical Software, 70(4), 1-40. doi:10.18637/jss.v070.i04

See Also

var_selection_by_permute, investigate_var_importance

Examples

```
## Not run:
#generate Friedman data
set.seed(11)
n  = 150
p = 100 ##95 useless predictors
X = data.frame(matrix(runif(n * p), ncol = p))
y = 10 * sin(pi* X[ ,1] * X[,2]) +20 * (X[,3] -.5)^2 + 10 * X[ ,4] + 5 * X[,5] + rnorm(n)
```

```
##build BART regression model (not actually used in variable selection)
bart_machine = bartMachine(X, y)

#variable selection via cross-validation
var_sel_cv = var_selection_by_permute_cv(bart_machine, k_folds = 3)
print(var_sel_cv$best_method)
print(var_sel_cv$important_vars_cv)

## End(Not run)
```

Index