Package 'vip'

October 12, 2022

Type Package

Title Variable Importance Plots

Version 0.3.2

Description A general framework for constructing variable importance plots from various types of machine learning models in R. Aside from some standard model-specific variable importance measures, this package also provides model-agnostic approaches that can be applied to any supervised learning algorithm. These include 1) an efficient permutation-based variable importance measure, 2) variable importance based on Shapley values (Strumbelj and Kononenko, 2014) <doi:10.1007/s10115-013-0679-x>, and 3) the variance-based approach described in Greenwell et al. (2018) <arXiv:1805.04755>. A variance-based method for quantifying the relative strength of interaction effects is also included (see the previous reference for details).

License GPL (>= 2)

URL https://github.com/koalaverse/vip/

BugReports https://github.com/koalaverse/vip/issues

Encoding UTF-8

VignetteBuilder knitr

LazyData true

Imports ggplot2 (>= 0.9.0), gridExtra, magrittr, plyr, stats, tibble, utils

Suggests DT, C50, caret, Ckmeans.1d.dp, covr, Cubist, doParallel, dplyr, earth, fastshap, gbm, glmnet, h2o, htmlwidgets, keras, knitr, lattice, mlbench, mlr, mlr3, neuralnet, NeuralNetTools, nnet, parsnip, party, partykit, pdp, pls, randomForest, ranger, rmarkdown, rpart, RSNNS, sparkline, sparklyr (>= 0.8.0), tinytest, varImp, xgboost

RoxygenNote 7.1.1

NeedsCompilation no

Author Brandon Greenwell [aut, cre] (<https://orcid.org/0000-0002-8120-0084>), Brad Boehmke [aut] (<https://orcid.org/0000-0002-3611-8516>), Bernie Gray [aut] (<https://orcid.org/0000-0001-9190-6032>)

Maintainer Brandon Greenwell <greenwell.brandon@gmail.com>

Repository CRAN

Date/Publication 2020-12-17 16:20:02 UTC

R topics documented:

`add_sparklines` *Add sparklines*

Description

Create an HTML widget to display variable importance scores with a sparkline representation of each features effect (i.e., its partial dependence function).

Usage

```
add_sparklines(object, fit, digits = 3, free_y = FALSE, verbose = FALSE, ...)

## S3 method for class 'vi'
add_sparklines(object, fit, digits = 3, free_y = FALSE, verbose = FALSE, ...)
```

Arguments

`object`	An object that inherits from class `"vi"`.
`fit`	The original fitted model. Only needed if ‘vi()‘ was not called with ‘method = "firm"‘.
`digits`	Integer specifying the minimal number of significant digits to use for displaying importance scores and, if available, their standard deviations.
`free_y`	Logical indicating whether or not the the y-axis limits should be allowed to vary for each sparkline. Default is `FALSE`.

verbose	Logical indicating whether or not to print progress. Default is FALSE.
...	Additional optional arguments to be passed on to partial.

Value

An object of class c("datatables", "htmlwidget"); essentially, a data frame with three columns: Variable, Importance, and Effect (a sparkline representation of the partial dependence function). For "lm"/"glm"-like objects, an additional column, called Sign, is also included which includes the sign (i.e., POS/NEG) of the original coefficient.

References

Greenwell, B. M., Boehmke, B. C., and McCarthy, A. J. A Simple and Effective Model-Based Variable Importance Measure. arXiv preprint arXiv:1805.04755 (2018).

---

gen_friedman *Friedman benchmark data*

Description

Simulate data from the Friedman 1 benchmark problem. See mlbench.friedman1 for details and references.

Usage

```
gen_friedman(
  n_samples = 100,
  n_features = 10,
  n_bins = NULL,
  sigma = 0.1,
  seed = NULL
)
```

Arguments

n_samples	Integer specifying the number of samples (i.e., rows) to generate. Default is 100.
n_features	Integer specifying the number of features to generate. Default is 10.
n_bins	Integer specifying the number of (roughly) equal sized bins to split the response into. Default is NULL for no binning. Setting to a positive integer > 1 effectively turns this into a classification problem where n_bins gives the number of classes.
sigma	Numeric specifying the standard deviation of the noise.
seed	Integer specifying the random seed. If NULL (the default) the results will be different each time the function is run.

Note

This function is mostly used for internal testing.

Examples

```
gen_friedman()
```

---

get_formula *Extract model formula*

Description

Calls `formula` to extract the formulae from various modeling objects, but returns `NULL` instead of an error for objects that do not contain a formula component.

Usage

```
get_formula(object)
```

Arguments

`object`	An appropriate fitted model object.

Value

Either a `formula` object or `NULL`.

---

list_metrics *List metrics*

Description

List all available performance metrics.

Usage

```
list_metrics()
```

Examples

```
(metrics <- list_metrics())
metrics[metrics$Task == "Multiclass classification", ]
```

metric_mse *Model metrics*

Description

Common model/evaluation metrics for machine learning.

Usage

```
metric_mse(actual, predicted, na.rm = FALSE)

metric_rmse(actual, predicted, na.rm = FALSE)

metric_sse(actual, predicted, na.rm = FALSE)

metric_mae(actual, predicted, na.rm = FALSE)

metric_rsquared(actual, predicted, na.rm = FALSE)

metric_accuracy(actual, predicted, na.rm = FALSE)

metric_error(actual, predicted, na.rm = FALSE)

metric_auc(actual, predicted)

metric_logLoss(actual, predicted)

metric_mauc(actual, predicted)
```

Arguments

actual	Vector of actual target values.
predicted	Vector of predicted target values.
na.rm	Logical indicating whether or not NA values should be stripped before the computation proceeds.

Note

The metric_auc and metric_logLoss functions are based on code from the Metrics package.

Examples

```
x <- rnorm(10)
y <- rnorm(10)
metric_mse(x, y)
metric_rsquared(x, y)
```

vi *Variable importance*

Description

Compute variable importance scores for the predictors in a model.

Usage

```
vi(object, ...)

## Default S3 method:
vi(
  object,
  method = c("model", "firm", "permute", "shap"),
  feature_names = NULL,
  FUN = NULL,
  var_fun = NULL,
  ice = FALSE,
  abbreviate_feature_names = NULL,
  sort = TRUE,
  decreasing = TRUE,
  scale = FALSE,
  rank = FALSE,
  ...
)

## S3 method for class 'model_fit'
vi(object, ...)

## S3 method for class 'WrappedModel'
vi(object, ...)

## S3 method for class 'Learner'
vi(object, ...)
```

Arguments

object	A fitted model object (e.g., a "randomForest" object) or an object that inherits from class "vi".
...	Additional optional arguments to be passed on to vi_model, vi_firm, vi_permute, or vi_shap.
method	Character string specifying the type of variable importance (VI) to compute. Current options are "model" (the default), for model-specific VI scores (see vi_model for details), "firm", for variance-based VI scores (see vi_firm for

details), "permute", for permutation-based VI scores (see ' vi_permute for details), or "shap", for Shapley-based VI scores. For more details on the variance-based methods, see Greenwell et al. (2018) and Scholbeck et al. (2019).

feature_names — Character string giving the names of the predictor variables (i.e., features) of interest.

FUN — Deprecated. Use var_fun instead.

var_fun — List with two components, "cat" and "con", containing the functions to use to quantify the variability of the feature effects (e.g., partial dependence values) for categorical and continuous features, respectively. If NULL, the standard deviation is used for continuous features. For categorical features, the range statistic is used (i.e., (max - min) / 4). Only applies when method = "firm".

ice — Logical indicating whether or not to estimate feature effects using *individual conditional expectation* (ICE) curves. Only applies when method = "firm". Default is FALSE. Setting ice = TRUE is preferred whenever strong interaction effects are potentially present.

abbreviate_feature_names — Integer specifying the length at which to abbreviate feature names. Default is NULL which results in no abbreviation (i.e., the full name of each feature will be printed).

sort — Logical indicating whether or not to order the sort the variable importance scores. Default is TRUE.

decreasing — Logical indicating whether or not the variable importance scores should be sorted in descending (TRUE) or ascending (FALSE) order of importance. Default is TRUE.

scale — Logical indicating whether or not to scale the variable importance scores so that the largest is 100. Default is FALSE.

rank — Logical indicating whether or not to rank the variable importance scores (i.e., convert to integer ranks). Default is FALSE. Potentially useful when comparing variable importance scores across different models using different methods.

Value

A tidy data frame (i.e., a "tibble" object) with at least two columns: Variable and Importance. For "lm"/"glm"-like objects, an additional column, called Sign, is also included which includes the sign (i.e., POS/NEG) of the original coefficient. If method = "permute" and nsim > 1, then an additional column, StDev, giving the standard deviation of the permutation-based variable importance scores is included.

References

Greenwell, B. M., Boehmke, B. C., and McCarthy, A. J. A Simple and Effective Model-Based Variable Importance Measure. arXiv preprint arXiv:1805.04755 (2018).

Examples

```
#
# A projection pursuit regression example
```

```
#

# Load the sample data
data(mtcars)

# Fit a projection pursuit regression model
mtcars.ppr <- ppr(mpg ~ ., data = mtcars, nterms = 1)

# Compute variable importance scores
vi(mtcars.ppr, method = "firm", ice = TRUE)
vi(mtcars.ppr, method = "firm", ice = TRUE,
   var_fun = list("con" = mad, "cat" = function(x) diff(range(x)) / 4))

# Plot variable importance scores
vip(mtcars.ppr, method = "firm", ice = TRUE)
```

---

vint *Interaction effects*

Description

Quantify the strength of two-way interaction effects using a simple *feature importance ranking measure* (FIRM) approach. For details, see Greenwell et al. (2018).

Usage

```
vint(
  object,
  feature_names,
  progress = "none",
  parallel = FALSE,
  paropts = NULL,
  ...
)
```

Arguments

object	A fitted model object (e.g., a "randomForest" object).
feature_names	Character string giving the names of the two features of interest.
progress	Character string giving the name of the progress bar to use while constructing the interaction statistics. See create_progress_bar for details. Default is "none".
parallel	Logical indicating whether or not to run partial in parallel using a backend provided by the foreach package. Default is FALSE.
paropts	List containing additional options to be passed on to foreach when parallel = TRUE.
...	Additional optional arguments to be passed on to partial.

Details

This function quantifies the strength of interaction between features X_1 and X_2 by measuring the change in variance along slices of the partial dependence of X_1 and X_2 on the target Y. See Greenwell et al. (2018) for details and examples.

References

Greenwell, B. M., Boehmke, B. C., and McCarthy, A. J.: A Simple and Effective Model-Based Variable Importance Measure. arXiv preprint arXiv:1805.04755 (2018).

Examples

```
## Not run:
#
# The Friedman 1 benchmark problem
#

# Load required packages
library(gbm)
library(ggplot2)
library(mlbench)

# Simulate training data
trn <- gen_friedman(500, seed = 101)  # ?vip::gen_friedman

#
# NOTE: The only interaction that actually occurs in the model from which
# these data are generated is between x.1 and x.2!
#

# Fit a GBM to the training data
set.seed(102)  # for reproducibility
fit <- gbm(y ~ ., data = trn, distribution = "gaussian", n.trees = 1000,
           interaction.depth = 2, shrinkage = 0.01, bag.fraction = 0.8,
           cv.folds = 5)
best_iter <- gbm.perf(fit, plot.it = FALSE, method = "cv")

# Quantify relative interaction strength
all_pairs <- combn(paste0("x.", 1:10), m = 2)
res <- NULL
for (i in seq_along(all_pairs)) {
  interact <- vint(fit, feature_names = all_pairs[, i], n.trees = best_iter)
  res <- rbind(res, interact)
}

# Plot top 20 results
top_20 <- res[1L:20L, ]
ggplot(top_20, aes(x = reorder(Variables, Interaction), y = Interaction)) +
  geom_col() +
  coord_flip() +
  xlab("") +
  ylab("Interaction strength")
```

```
## End(Not run)
```

---

vip *Variable importance plots*

Description

Plot variable importance scores for the predictors in a model.

Usage

```
vip(object, ...)

## Default S3 method:
vip(
  object,
  num_features = 10L,
  geom = c("col", "point", "boxplot", "violin"),
  mapping = NULL,
  aesthetics = list(),
  horizontal = TRUE,
  all_permutations = FALSE,
  jitter = FALSE,
  include_type = FALSE,
  ...
)

## S3 method for class 'model_fit'
vip(object, ...)
```

Arguments

`object`	A fitted model object (e.g., a `"randomForest"` object) or an object that inherits from class `"vi"`.
`...`	Additional optional arguments to be passed on to `vi`.
`num_features`	Integer specifying the number of variable importance scores to plot. Default is `10`.
`geom`	Character string specifying which type of plot to construct. The currently available options are described below.

- `geom = "col"` uses `geom_col` to construct a bar chart of the variable importance scores.
- `geom = "point"` uses `geom_point` to construct a Cleveland dot plot of the variable importance scores.

- geom = "boxplot" uses geom_boxplot to construct a boxplot plot of the variable importance scores. This option can only for the permutation-based importance method with nsim > 1 and keep = TRUE; see vi_permute for details.
- geom = "violin" uses geom_violin to construct a violin plot of the variable importance scores. This option can only for the permutation-based importance method with nsim > 1 and keep = TRUE; see vi_permute for details.

mapping — Set of aesthetic mappings created by aes or aes_. See example usage below.

aesthetics — List specifying additional arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. See example usage below.

horizontal — Logical indicating whether or not to plot the importance scores on the x-axis (TRUE). Default is TRUE.

all_permutations — Logical indicating whether or not to plot all permutation scores along with the average. Default is FALSE. (Only used for permutation scores when nsim > 1.)

jitter — Logical indicating whether or not to jitter the raw permutation scores. Default is FALSE. (Only used when all_permutations = TRUE.)

include_type — Logical indicating whether or not to include the type of variable importance computed in the axis label. Default is FALSE.

Examples

```
#
# A projection pursuit regression example
#

# Load the sample data
data(mtcars)

# Fit a projection pursuit regression model
model <- ppr(mpg ~ ., data = mtcars, nterms = 1)

# Construct variable importance plot
vip(model, method = "firm")

# Better yet, store the variable importance scores and then plot
vi_scores <- vi(model, method = "firm")
vip(vi_scores, geom = "point", horiz = FALSE)
vip(vi_scores, geom = "point", horiz = FALSE, aesthetics = list(size = 3))

# The `%T>\%` operator is imported for convenience; see ?magrittr::`%T>%`
# for details
vi_scores <- model %>%
  vi(method = "firm") %T>%
  {print(vip(.))}
vi_scores
```

```
# Permutation scores (barplot w/ raw values and jittering)
pfun <- function(object, newdata) predict(object, newdata = newdata)
vip(model, method = "permute", train = mtcars, target = "mpg", nsim = 10,
    metric = "rmse", pred_wrapper = pfun,
    aesthetics = list(color = "grey50", fill = "grey50"),
    all_permutations = TRUE, jitter = TRUE)

# Permutation scores (boxplot)
vip(model, method = "permute", train = mtcars, target = "mpg", nsim = 10,
    metric = "rmse", pred_wrapper = pfun, geom = "boxplot")

# Permutation scores (boxplot colored by feature)
library(ggplot2)  # for `aes_string()` function
vip(model, method = "permute", train = mtcars, target = "mpg", nsim = 10,
    metric = "rmse", pred_wrapper = pfun, geom = "boxplot",
    all_permutations = TRUE, mapping = aes_string(fill = "Variable"),
    aesthetics = list(color = "grey35", size = 0.8))
```

`vi_firm` *Variance-based variable importance*

Description

Compute variance-based variable importance using a simple *feature importance ranking measure* (FIRM) approach; for details, see Greenwell et al. (2018) and Scholbeck et al. (2019).

Usage

```
vi_firm(object, ...)

## Default S3 method:
vi_firm(object, feature_names, FUN = NULL, var_fun = NULL, ice = FALSE, ...)
```

Arguments

`object`	A fitted model object (e.g., a `"randomForest"` object).
`...`	Additional optional arguments to be passed on to `partial`.
`feature_names`	Character string giving the names of the predictor variables (i.e., features) of interest.
`FUN`	Deprecated. Use `var_fun` instead.
`var_fun`	List with two components, `"cat"` and `"con"`, containing the functions to use to quantify the variability of the feature effects (e.g., partial dependence values) for categorical and continuous features, respectively. If `NULL`, the standard deviation is used for continuous features. For categorical features, the range statistic is used (i.e., (max - min) / 4). Only applies when `method = "firm"`.
`ice`	Logical indicating whether or not to estimate feature effects using *individual conditional expectation* (ICE) curves. Only applies when `method = "firm"`. Default is `FALSE`. Setting `ice = TRUE` is preferred whenever strong interaction effects are potentially present.

Details

This approach to computing VI scores is based on quantifying the relative "flatness" of the effect of each feature. Feature effects can be assessed using *partial dependence plots* (PDPs) or *individual conditional expectation* (ICE) curves. These approaches are model-agnostic and can be applied to any supervised learning algorithm. By default, relative "flatness" is defined by computing the standard deviation of the y-axis values for each feature effect plot for numeric features; for categorical features, the default is to use range divided by 4. This can be changed via the ‘var_fun‘ argument. See Greenwell et al. (2018) for details and additional examples.

Value

A tidy data frame (i.e., a `"tibble"` object) with two columns, `Variable` and `Importance`, containing the variable name and its associated importance score, respectively.

References

Greenwell, B. M., Boehmke, B. C., and McCarthy, A. J. A Simple and Effective Model-Based Variable Importance Measure. arXiv preprint arXiv:1805.04755 (2018).

Scholbeck, C. A. Scholbeck, and Molnar, C., and Heumann C., and Bischl, B., and Casalicchio, G. Sampling, Intervention, Prediction, Aggregation: A Generalized Framework for Model-Agnostic Interpretations. arXiv preprint arXiv:1904.03959 (2019).

`vi_model` *Model-specific variable importance*

Description

Compute model-specific variable importance scores for the predictors in a model.

Usage

```
vi_model(object, ...)

## Default S3 method:
vi_model(object, ...)

## S3 method for class 'C5.0'
vi_model(object, type = c("usage", "splits"), ...)

## S3 method for class 'train'
vi_model(object, ...)

## S3 method for class 'cubist'
vi_model(object, ...)

## S3 method for class 'earth'
```

```
vi_model(object, type = c("nsubsets", "rss", "gcv"), ...)

## S3 method for class 'gbm'
vi_model(object, type = c("relative.influence", "permutation"), ...)

## S3 method for class 'glmnet'
vi_model(object, lambda = NULL, ...)

## S3 method for class 'cv.glmnet'
vi_model(object, lambda = NULL, ...)

## S3 method for class 'H2OBinomialModel'
vi_model(object, ...)

## S3 method for class 'H2OMultinomialModel'
vi_model(object, ...)

## S3 method for class 'H2ORegressionModel'
vi_model(object, ...)

## S3 method for class 'WrappedModel'
vi_model(object, ...)

## S3 method for class 'Learner'
vi_model(object, ...)

## S3 method for class 'nn'
vi_model(object, type = c("olden", "garson"), ...)

## S3 method for class 'nnet'
vi_model(object, type = c("olden", "garson"), ...)

## S3 method for class 'model_fit'
vi_model(object, ...)

## S3 method for class 'RandomForest'
vi_model(object, type = c("accuracy", "auc"), ...)

## S3 method for class 'constparty'
vi_model(object, ...)

## S3 method for class 'cforest'
vi_model(object, ...)

## S3 method for class 'mvr'
vi_model(object, ...)

## S3 method for class 'randomForest'
```

```
vi_model(object, ...)

## S3 method for class 'ranger'
vi_model(object, ...)

## S3 method for class 'rpart'
vi_model(object, ...)

## S3 method for class 'mlp'
vi_model(object, type = c("olden", "garson"), ...)

## S3 method for class 'ml_model_decision_tree_regression'
vi_model(object, ...)

## S3 method for class 'ml_model_decision_tree_classification'
vi_model(object, ...)

## S3 method for class 'ml_model_gbt_regression'
vi_model(object, ...)

## S3 method for class 'ml_model_gbt_classification'
vi_model(object, ...)

## S3 method for class 'ml_model_generalized_linear_regression'
vi_model(object, ...)

## S3 method for class 'ml_model_linear_regression'
vi_model(object, ...)

## S3 method for class 'ml_model_random_forest_regression'
vi_model(object, ...)

## S3 method for class 'ml_model_random_forest_classification'
vi_model(object, ...)

## S3 method for class 'lm'
vi_model(object, type = c("stat", "raw"), ...)

## S3 method for class 'xgb.Booster'
vi_model(object, type = c("gain", "cover", "frequency"), ...)
```

Arguments

object	A fitted model object (e.g., a "randomForest" object).
...	Additional optional arguments to be passed on to other methods.
type	Character string specifying the type of variable importance to return (only used for some models). See details for which methods this argument applies to.
lambda	Numeric value for the penalty parameter of a glmnet model (this is equivalent

to the s argument in coef.glmnet). See the section on glmnet in the details below.

Details

Computes model-specific variable importance scores depending on the class of object:

C5.0 Variable importance is measured by determining the percentage of training set samples that fall into all the terminal nodes after the split. For example, the predictor in the first split automatically has an importance measurement of 100 percent since all samples are affected by this split. Other predictors may be used frequently in splits, but if the terminal nodes cover only a handful of training set samples, the importance scores may be close to zero. The same strategy is applied to rule-based models and boosted versions of the model. The underlying function can also return the number of times each predictor was involved in a split by using the option metric = "usage". See C5imp for details.

cubist The Cubist output contains variable usage statistics. It gives the percentage of times where each variable was used in a condition and/or a linear model. Note that this output will probably be inconsistent with the rules shown in the output from summary.cubist. At each split of the tree, Cubist saves a linear model (after feature selection) that is allowed to have terms for each variable used in the current split or any split above it. Quinlan (1992) discusses a smoothing algorithm where each model prediction is a linear combination of the parent and child model along the tree. As such, the final prediction is a function of all the linear models from the initial node to the terminal node. The percentages shown in the Cubist output reflects all the models involved in prediction (as opposed to the terminal models shown in the output). The variable importance used here is a linear combination of the usage in the rule conditions and the model. See summary.cubist and varImp.cubist for details.

glmnet Similar to (generalized) linear models, the absolute value of the coefficients are returned for a specific model. It is important that the features (and hence, the estimated coefficients) be standardized prior to fitting the model. You can specify which coefficients to return by passing the specific value of the penalty parameter via the lambda argument (this is equivalent to the s argument in coef.glmnet). By default, lambda = NULL and the coefficients corresponding to the final penalty value in the sequence are returned; in other words, you should ALWAYS SPECIFY lambda! For "cv.glmnet" objects, the largest value of lambda such that the error is within one standard error of the minimum is used by default. For "multnet" objects, the coefficients corresponding to the first class are used; that is, the first component of coef.glmnet.

cforest Variable importance is measured in a way similar to those computed by importance. Besides the standard version, a conditional version is available that adjusts for correlations between predictor variables. If conditional = TRUE, the importance of each variable is computed by permuting within a grid defined by the predictors that are associated (with 1 - p-value greater than threshold) to the variable of interest. The resulting variable importance score is conditional in the sense of beta coefficients in regression models, but represents the effect of a variable in both main effects and interactions. See Strobl et al. (2008) for details. Note, however, that all random forest results are subject to random variation. Thus, before interpreting the importance ranking, check whether the same ranking is achieved with a different random seed - or otherwise increase the number of trees ntree in ctree_control. Note that in the presence of missings in the predictor variables the procedure described in Hapfelmeier et al. (2012) is performed. See varimp for details.

`earth` The `earth` package uses three criteria for estimating the variable importance in a MARS model (see `evimp` for details):

- The `nsubsets` criterion (`type = "nsubsets"`) counts the number of model subsets that include each feature. Variables that are included in more subsets are considered more important. This is the criterion used by `summary.earth` to print variable importance. By "subsets" we mean the subsets of terms generated by `earth()`'s backward pass. There is one subset for each model size (from one to the size of the selected model) and the subset is the best set of terms for that model size. (These subsets are specified in the `$prune.terms` component of `earth()`'s return value.) Only subsets that are smaller than or equal in size to the final model are used for estimating variable importance. This is the default method used by **vip**.
- The `rss` criterion (`type = "rss"`) first calculates the decrease in the RSS for each subset relative to the previous subset during `earth()`'s backward pass. (For multiple response models, RSS's are calculated over all responses.) Then for each variable it sums these decreases over all subsets that include the variable. Finally, for ease of interpretation the summed decreases are scaled so the largest summed decrease is 100. Variables which cause larger net decreases in the RSS are considered more important.
- The `gcv` criterion (`type = "gcv"`) is similar to the `rss` approach, but uses the GCV statistic instead of the RSS. Note that adding a variable can sometimes increase the GCV. (Adding the variable has a deleterious effect on the model, as measured in terms of its estimated predictive power on unseen data.) If that happens often enough, the variable can have a negative total importance, and thus appear less important than unused variables.

`gbm` Variable importance is computed using one of two approaches (See `summary.gbm` for details):

- The standard approach (`type = "relative.influence"`) described in Friedman (2001). When `distribution = "gaussian"` this returns the reduction of squared error attributable to each variable. For other loss functions this returns the reduction attributable to each variable in sum of squared error in predicting the gradient on each iteration. It describes the *relative influence* of each variable in reducing the loss function. This is the default method used by **vip**.
- An experimental permutation-based approach (`type = "permutation"`). This method randomly permutes each predictor variable at a time and computes the associated reduction in predictive performance. This is similar to the variable importance measures Leo Breiman uses for random forests, but **gbm** currently computes using the entire training dataset (not the out-of-bag observations).

`H2OModel` See `h2o.varimp` or visit `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/variable-importance.html` for details.

`nnet` Two popular methods for constructing variable importance scores with neural networks are the Garson algorithm (Garson 1991), later modified by Goh (1995), and the Olden algorithm (Olden et al. 2004). For both algorithms, the basis of these importance scores is the network's connection weights. The Garson algorithm determines variable importance by identifying all weighted connections between the nodes of interest. Olden's algorithm, on the other hand, uses the product of the raw connection weights between each input and output neuron and sums the product across all hidden neurons. This has been shown to outperform the Garson method in various simulations. For DNNs, a similar method due to Gedeon (1997) considers the weights connecting the input features to the first two hidden layers (for simplicity and speed); but this method can be slow for large networks.. To implement the Olden and Garson

algorithms, use type = "olden" and type = "garson", respectively. See garson and olden for details.

lm In (generalized) linear models, variable importance is typically based on the absolute value of the corresponding *t*-statistics. For such models, the sign of the original coefficient is also returned. By default, type = "stat" is used; however, if the inputs have been appropriately standardized then the raw coefficients can be used with type = "raw".

ml_feature_importances The Spark ML library provides standard variable importance for tree-based methods (e.g., random forests). See ml_feature_importances for details.

randomForest Random forests typically provide two measures of variable importance. The first measure is computed from permuting out-of-bag (OOB) data: for each tree, the prediction error on the OOB portion of the data is recorded (error rate for classification and MSE for regression). Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees in the forest, and normalized by the standard deviation of the differences. If the standard deviation of the differences is equal to 0 for a variable, the division is not done (but the average is almost always equal to 0 in that case). See importance for details, including additional arguments that can be passed via the ... argument.

The second measure is the total decrease in node impurities from splitting on the variable, averaged over all trees. For classification, the node impurity is measured by the Gini index. For regression, it is measured by residual sum of squares. See importance for details.

cforest Same approach described in cforest above. See varimp and varimpAUC (if type = "auc") for details.

ranger Variable importance for ranger objects is computed in the usual way for random forests. The approach used depends on the importance argument provided in the initial call to ranger. See importance for details.

rpart As stated in one of the **rpart** vignettes. A variable may appear in the tree many times, either as a primary or a surrogate variable. An overall measure of variable importance is the sum of the goodness of split measures for each split for which it was the primary variable, plus "goodness" * (adjusted agreement) for all splits in which it was a surrogate. Imagine two variables which were essentially duplicates of each other; if we did not count surrogates, they would split the importance with neither showing up as strongly as it should. See rpart for details.

train Various model-specific and model-agnostic approaches that depend on the learning algorithm employed in the original call to train. See varImp for details.

xgboost For linear models, the variable importance is the absolute magnitude of the estimated coefficients. For that reason, in order to obtain a meaningful ranking by importance for a linear model, the features need to be on the same scale (which you also would want to do when using either L1 or L2 regularization). Otherwise, the approach described in Friedman (2001) for gbms is used. See xgb.importance for details. For tree models, you can obtain three different types of variable importance:

- Using type = "gain" (the default) gives the fractional contribution of each feature to the model based on the total gain of the corresponding feature's splits.
- Using type = "cover" gives the number of observations related to each feature.
- Using type = "frequency" gives the percentages representing the relative number of times each feature has been used throughout each tree in the ensemble.

Value

A tidy data frame (i.e., a `"tibble"` object) with two columns: `Variable` and `Importance`. For `"lm"`/`"glm"`-like object, an additional column, called `Sign`, is also included which includes the sign (i.e., POS/NEG) of the original coefficient.

Note

Inspired by the `varImp` function.

`vi_pdp` *Deprecated*

Description

These functions have been deprecated and should not be used. They will be removed in the next release.

Usage

```
vi_pdp(...)

vi_ice(...)
```

Arguments

`...`	Arguments passed on to `vi_firm`.

`vi_permute` *Permutation-based variable importance*

Description

Compute permutation-based variable importance scores for the predictors in a model.

Usage

```
vi_permute(object, ...)

## Default S3 method:
vi_permute(
  object,
  feature_names = NULL,
  train = NULL,
  target = NULL,
  metric = NULL,
```

```
  smaller_is_better = NULL,
  type = c("difference", "ratio"),
  nsim = 1,
  keep = TRUE,
  sample_size = NULL,
  sample_frac = NULL,
  reference_class = NULL,
  pred_fun = NULL,
  pred_wrapper = NULL,
  verbose = FALSE,
  progress = "none",
  parallel = FALSE,
  paropts = NULL,
  ...
)
```

Arguments

object	A fitted model object (e.g., a "randomForest" object).
...	Additional optional arguments. (Currently ignored.)
feature_names	Character string giving the names of the predictor variables (i.e., features) of interest. If NULL (the default) then the internal 'get_feature_names()' function will be called to try and extract them automatically. It is good practice to always specify this argument.
train	A matrix-like R object (e.g., a data frame or matrix) containing the training data. If NULL (the default) then the internal 'get_training_data()' function will be called to try and extract it automatically. It is good practice to always specify this argument.
target	Either a character string giving the name (or position) of the target column in train or, if train only contains feature columns, a vector containing the target values used to train object.
metric	Either a function or character string specifying the performance metric to use in computing model performance (e.g., RMSE for regression or accuracy for binary classification). If metric is a function, then it requires two arguments, actual and predicted, and should return a single, numeric value. Ideally, this should be the same metric that was used to train object. See list_metrics for a list of built-in metrics.
smaller_is_better	Logical indicating whether or not a smaller value of metric is better. Default is NULL. Must be supplied if metric is a user-supplied function.
type	Character string specifying how to compare the baseline and permuted performance metrics. Current options are "difference" (the default) and "ratio".
nsim	Integer specifying the number of Monte Carlo replications to perform. Default is 1. If nsim > 1, the results from each replication are simply averaged together (the standard deviation will also be returned).
keep	Logical indicating whether or not to keep the individual permutation scores for all nsim repetitions. If TRUE (the default) then the individual variable importance

scores will be stored in an attribute called "raw_scores". (Only used when nsim > 1.)

sample_size
: Integer specifying the size of the random sample to use for each Monte Carlo repetition. Default is NULL (i.e., use all of the available training data). Cannot be specified with sample_frac. Can be used to reduce computation time with large data sets.

sample_frac
: Proportion specifying the size of the random sample to use for each Monte Carlo repetition. Default is NULL (i.e., use all of the available training data). Cannot be specified with sample_size. Can be used to reduce computation time with large data sets.

reference_class
: Character string specifying which response category represents the "reference" class (i.e., the class for which the predicted class probabilities correspond to). Only needed for binary classification problems.

pred_fun
: Deprecated. Use pred_wrapper instead.

pred_wrapper
: Prediction function that requires two arguments, object and newdata. The output of this function should be determined by the metric being used:

  Regression A numeric vector of predicted outcomes.

  Binary classification A vector of predicted class labels (e.g., if using misclassification error) or a vector of predicted class probabilities for the reference class (e.g., if using log loss or AUC).

  Multiclass classification A vector of predicted class labels (e.g., if using misclassification error) or a A matrix/data frame of predicted class probabilities for each class (e.g., if using log loss or AUC).

verbose
: Logical indicating whether or not to print information during the construction of variable importance scores. Default is FALSE.

progress
: Character string giving the name of the progress bar to use. See create_progress_bar for details. Default is "none".

parallel
: Logical indicating whether or not to run vi_permute() in parallel (using a backend provided by the foreach package). Default is FALSE. If TRUE, an appropriate backend must be provided by foreach.

paropts
: List containing additional options to be passed on to foreach when parallel = TRUE.

Details

Coming soon!

Value

A tidy data frame (i.e., a "tibble" object) with two columns: Variable and Importance.

Examples

```
## Not run:
# Load required packages
```

```
library(ggplot2)  # for ggtitle() function
library(nnet)     # for fitting neural networks

# Simulate training data
trn <- gen_friedman(500, seed = 101)  # ?vip::gen_friedman

# Inspect data
tibble::as_tibble(trn)

# Fit PPR and NN models (hyperparameters were chosen using the caret package
# with 5 repeats of 5-fold cross-validation)
pp <- ppr(y ~ ., data = trn, nterms = 11)
set.seed(0803) # for reproducibility
nn <- nnet(y ~ ., data = trn, size = 7, decay = 0.1, linout = TRUE,
           maxit = 500)

# Plot VI scores
set.seed(2021)  # for reproducibility
p1 <- vip(pp, method = "permute", target = "y", metric = "rsquared",
          pred_wrapper = predict) + ggtitle("PPR")
p2 <- vip(nn, method = "permute", target = "y", metric = "rsquared",
          pred_wrapper = predict) + ggtitle("NN")
grid.arrange(p1, p2, ncol = 2)

# Mean absolute error
mae <- function(actual, predicted) {
  mean(abs(actual - predicted))
}

# Permutation-based VIP with user-defined MAE metric
set.seed(1101)  # for reproducibility
vip(pp, method = "permute", target = "y", metric = mae,
    smaller_is_better = TRUE,
    pred_wrapper = function(object, newdata) predict(object, newdata)
) + ggtitle("PPR")

## End(Not run)
```

---

vi_shap *SHAP-based variable importance*

Description

Compute SHAP-based VI scores for the predictors in a model. See details below.

Usage

```
vi_shap(object, ...)

## Default S3 method:
vi_shap(object, feature_names = NULL, train = NULL, ...)
```

Arguments

object	A fitted model object (e.g., a "randomForest" object).
...	Additional optional arguments to be passed on to explain.
feature_names	Character string giving the names of the predictor variables (i.e., features) of interest. If NULL (the default) then the internal 'get_feature_names()' function will be called to try and extract them automatically. It is good practice to always specify this argument.
train	A matrix-like R object (e.g., a data frame or matrix) containing the training data. If NULL (the default) then the internal 'get_training_data()' function will be called to try and extract it automatically. It is good practice to always specify this argument.

Details

This approach to computing VI scores is based on the mean absolute value of the SHAP values for each feature; see, for example, https://github.com/slundberg/shap and the references therein.

Strumbelj, E., and Kononenko, I. Explaining prediction models and individual predictions with feature contributions. Knowledge and information systems 41.3 (2014): 647-665.

Value

A tidy data frame (i.e., a "tibble" object) with two columns, Variable and Importance, containing the variable name and its associated importance score, respectively.

Index