Package 'hstats'

February 3, 2024

Title Interaction Statistics

Version 1.1.2

Description Fast, model-agnostic implementation of different H-statistics introduced by Jerome H. Friedman and Bogdan E. Popescu (2008) <doi:10.1214/07-AOAS148>. These statistics quantify interaction strength per feature, feature pair, and feature triple. The package supports multi-output predictions and can account for case weights. In addition, several variants of the original statistics are provided. The shape of the interactions can be explored through partial dependence plots or individual conditional expectation plots. 'DALEX' explainers, meta learners ('mlr3', 'tidymodels', 'caret') and most other models work out-of-the-box.

License GPL (>= 2)

Depends R (>= 3.2.0)

Encoding UTF-8

RoxygenNote 7.3.1

Imports ggplot2, stats, utils

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/mayer79/hstats, https://mayer79.github.io/hstats/

BugReports https://github.com/mayer79/hstats/issues

NeedsCompilation no

Author Michael Mayer [aut, cre]

Maintainer Michael Mayer <mayermichael79@gmail.com>

Repository CRAN

Date/Publication 2024-02-03 15:50:02 UTC

R topics documented:

`average_loss` *Average Loss*

Description

Calculates the average loss of a model on a given dataset, optionally grouped by a variable. Use `plot()` to visualize the results.

Usage

```
average_loss(object, ...)

## Default S3 method:
average_loss(
  object,
  X,
  y,
```

```
  pred_fun = stats::predict,
  loss = "squared_error",
  agg_cols = FALSE,
  BY = NULL,
  by_size = 4L,
  w = NULL,
  ...
)

## S3 method for class 'ranger'
average_loss(
  object,
  X,
  y,
  pred_fun = function(m, X, ...) stats::predict(m, X, ...)$predictions,
  loss = "squared_error",
  agg_cols = FALSE,
  BY = NULL,
  by_size = 4L,
  w = NULL,
  ...
)

## S3 method for class 'explainer'
average_loss(
  object,
  X = object[["data"]],
  y = object[["y"]],
  pred_fun = object[["predict_function"]],
  loss = "squared_error",
  agg_cols = FALSE,
  BY = NULL,
  by_size = 4L,
  w = object[["weights"]],
  ...
)
```

Arguments

`object`	Fitted model object.
`...`	Additional arguments passed to `pred_fun(object, X, ...)`, for instance `type = "response"` in a `glm()` model, or `reshape = TRUE` in a multiclass XGBoost model.
`X`	A data.frame or matrix serving as background dataset.
`y`	Vector/matrix of the response, or the corresponding column names in `X`.
`pred_fun`	Prediction function of the form `function(object, X, ...)`, providing $K \geq 1$ predictions per row. Its first argument represents the model `object`, its second argument a data structure like `X`. Additional arguments (such as `type =`

	"response" in a GLM, or reshape = TRUE in a multiclass XGBoost model) can be passed via .... The default, stats::predict(), will work in most cases.
loss	One of "squared_error", "logloss", "mlogloss", "poisson", "gamma", "absolute_error", "classification_error". Alternatively, a loss function can be provided that turns observed and predicted values into a numeric vector or matrix of unit losses of the same length as X. For "mlogloss", the response y can either be a dummy matrix or a discrete vector. The latter case is handled via a fast version of model.matrix(~ as.factor(y) + 0). For "classification_error", both predictions and responses can be non-numeric. For "squared_error", both predictions and responses can be factors with identical levels. In this case, squared error is evaulated for each one-hot-encoded column.
agg_cols	Should multivariate losses be summed up? Default is FALSE. In combination with the squared error loss, agg_cols = TRUE gives the Brier score for (probabilistic) classification.
BY	Optional grouping vector or column name. Numeric BY variables with more than by_size disjoint values will be binned into by_size quantile groups of similar size.
by_size	Numeric BY variables with more than by_size unique values will be binned into quantile groups. Only relevant if BY is not NULL.
w	Optional vector of case weights. Can also be a column name of X.

Value

An object of class "hstats_matrix" containing these elements:

- M: Matrix of statistics (one column per prediction dimension), or NULL.
- SE: Matrix with standard errors of M, or NULL. Multiply with sqrt(m_rep) to get *standard deviations* instead. Currently, supported only for perm_importance().
- m_rep: The number of repetitions behind standard errors SE, or NULL. Currently, supported only for perm_importance().
- statistic: Name of the function that generated the statistic.
- description: Description of the statistic.

Methods (by class)

- average_loss(default): Default method.
- average_loss(ranger): Method for "ranger" models.
- average_loss(explainer): Method for DALEX "explainer".

Losses

The default loss is the "squared_error". Other choices:

- "absolute_error": The absolute error is the loss corresponding to median regression.
- "poisson": Unit Poisson deviance, i.e., the loss function used in Poisson regression. Actual values y and predictions must be non-negative.

- "gamma": Unit gamma deviance, i.e., the loss function of Gamma regression. Actual values y and predictions must be positive.
- "logloss": The Log Loss is the loss function used in logistic regression, and the top choice in probabilistic binary classification. Responses y and predictions must be between 0 and 1. Predictions represent probabilities of having a "1".
- "mlogloss": Multi-Log-Loss is the natural loss function in probabilistic multi-class situations. If there are K classes and n observations, the predictions form a (n x K) matrix of probabilities (with row-sums 1). The observed values y are either passed as (n x K) dummy matrix, or as discrete vector with corresponding levels. The latter case is turned into a dummy matrix by a fast version of `model.matrix(~ as.factor(y) + 0)`.
- "classification_error": Misclassification error. Both the observed values y and the predictions can be character/factor. This loss function can be used in non-probabilistic classification settings. BUT: Probabilistic classification (with "mlogloss") is clearly preferred in most situations.
- A function with signature `f(actual, predicted)`, returning a numeric vector or matrix of the same length as the input.

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ ., data = iris)
average_loss(fit, X = iris, y = "Sepal.Length")
average_loss(fit, X = iris, y = iris$Sepal.Length, BY = iris$Sepal.Width)
average_loss(fit, X = iris, y = "Sepal.Length", BY = "Sepal.Width")

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width + Species, data = iris)
average_loss(fit, X = iris, y = iris[, 1:2])
L <- average_loss(
  fit, X = iris, y = iris[, 1:2], loss = "gamma", BY = "Species"
)
L
plot(L)
```

`dim.hstats_matrix` *Dimensions of "hstats_matrix" Object*

Description

Implies `nrow()` and `ncol()`.

Usage

```
## S3 method for class 'hstats_matrix'
dim(x)
```

Arguments

x	An object of class "hstats_matrix".

Value

A numeric vector of length two providing the number of rows and columns of "M" object stored in x.

Examples

```
fit <- lm(Sepal.Length ~ . + Petal.Width:Species, data = iris)
s <- hstats(fit, X = iris[-1])
x <- h2_pairwise(s)
dim(x)
nrow(x)
ncol(x)
```

---

dimnames.hstats_matrix

Dimnames of "hstats_matrix" Object

Description

Extracts dimnames of the "M" matrix in x. Implies rownames() and colnames().

Usage

```
## S3 method for class 'hstats_matrix'
dimnames(x)
```

Arguments

x	An object of class "hstats_matrix".

Value

Dimnames of the statistics matrix.

Examples

```
fit <- lm(as.matrix(iris[1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
s <- hstats(fit, X = iris[3:5], verbose = FALSE)
x <- h2_pairwise(s)
dimnames(x)
rownames(x)
colnames(x)
```

dimnames<-.hstats_matrix *Dimnames (Replacement Method) of "hstats_matrix"*

Description

This implies `colnames(x) <- ...`.

Usage

```
## S3 replacement method for class 'hstats_matrix'
dimnames(x) <- value
```

Arguments

x	An object of class "hstats_matrix".
value	A list with rownames and column names compliant with $M (and $SE).

Value

Like x, but with replaced dimnames.

Examples

```
fit <- lm(as.matrix(iris[1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
s <- hstats(fit, X = iris[3:5], verbose = FALSE)
x <- h2_overall(s)
colnames(x) <- c("Sepal Length", "Sepal Width")
plot(x)

rownames(x)[2:3] <- c("Petal Width", "Petal Length")
plot(x)
```

h2 *Total Interaction Strength*

Description

Proportion of prediction variability unexplained by main effects of v, see Details. Use `plot()` to get a barplot.

Usage

```
h2(object, ...)

## Default S3 method:
h2(object, ...)

## S3 method for class 'hstats'
h2(object, normalize = TRUE, squared = TRUE, ...)
```

Arguments

object	Object of class "hstats".
...	Currently unused.
normalize	Should statistics be normalized? Default is TRUE.
squared	Should *squared* statistics be returned? Default is TRUE.

Details

If the model is additive in all features, then the (centered) prediction function F equals the sum of the (centered) partial dependence functions $F_j(x_j)$, i.e.,

$$F(\mathbf{x}) = \sum_j^p F_j(x_j)$$

(check partial_dep() for all definitions). To measure the relative amount of variability unexplained by all main effects, we can therefore study the test statistic of total interaction strength

$$H^2 = \frac{\frac{1}{n}\sum_{i=1}^n \left[F(\mathbf{x}_i) - \sum_{j=1}^p \hat{F}_j(x_{ij})\right]^2}{\frac{1}{n}\sum_{i=1}^n \left[F(\mathbf{x}_i)\right]^2}.$$

A value of 0 means there are no interaction effects at all. Due to (typically undesired) extrapolation effects, depending on the model, values above 1 may occur.

In Żółkowski et al. (2023), $1 - H^2$ is called *additivity index*. A similar measure using accumulated local effects is discussed in Molnar (2020).

Value

An object of class "hstats_matrix" containing these elements:

- M: Matrix of statistics (one column per prediction dimension), or NULL.
- SE: Matrix with standard errors of M, or NULL. Multiply with sqrt(m_rep) to get *standard deviations* instead. Currently, supported only for perm_importance().
- m_rep: The number of repetitions behind standard errors SE, or NULL. Currently, supported only for perm_importance().
- statistic: Name of the function that generated the statistic.
- description: Description of the statistic.

Methods (by class)

- `h2(default)`: Default method of total interaction strength.
- `h2(hstats)`: Total interaction strength from "interact" object.

References

1. Żółkowski, Artur, Mateusz Krzyziński, and Paweł Fijałkowski. *Methods for extraction of interactions from predictive models.* Undergraduate thesis. Faculty of Mathematics and Information Science, Warsaw University of Technology (2023).
2. Molnar, Christoph, Giuseppe Casalicchio, and Bernd Bischl". *Quantifying Model Complexity via Functional Decomposition for Better Post-hoc Interpretability*, in Machine Learning and Knowledge Discovery in Databases, Springer International Publishing (2020): 193-204.

See Also

`hstats()`, `h2_overall()`, `h2_pairwise()`, `h2_threeway()`

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ . + Petal.Width:Species, data = iris)
s <- hstats(fit, X = iris[, -1])
h2(s)

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
s <- hstats(fit, X = iris[, 3:5])
h2(s)

# MODEL 3: No interactions
fit <- lm(Sepal.Length ~ ., data = iris)
s <- hstats(fit, X = iris[, -1], verbose = FALSE)
h2(s)
```

`h2_overall` *Overall Interaction Strength*

Description

Friedman and Popescu's statistic of overall interaction strength per feature, see Details. Use `plot()` to get a barplot.

Usage

```
h2_overall(object, ...)

## Default S3 method:
h2_overall(object, ...)

## S3 method for class 'hstats'
h2_overall(
  object,
  normalize = TRUE,
  squared = TRUE,
  sort = TRUE,
  zero = TRUE,
  ...
)
```

Arguments

object	Object of class "hstats".
...	Currently unused.
normalize	Should statistics be normalized? Default is TRUE.
squared	Should *squared* statistics be returned? Default is TRUE.
sort	Should results be sorted? Default is TRUE. (Multi-output is sorted by row means.)
zero	Should rows with all 0 be shown? Default is TRUE.

Details

The logic of Friedman and Popescu (2008) is as follows: If there are no interactions involving feature x_j, we can decompose the (centered) prediction function F into the sum of the (centered) partial dependence F_j on x_j and the (centered) partial dependence $F_{\setminus j}$ on all other features $\mathbf{x}_{\setminus j}$, i.e.,

$$F(\mathbf{x}) = F_j(x_j) + F_{\setminus j}(\mathbf{x}_{\setminus j}).$$

Correspondingly, Friedman and Popescu's statistic of overall interaction strength of x_j is given by

$$H_j^2 = \frac{\frac{1}{n}\sum_{i=1}^{n}\left[F(\mathbf{x}_i) - \hat{F}_j(x_{ij}) - \hat{F}_{\setminus j}(\mathbf{x}_{i\setminus j})\right]^2}{\frac{1}{n}\sum_{i=1}^{n}\left[F(\mathbf{x}_i)\right]^2}$$

(check partial_dep() for all definitions).

Remarks:

1. Partial dependence functions (and F) are all centered to (possibly weighted) mean 0.
2. Partial dependence functions (and F) are evaluated over the data distribution. This is different to partial dependence plots, where one uses a fixed grid.
3. Weighted versions follow by replacing all arithmetic means by corresponding weighted means.
4. Multivariate predictions can be treated in a component-wise manner.

5. Due to (typically undesired) extrapolation effects of partial dependence functions, depending on the model, values above 1 may occur.
6. $H_j^2 = 0$ means there are no interactions associated with x_j. The higher the value, the more prediction variability comes from interactions with x_j.
7. Since the denominator is the same for all features, the values of the test statistics can be compared across features.

Value

An object of class "hstats_matrix" containing these elements:

- `M`: Matrix of statistics (one column per prediction dimension), or `NULL`.
- `SE`: Matrix with standard errors of `M`, or `NULL`. Multiply with `sqrt(m_rep)` to get *standard deviations* instead. Currently, supported only for `perm_importance()`.
- `m_rep`: The number of repetitions behind standard errors `SE`, or `NULL`. Currently, supported only for `perm_importance()`.
- `statistic`: Name of the function that generated the statistic.
- `description`: Description of the statistic.

Methods (by class)

- `h2_overall(default)`: Default method of overall interaction strength.
- `h2_overall(hstats)`: Overall interaction strength from "hstats" object.

References

Friedman, Jerome H., and Bogdan E. Popescu. *"Predictive Learning via Rule Ensembles."* The Annals of Applied Statistics 2, no. 3 (2008): 916-54.

See Also

`hstats()`, `h2()`, `h2_pairwise()`, `h2_threeway()`

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ . + Petal.Width:Species, data = iris)
s <- hstats(fit, X = iris[, -1])
h2_overall(s)
plot(h2_overall(s))

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
s <- hstats(fit, X = iris[, 3:5], verbose = FALSE)
plot(h2_overall(s, zero = FALSE))
```

h2_pairwise *Pairwise Interaction Strength*

Description

Friedman and Popescu's statistic of pairwise interaction strength, see Details. Use `plot()` to get a barplot.

Usage

```
h2_pairwise(object, ...)

## Default S3 method:
h2_pairwise(object, ...)

## S3 method for class 'hstats'
h2_pairwise(
  object,
  normalize = TRUE,
  squared = TRUE,
  sort = TRUE,
  zero = TRUE,
  ...
)
```

Arguments

`object`	Object of class "hstats".
`...`	Currently unused.
`normalize`	Should statistics be normalized? Default is `TRUE`.
`squared`	Should *squared* statistics be returned? Default is `TRUE`.
`sort`	Should results be sorted? Default is `TRUE`. (Multi-output is sorted by row means.)
`zero`	Should rows with all 0 be shown? Default is `TRUE`.

Details

Following Friedman and Popescu (2008), if there are no interaction effects between features x_j and x_k, their two-dimensional (centered) partial dependence function F_{jk} can be written as the sum of the (centered) univariate partial dependencies F_j and F_k, i.e.,

$$F_{jk}(x_j, x_k) = F_j(x_j) + F_k(x_k).$$

Correspondingly, Friedman and Popescu's statistic of pairwise interaction strength between x_j and x_k is defined as

$$H^2_{jk} = \frac{A_{jk}}{\frac{1}{n}\sum_{i=1}^{n}\left[\hat{F}_{jk}(x_{ij}, x_{ik})\right]^2},$$

where

$$A_{jk} = \frac{1}{n}\sum_{i=1}^{n}\left[\hat{F}_{jk}(x_{ij}, x_{ik}) - \hat{F}_j(x_{ij}) - \hat{F}_k(x_{ik})\right]^2$$

(check `partial_dep()` for all definitions).

Remarks:

1. Remarks 1 to 5 of `h2_overall()` also apply here.
2. $H^2_{jk} = 0$ means there are no interaction effects between x_j and x_k. The larger the value, the more of the joint effect of the two features comes from the interaction.
3. Since the denominator differs between variable pairs, unlike H_j, this test statistic is difficult to compare between variable pairs. If both main effects are very weak, a negligible interaction can get a high H^2_{jk}. Therefore, Friedman and Popescu (2008) suggests to calculate H^2_{jk} only for *important* variables (see "Modification" below).

Modification

To be better able to compare pairwise interaction strength across variable pairs, and to overcome the problem mentioned in the last remark, we suggest as alternative the unnormalized test statistic on the scale of the predictions, i.e., $\sqrt{A_{jk}}$. Set `normalize = FALSE` and `squared = FALSE` to obtain this statistic. Furthermore, instead of focusing on pairwise calculations for the most *important* features, we can select features with *strongest overall interactions*.

Value

An object of class "hstats_matrix" containing these elements:

- `M`: Matrix of statistics (one column per prediction dimension), or `NULL`.
- `SE`: Matrix with standard errors of `M`, or `NULL`. Multiply with `sqrt(m_rep)` to get *standard deviations* instead. Currently, supported only for `perm_importance()`.
- `m_rep`: The number of repetitions behind standard errors `SE`, or `NULL`. Currently, supported only for `perm_importance()`.
- `statistic`: Name of the function that generated the statistic.
- `description`: Description of the statistic.

Methods (by class)

- `h2_pairwise(default)`: Default pairwise interaction strength.
- `h2_pairwise(hstats)`: Pairwise interaction strength from "hstats" object.

References

Friedman, Jerome H., and Bogdan E. Popescu. *"Predictive Learning via Rule Ensembles."* The Annals of Applied Statistics 2, no. 3 (2008): 916-54.

See Also

`hstats()`, `h2()`, `h2_overall()`, `h2_threeway()`

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ . + Petal.Width:Species, data = iris)
s <- hstats(fit, X = iris[, -1])

# Proportion of joint effect coming from pairwise interaction
# (for features with strongest overall interactions)
h2_pairwise(s)
h2_pairwise(s, zero = FALSE)  # Drop 0

# Absolute measure as alternative
abs_h <- h2_pairwise(s, normalize = FALSE, squared = FALSE, zero = FALSE)
abs_h
abs_h$M

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
s <- hstats(fit, X = iris[, 3:5], verbose = FALSE)
x <- h2_pairwise(s)
plot(x)
```

---

h2_threeway *Three-way Interaction Strength*

Description

Friedman and Popescu's statistic of three-way interaction strength, see Details. Use `plot()` to get a barplot. In `hstats()`, set `threeway_m` to a value above 2 to calculate this statistic for all feature triples of the `threeway_m` features with strongest overall interaction.

Usage

```
h2_threeway(object, ...)

## Default S3 method:
h2_threeway(object, ...)

## S3 method for class 'hstats'
h2_threeway(
  object,
  normalize = TRUE,
  squared = TRUE,
  sort = TRUE,
  zero = TRUE,
  ...
)
```

Arguments

`object`	Object of class "hstats".
`...`	Currently unused.
`normalize`	Should statistics be normalized? Default is `TRUE`.
`squared`	Should *squared* statistics be returned? Default is `TRUE`.
`sort`	Should results be sorted? Default is `TRUE`. (Multi-output is sorted by row means.)
`zero`	Should rows with all 0 be shown? Default is `TRUE`.

Details

Friedman and Popescu (2008) describe a test statistic to measure three-way interactions: in case there are no three-way interactions between features x_j, x_k and x_l, their (centered) three-dimensional partial dependence function F_{jkl} can be decomposed into lower order terms:

$$F_{jkl}(x_j, x_k, x_l) = B_{jkl} - C_{jkl}$$

with

$$B_{jkl} = F_{jk}(x_j, x_k) + F_{jl}(x_j, x_l) + F_{kl}(x_k, x_l)$$

and

$$C_{jkl} = F_j(x_j) + F_k(x_k) + F_l(x_l).$$

The squared and scaled difference between the two sides of the equation leads to the statistic

$$H^2_{jkl} = \frac{\frac{1}{n}\sum_{i=1}^{n}\left[\hat{F}_{jkl}(x_{ij}, x_{ik}, x_{il}) - B^{(i)}_{jkl} + C^{(i)}_{jkl}\right]^2}{\frac{1}{n}\sum_{i=1}^{n}\hat{F}_{jkl}(x_{ij}, x_{ik}, x_{il})^2},$$

where

$$B^{(i)}_{jkl} = \hat{F}_{jk}(x_{ij}, x_{ik}) + \hat{F}_{jl}(x_{ij}, x_{il}) + \hat{F}_{kl}(x_{ik}, x_{il})$$

and

$$C^{(i)}_{jkl} = \hat{F}_j(x_{ij}) + \hat{F}_k(x_{ik}) + \hat{F}_l(x_{il}).$$

Similar remarks as for `h2_pairwise()` apply.

Value

An object of class "hstats_matrix" containing these elements:

- `M`: Matrix of statistics (one column per prediction dimension), or `NULL`.
- `SE`: Matrix with standard errors of `M`, or `NULL`. Multiply with `sqrt(m_rep)` to get *standard deviations* instead. Currently, supported only for `perm_importance()`.
- `m_rep`: The number of repetitions behind standard errors `SE`, or `NULL`. Currently, supported only for `perm_importance()`.
- `statistic`: Name of the function that generated the statistic.
- `description`: Description of the statistic.

Methods (by class)

- `h2_threeway(default)`: Default pairwise interaction strength.
- `h2_threeway(hstats)`: Pairwise interaction strength from "hstats" object.

References

Friedman, Jerome H., and Bogdan E. Popescu. *"Predictive Learning via Rule Ensembles."* The Annals of Applied Statistics 2, no. 3 (2008): 916-54.

See Also

`hstats()`, `h2()`, `h2_overall()`, `h2_pairwise()`

Examples

```
# MODEL 1: Linear regression
fit <- lm(uptake ~ Type * Treatment * conc, data = CO2)
s <- hstats(fit, X = CO2[, 2:4], threeway_m = 5)
h2_threeway(s)

#' MODEL 2: Multivariate output (taking just twice the same response as example)
fit <- lm(cbind(up = uptake, up2 = 2 * uptake) ~ Type * Treatment * conc, data = CO2)
s <- hstats(fit, X = CO2[, 2:4], threeway_m = 5)
h2_threeway(s)
h2_threeway(s, normalize = FALSE, squared = FALSE)  # Unnormalized H
plot(h2_threeway(s))
```

`hstats` *Calculate Interaction Statistics*

Description

This is the main function of the package. It does the expensive calculations behind the following H-statistics:

- Total interaction strength H^2, a statistic measuring the proportion of prediction variability unexplained by main effects of `v`, see `h2()` for details.
- Friedman and Popescu's statistic H_j^2 of overall interaction strength per feature, see `h2_overall()` for details.
- Friedman and Popescu's statistic H_{jk}^2 of pairwise interaction strength, see `h2_pairwise()` for details.
- Friedman and Popescu's statistic H_{jkl}^2 of three-way interaction strength, see `h2_threeway()` for details. To save time, this statistic is not calculated by default. Set `threeway_m` to a value above 2 to get three-way statistics of the `threeway_m` variables with strongest overall interaction.

Furthermore, it allows to calculate an experimental partial dependence based measure of feature importance, PDI_j^2. It equals the proportion of prediction variability unexplained by other features, see `pd_importance()` for details. This statistic is not shown by `summary()` or `plot()`.

Instead of using `summary()`, interaction statistics can also be obtained via the more flexible functions `h2()`, `h2_overall()`, `h2_pairwise()`, and `h2_threeway()`.

Usage

```
hstats(object, ...)

## Default S3 method:
hstats(
  object,
  X,
  v = NULL,
  pred_fun = stats::predict,
  pairwise_m = 5L,
  threeway_m = 0L,
  approx = FALSE,
  grid_size = 50L,
  n_max = 500L,
  eps = 1e-10,
  w = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'ranger'
hstats(
  object,
  X,
  v = NULL,
  pred_fun = function(m, X, ...) stats::predict(m, X, ...)$predictions,
  pairwise_m = 5L,
  threeway_m = 0L,
  approx = FALSE,
  grid_size = 50L,
  n_max = 500L,
  eps = 1e-10,
  w = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'explainer'
hstats(
  object,
  X = object[["data"]],
```

```
  v = NULL,
  pred_fun = object[["predict_function"]],
  pairwise_m = 5L,
  threeway_m = 0L,
  approx = FALSE,
  grid_size = 50L,
  n_max = 500L,
  eps = 1e-10,
  w = object[["weights"]],
  verbose = TRUE,
  ...
)
```

Arguments

object	Fitted model object.
...	Additional arguments passed to pred_fun(object, X, ...), for instance type = "response" in a glm() model, or reshape = TRUE in a multiclass XGBoost model.
X	A data.frame or matrix serving as background dataset.
v	Vector of feature names. The default (NULL) will use all column names of X except the column name of the optional case weight w (if specified as name).
pred_fun	Prediction function of the form function(object, X, ...), providing $K \geq 1$ predictions per row. Its first argument represents the model object, its second argument a data structure like X. Additional arguments (such as type = "response" in a GLM, or reshape = TRUE in a multiclass XGBoost model) can be passed via .... The default, stats::predict(), will work in most cases.
pairwise_m	Number of features for which pairwise statistics are to be calculated. The features are selected based on Friedman and Popescu's overall interaction strength H_j^2. Set to to 0 to avoid pairwise calculations. For multivariate predictions, the union of the pairwise_m column-wise strongest variable names is taken. This can lead to very long run-times.
threeway_m	Like pairwise_m, but controls the feature count for three-way interactions. Cannot be larger than pairwise_m. To save computation time, the default is 0.
approx	Should quantile approximation be applied to dense numeric features? The default is FALSE. Setting this option to TRUE brings a massive speed-up for one-way calculations. It can, e.g., be used when the number of features is very large.
grid_size	Integer controlling the number of quantile midpoints used to approximate dense numerics. The quantile midpoints are calculated after subampling via n_max. Only relevant if approx = TRUE.
n_max	If X has more than n_max rows, a random sample of n_max rows is selected from X. In this case, set a random seed for reproducibility.
eps	Threshold below which numerator values are set to 0. Default is 1e-10.
w	Optional vector of case weights. Can also be a column name of X.
verbose	Should a progress bar be shown? The default is TRUE.

Value

An object of class "hstats" containing these elements:

- `X`: Input `X` (sampled to `n_max` rows, after optional quantile approximation).
- `w`: Case weight vector `w` (sampled to `n_max` values), or `NULL`.
- `v`: Vector of column names in `X` for which overall H statistics have been calculated.
- `f`: Matrix with (centered) predictions F.
- `mean_f2`: (Weighted) column means of `f`. Used to normalize H^2 and H_j^2.
- `F_j`: List of matrices, each representing (centered) partial dependence functions F_j.
- `F_not_j`: List of matrices with (centered) partial dependence functions $F_{\setminus j}$ of other features.
- `K`: Number of columns of prediction matrix.
- `pred_names`: Column names of prediction matrix.
- `pairwise_m`: Like input `pairwise_m`, but capped at `length(v)`.
- `threeway_m`: Like input `threeway_m`, but capped at the smaller of `length(v)` and `pairwise_m`.
- `eps`: Like input `eps`.
- `pd_importance`: List with numerator and denominator of PDI_j.
- `h2`: List with numerator and denominator of H^2.
- `h2_overall`: List with numerator and denominator of H_j^2.
- `v_pairwise`: Subset of `v` with largest H_j^2 used for pairwise calculations. Only if pairwise calculations have been done.
- `combs2`: Named list of variable pairs for which pairwise partial dependence functions are available. Only if pairwise calculations have been done.
- `F_jk`: List of matrices, each representing (centered) bivariate partial dependence functions F_{jk}. Only if pairwise calculations have been done.
- `h2_pairwise`: List with numerator and denominator of H_{jk}^2. Only if pairwise calculations have been done.
- `v_threeway`: Subset of `v` with largest `h2_overall()` used for three-way calculations. Only if three-way calculations have been done.
- `combs3`: Named list of variable triples for which three-way partial dependence functions are available. Only if three-way calculations have been done.
- `F_jkl`: List of matrices, each representing (centered) three-way partial dependence functions F_{jkl}. Only if three-way calculations have been done.
- `h2_threeway`: List with numerator and denominator of H_{jkl}^2. Only if three-way calculations have been done.

Methods (by class)

- `hstats(default)`: Default hstats method.
- `hstats(ranger)`: Method for "ranger" models.
- `hstats(explainer)`: Method for DALEX "explainer".

References

Friedman, Jerome H., and Bogdan E. Popescu. *"Predictive Learning via Rule Ensembles."* The Annals of Applied Statistics 2, no. 3 (2008): 916-54.

See Also

`h2()`, `h2_overall()`, `h2_pairwise()`, `h2_threeway()`, and `pd_importance()` for specific statistics calculated from the resulting object.

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ . + Petal.Width:Species, data = iris)
s <- hstats(fit, X = iris[, -1])
s
plot(s)
plot(s, zero = FALSE)  # Drop 0
summary(s)

# Absolute pairwise interaction strengths
h2_pairwise(s, normalize = FALSE, squared = FALSE, zero = FALSE)

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
s <- hstats(fit, X = iris[, 3:5], verbose = FALSE)
plot(s)
summary(s)

# MODEL 3: Gamma GLM with log link
fit <- glm(Sepal.Length ~ ., data = iris, family = Gamma(link = log))

# No interactions for additive features, at least on link scale
s <- hstats(fit, X = iris[, -1], verbose = FALSE)
summary(s)

# On original scale, we have interactions everywhere.
# To see three-way interactions, we set threeway_m to a value above 2.
s <- hstats(fit, X = iris[, -1], type = "response", threeway_m = 5)
plot(s, ncol = 1)  # All three types use different denominators

# All statistics on same scale (of predictions)
plot(s, squared = FALSE, normalize = FALSE, facet_scale = "free_y")
```

---

ice *Individual Conditional Expectations*

Description

Disaggregated partial dependencies, see reference. The plot method supports up to two grouping variables via `BY`.

Usage

```
ice(object, ...)

## Default S3 method:
ice(
  object,
  v,
  X,
  pred_fun = stats::predict,
  BY = NULL,
  grid = NULL,
  grid_size = 49L,
  trim = c(0.01, 0.99),
  strategy = c("uniform", "quantile"),
  na.rm = TRUE,
  n_max = 100L,
  ...
)

## S3 method for class 'ranger'
ice(
  object,
  v,
  X,
  pred_fun = function(m, X, ...) stats::predict(m, X, ...)$predictions,
  BY = NULL,
  grid = NULL,
  grid_size = 49L,
  trim = c(0.01, 0.99),
  strategy = c("uniform", "quantile"),
  na.rm = TRUE,
  n_max = 100L,
  ...
)

## S3 method for class 'explainer'
ice(
  object,
  v = v,
  X = object[["data"]],
  pred_fun = object[["predict_function"]],
  BY = NULL,
  grid = NULL,
  grid_size = 49L,
  trim = c(0.01, 0.99),
  strategy = c("uniform", "quantile"),
  na.rm = TRUE,
  n_max = 100L,
```

```
  ...
)
```

Arguments

object	Fitted model object.
...	Additional arguments passed to pred_fun(object, X, ...), for instance type = "response" in a glm() model, or reshape = TRUE in a multiclass XGBoost model.
v	One or more column names over which you want to calculate the ICE.
X	A data.frame or matrix serving as background dataset.
pred_fun	Prediction function of the form function(object, X, ...), providing $K \geq 1$ predictions per row. Its first argument represents the model object, its second argument a data structure like X. Additional arguments (such as type = "response" in a GLM, or reshape = TRUE in a multiclass XGBoost model) can be passed via .... The default, stats::predict(), will work in most cases.
BY	Optional grouping vector/matrix/data.frame (up to two columns), or up to two column names. Unlike with partial_dep(), these variables are not binned. The first variable is visualized on the color scale, while the second one goes into a facet_wrap(). Thus, make sure that the second variable is discrete.
grid	Evaluation grid. A vector (if length(v) == 1L), or a matrix/data.frame otherwise. If NULL, calculated via multivariate_grid().
grid_size	Controls the approximate grid size. If x has p columns, then each (non-discrete) column will be reduced to about the p-th root of grid_size values.
trim	The default c(0.01, 0.99) means that values outside the 1% and 99% quantiles of non-discrete numeric columns are removed before calculation of grid values. Set to 0:1 for no trimming.
strategy	How to find grid values of non-discrete numeric columns? Either "uniform" or "quantile", see description of univariate_grid().
na.rm	Should missing values be dropped from the grid? Default is TRUE.
n_max	If X has more than n_max rows, a random sample of n_max rows is selected from X. In this case, set a random seed for reproducibility.

Value

An object of class "ice" containing these elements:

- data: data.frame containing the ice values.
- grid: Vector, matrix or data.frame of grid values.
- v: Same as input v.
- K: Number of columns of prediction matrix.
- pred_names: Column names of prediction matrix.
- by_names: Column name(s) of grouping variable(s) (or NULL).

Methods (by class)

- `ice(default)`: Default method.
- `ice(ranger)`: Method for "ranger" models.
- `ice(explainer)`: Method for DALEX "explainer".

References

Goldstein, Alex, and Adam Kapelner and Justin Bleich and Emil Pitkin. *Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation.* Journal of Computational and Graphical Statistics, 24, no. 1 (2015): 44-65.

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ . + Species * Petal.Length, data = iris)
plot(ice(fit, v = "Sepal.Width", X = iris))

# Stratified by one variable
ic <- ice(fit, v = "Petal.Length", X = iris, BY = "Species")
ic
plot(ic)
plot(ic, center = TRUE)

## Not run:
# Stratified by two variables (the second one goes into facets)
ic <- ice(fit, v = "Petal.Length", X = iris, BY = c("Petal.Width", "Species"))
plot(ic)
plot(ic, center = TRUE)

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
ic <- ice(fit, v = "Petal.Width", X = iris, BY = iris$Species)
plot(ic)
plot(ic, center = TRUE)
plot(ic, swap_dim = TRUE)

## End(Not run)

# MODEL 3: Gamma GLM -> pass options to predict() via ...
fit <- glm(Sepal.Length ~ ., data = iris, family = Gamma(link = log))
plot(ice(fit, v = "Petal.Length", X = iris, BY = "Species"))
plot(ice(fit, v = "Petal.Length", X = iris, type = "response", BY = "Species"))
```

`multivariate_grid` *Multivariate Grid*

Description

This function creates a multivariate grid. Each column of the input `x` is turned (independently) into a vector of grid values via `univariate_grid()`. Combinations are then formed by calling `expand.grid()`.

Usage

```
multivariate_grid(
  x,
  grid_size = 49L,
  trim = c(0.01, 0.99),
  strategy = c("uniform", "quantile"),
  na.rm = TRUE
)
```

Arguments

`x`	A vector, matrix, or data.frame to turn into a grid of values.
`grid_size`	Controls the approximate grid size. If `x` has p columns, then each (non-discrete) column will be reduced to about the p-th root of `grid_size` values.
`trim`	The default `c(0.01, 0.99)` means that values outside the 1% and 99% quantiles of non-discrete numeric columns are removed before calculation of grid values. Set to `0:1` for no trimming.
`strategy`	How to find grid values of non-discrete numeric columns? Either "uniform" or "quantile", see description of `univariate_grid()`.
`na.rm`	Should missing values be dropped from the grid? Default is `TRUE`.

Value

A vector, matrix, or data.frame with evaluation points.

See Also

`univariate_grid()`

Examples

```
multivariate_grid(iris[1:2], grid_size = 4)
multivariate_grid(iris$Species)  # Works also in the univariate case
```

partial_dep *Partial Dependence Plot*

Description

Estimates the partial dependence function of feature(s) v over a grid of values. Both multivariate and multivariable situations are supported. The resulting object can be plotted via `plot()`.

Usage

```
partial_dep(object, ...)

## Default S3 method:
partial_dep(
  object,
  v,
  X,
  pred_fun = stats::predict,
  BY = NULL,
  by_size = 4L,
  grid = NULL,
  grid_size = 49L,
  trim = c(0.01, 0.99),
  strategy = c("uniform", "quantile"),
  na.rm = TRUE,
  n_max = 1000L,
  w = NULL,
  ...
)

## S3 method for class 'ranger'
partial_dep(
  object,
  v,
  X,
  pred_fun = function(m, X, ...) stats::predict(m, X, ...)$predictions,
  BY = NULL,
  by_size = 4L,
  grid = NULL,
  grid_size = 49L,
  trim = c(0.01, 0.99),
  strategy = c("uniform", "quantile"),
  na.rm = TRUE,
  n_max = 1000L,
  w = NULL,
  ...
)
```

```
## S3 method for class 'explainer'
partial_dep(
  object,
  v,
  X = object[["data"]],
  pred_fun = object[["predict_function"]],
  BY = NULL,
  by_size = 4L,
  grid = NULL,
  grid_size = 49L,
  trim = c(0.01, 0.99),
  strategy = c("uniform", "quantile"),
  na.rm = TRUE,
  n_max = 1000L,
  w = object[["weights"]],
  ...
)
```

Arguments

object	Fitted model object.
...	Additional arguments passed to pred_fun(object, X, ...), for instance type = "response" in a glm() model, or reshape = TRUE in a multiclass XGBoost model.
v	One or more column names over which you want to calculate the partial dependence.
X	A data.frame or matrix serving as background dataset.
pred_fun	Prediction function of the form function(object, X, ...), providing $K \geq 1$ predictions per row. Its first argument represents the model object, its second argument a data structure like X. Additional arguments (such as type = "response" in a GLM, or reshape = TRUE in a multiclass XGBoost model) can be passed via .... The default, stats::predict(), will work in most cases.
BY	Optional grouping vector or column name. The partial dependence function is calculated per BY group. Each BY group uses the same evaluation grid to improve assessment of (non-)additivity. Numeric BY variables with more than by_size disjoint values will be binned into by_size quantile groups of similar size. To improve robustness, subsampling of X is done within group. This only applies to BY groups with more than n_max rows.
by_size	Numeric BY variables with more than by_size unique values will be binned into quantile groups. Only relevant if BY is not NULL.
grid	Evaluation grid. A vector (if length(v) == 1L), or a matrix/data.frame otherwise. If NULL, calculated via multivariate_grid().
grid_size	Controls the approximate grid size. If x has p columns, then each (non-discrete) column will be reduced to about the p-th root of grid_size values.

`trim`	The default `c(0.01, 0.99)` means that values outside the 1% and 99% quantiles of non-discrete numeric columns are removed before calculation of grid values. Set to `0:1` for no trimming.
`strategy`	How to find grid values of non-discrete numeric columns? Either "uniform" or "quantile", see description of `univariate_grid()`.
`na.rm`	Should missing values be dropped from the grid? Default is `TRUE`.
`n_max`	If `X` has more than `n_max` rows, a random sample of `n_max` rows is selected from `X`. In this case, set a random seed for reproducibility.
`w`	Optional vector of case weights. Can also be a column name of `X`.

Value

An object of class "partial_dep" containing these elements:

- `data`: data.frame containing the partial dependencies.
- `v`: Same as input `v`.
- `K`: Number of columns of prediction matrix.
- `pred_names`: Column names of prediction matrix.
- `by_name`: Column name of grouping variable (or `NULL`).

Methods (by class)

- `partial_dep(default)`: Default method.
- `partial_dep(ranger)`: Method for "ranger" models.
- `partial_dep(explainer)`: Method for DALEX "explainer".

Partial Dependence Functions

Let $F : R^p \to R$ denote the prediction function that maps the p-dimensional feature vector $\mathbf{x} = (x_1, \ldots, x_p)$ to its prediction. Furthermore, let

$$F_s(\mathbf{x}_s) = E_{\mathbf{x}_{\setminus s}}(F(\mathbf{x}_s, \mathbf{x}_{\setminus s}))$$

be the partial dependence function of F on the feature subset $\mathbf{x}_s$, where $s \subseteq \{1, \ldots, p\}$, as introduced in Friedman (2001). Here, the expectation runs over the joint marginal distribution of features $\mathbf{x}_{\setminus s}$ not in $\mathbf{x}_s$.

Given data, $F_s(\mathbf{x}_s)$ can be estimated by the empirical partial dependence function

$$\hat{F}_s(\mathbf{x}_s) = \frac{1}{n} \sum_{i=1}^{n} F(\mathbf{x}_s, \mathbf{x}_{i\setminus s}),$$

where $\mathbf{x}_{i\setminus s}$ $i = 1, \ldots, n$, are the observed values of $\mathbf{x}_{\setminus s}$.

A partial dependence plot (PDP) plots the values of $\hat{F}_s(\mathbf{x}_s)$ over a grid of evaluation points $\mathbf{x}_s$.

References

Friedman, Jerome H. *"Greedy Function Approximation: A Gradient Boosting Machine."* Annals of Statistics 29, no. 5 (2001): 1189-1232.

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ . + Species * Petal.Length, data = iris)
(pd <- partial_dep(fit, v = "Species", X = iris))
plot(pd)

## Not run:
# Stratified by BY variable (numerics are automatically binned)
pd <- partial_dep(fit, v = "Species", X = iris, BY = "Petal.Length")
plot(pd)

# Multivariable input
v <- c("Species", "Petal.Length")
pd <- partial_dep(fit, v = v, X = iris, grid_size = 100L)
plot(pd, rotate_x = TRUE)
plot(pd, d2_geom = "line")  # often better to read

# With grouping
pd <- partial_dep(fit, v = v, X = iris, grid_size = 100L, BY = "Petal.Width")
plot(pd, rotate_x = TRUE)
plot(pd, rotate_x = TRUE, d2_geom = "line")
plot(pd, rotate_x = TRUE, d2_geom = "line", swap_dim = TRUE)

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
pd <- partial_dep(fit, v = "Petal.Width", X = iris, BY = "Species")
plot(pd, show_points = FALSE)
pd <- partial_dep(fit, v = c("Species", "Petal.Width"), X = iris)
plot(pd, rotate_x = TRUE)
plot(pd, d2_geom = "line", rotate_x = TRUE)
plot(pd, d2_geom = "line", rotate_x = TRUE, swap_dim = TRUE)

# Multivariate, multivariable, and BY (no plot available)
pd <- partial_dep(
  fit, v = c("Petal.Width", "Petal.Length"), X = iris, BY = "Species"
)
pd

## End(Not run)

# MODEL 3: Gamma GLM -> pass options to predict() via ...
fit <- glm(Sepal.Length ~ ., data = iris, family = Gamma(link = log))
plot(partial_dep(fit, v = "Petal.Length", X = iris), show_points = FALSE)
plot(partial_dep(fit, v = "Petal.Length", X = iris, type = "response"))
```

`pd_importance` *PD Bases Importance (Experimental)*

Description

Experimental variable importance method based on partial dependence functions. While related to Greenwell et al., our suggestion measures not only main effect strength but also interaction effects. It is very closely related to H_j^2, see Details. Use `plot()` to get a barplot.

Usage

```
pd_importance(object, ...)

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

## S3 method for class 'hstats'
pd_importance(
  object,
  normalize = TRUE,
  squared = TRUE,
  sort = TRUE,
  zero = TRUE,
  ...
)
```

Arguments

`object`	Object of class "hstats".
`...`	Currently unused.
`normalize`	Should statistics be normalized? Default is `TRUE`.
`squared`	Should *squared* statistics be returned? Default is `TRUE`.
`sort`	Should results be sorted? Default is `TRUE`. (Multi-output is sorted by row means.)
`zero`	Should rows with all 0 be shown? Default is `TRUE`.

Details

If x_j has no effects, the (centered) prediction function F equals the (centered) partial dependence $F_{\setminus j}$ on all other features $\mathbf{x}_{\setminus j}$, i.e.,

$$F(\mathbf{x}) = F_{\setminus j}(\mathbf{x}_{\setminus j}).$$

Therefore, the following measure of variable importance follows:

$$\text{PDI}_j = \frac{\frac{1}{n}\sum_{i=1}^{n}\left[F(\mathbf{x}_i) - \hat{F}_{\setminus j}(\mathbf{x}_{i\setminus j})\right]^2}{\frac{1}{n}\sum_{i=1}^{n}\left[F(\mathbf{x}_i)\right]^2}.$$

It differs from H_j^2 only by not subtracting the main effect of the j-th feature in the numerator. It can be read as the proportion of prediction variability unexplained by all other features. As such, it measures variable importance of the j-th feature, including its interaction effects (check `partial_dep()` for all definitions).

Remarks 1 to 4 of `h2_overall()` also apply here.

Value

An object of class "hstats_matrix" containing these elements:

- `M`: Matrix of statistics (one column per prediction dimension), or `NULL`.
- `SE`: Matrix with standard errors of `M`, or `NULL`. Multiply with `sqrt(m_rep)` to get *standard deviations* instead. Currently, supported only for `perm_importance()`.
- `m_rep`: The number of repetitions behind standard errors `SE`, or `NULL`. Currently, supported only for `perm_importance()`.
- `statistic`: Name of the function that generated the statistic.
- `description`: Description of the statistic.

Methods (by class)

- `pd_importance(default)`: Default method of PD based feature importance.
- `pd_importance(hstats)`: PD based feature importance from "hstats" object.

References

Greenwell, Brandon M., Bradley C. Boehmke, and Andrew J. McCarthy. *A Simple and Effective Model-Based Variable Importance Measure.* Arxiv (2018).

See Also

`hstats()`, `perm_importance()`

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ . , data = iris)
s <- hstats(fit, X = iris[, -1])
plot(pd_importance(s))

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width + Species, data = iris)
s <- hstats(fit, X = iris[, 3:5])
plot(pd_importance(s))
```

---

perm_importance *Permutation Importance*

Description

Calculates permutation importance for a set of features or a set of feature groups. By default, importance is calculated for all columns in `X` (except column names used as response `y` or case weight `w`).

Usage

```
perm_importance(object, ...)

## Default S3 method:
perm_importance(
  object,
  X,
  y,
  v = NULL,
  pred_fun = stats::predict,
  loss = "squared_error",
  m_rep = 4L,
  agg_cols = FALSE,
  normalize = FALSE,
  n_max = 10000L,
  w = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'ranger'
perm_importance(
  object,
  X,
  y,
  v = NULL,
  pred_fun = function(m, X, ...) stats::predict(m, X, ...)$predictions,
  loss = "squared_error",
  m_rep = 4L,
  agg_cols = FALSE,
  normalize = FALSE,
  n_max = 10000L,
  w = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'explainer'
perm_importance(
  object,
  X = object[["data"]],
  y = object[["y"]],
  v = NULL,
  pred_fun = object[["predict_function"]],
  loss = "squared_error",
  m_rep = 4L,
  agg_cols = FALSE,
  normalize = FALSE,
```

```
  n_max = 10000L,
  w = object[["weights"]],
  verbose = TRUE,
  ...
)
```

Arguments

object	Fitted model object.
...	Additional arguments passed to pred_fun(object, X, ...), for instance type = "response" in a glm() model, or reshape = TRUE in a multiclass XGBoost model.
X	A data.frame or matrix serving as background dataset.
y	Vector/matrix of the response, or the corresponding column names in X.
v	Vector of feature names, or named list of feature groups. The default (NULL) will use all column names of X with the following exception: If y or w are passed as column names, they are dropped.
pred_fun	Prediction function of the form function(object, X, ...), providing $K \geq 1$ predictions per row. Its first argument represents the model object, its second argument a data structure like X. Additional arguments (such as type = "response" in a GLM, or reshape = TRUE in a multiclass XGBoost model) can be passed via .... The default, stats::predict(), will work in most cases.
loss	One of "squared_error", "logloss", "mlogloss", "poisson", "gamma", "absolute_error", "classification_error". Alternatively, a loss function can be provided that turns observed and predicted values into a numeric vector or matrix of unit losses of the same length as X. For "mlogloss", the response y can either be a dummy matrix or a discrete vector. The latter case is handled via a fast version of model.matrix(~ as.factor(y) + 0). For "classification_error", both predictions and responses can be non-numeric. For "squared_error", both predictions and responses can be factors with identical levels. In this case, squared error is evaulated for each one-hot-encoded column.
m_rep	Number of permutations (default 4).
agg_cols	Should multivariate losses be summed up? Default is FALSE. In combination with the squared error loss, agg_cols = TRUE gives the Brier score for (probabilistic) classification.
normalize	Should importance statistics be divided by average loss? Default is FALSE. If TRUE, an importance of 1 means that the average loss has been doubled by shuffling that feature's column.
n_max	If X has more than n_max rows, a random sample of n_max rows is selected from X. In this case, set a random seed for reproducibility.
w	Optional vector of case weights. Can also be a column name of X.
verbose	Should a progress bar be shown? The default is TRUE.

Details

The permutation importance of a feature is defined as the increase in the average loss when shuffling the corresponding feature values before calculating predictions. By default, the process is repeated `m_rep = 4` times, and the results are averaged. In most of the cases, importance values should be derived from an independent test data set. Set `normalize = TRUE` to get *relative* increases in average loss.

Value

An object of class "hstats_matrix" containing these elements:

- `M`: Matrix of statistics (one column per prediction dimension), or `NULL`.
- `SE`: Matrix with standard errors of `M`, or `NULL`. Multiply with `sqrt(m_rep)` to get *standard deviations* instead. Currently, supported only for `perm_importance()`.
- `m_rep`: The number of repetitions behind standard errors `SE`, or `NULL`. Currently, supported only for `perm_importance()`.
- `statistic`: Name of the function that generated the statistic.
- `description`: Description of the statistic.

Methods (by class)

- `perm_importance(default)`: Default method.
- `perm_importance(ranger)`: Method for "ranger" models.
- `perm_importance(explainer)`: Method for DALEX "explainer".

Losses

The default `loss` is the "squared_error". Other choices:

- "absolute_error": The absolute error is the loss corresponding to median regression.
- "poisson": Unit Poisson deviance, i.e., the loss function used in Poisson regression. Actual values `y` and predictions must be non-negative.
- "gamma": Unit gamma deviance, i.e., the loss function of Gamma regression. Actual values `y` and predictions must be positive.
- "logloss": The Log Loss is the loss function used in logistic regression, and the top choice in probabilistic binary classification. Responses `y` and predictions must be between 0 and 1. Predictions represent probabilities of having a "1".
- "mlogloss": Multi-Log-Loss is the natural loss function in probabilistic multi-class situations. If there are K classes and n observations, the predictions form a (n x K) matrix of probabilities (with row-sums 1). The observed values `y` are either passed as (n x K) dummy matrix, or as discrete vector with corresponding levels. The latter case is turned into a dummy matrix by a fast version of `model.matrix(~ as.factor(y) + 0)`.
- "classification_error": Misclassification error. Both the observed values `y` and the predictions can be character/factor. This loss function can be used in non-probabilistic classification settings. BUT: Probabilistic classification (with "mlogloss") is clearly preferred in most situations.
- A function with signature `f(actual, predicted)`, returning a numeric vector or matrix of the same length as the input.

References

Fisher A., Rudin C., Dominici F. (2018). All Models are Wrong but many are Useful: Variable Importance for Black-Box, Proprietary, or Misspecified Prediction Models, using Model Class Reliance. Arxiv.

Examples

```
# MODEL 1: Linear regression
fit <- lm(Sepal.Length ~ ., data = iris)
s <- perm_importance(fit, X = iris, y = "Sepal.Length")
s
s$M
s$SE  # Standard errors are available thanks to repeated shuffling
plot(s)
plot(s, err_type = "SD")  # Standard deviations instead of standard errors

# Groups of features can be passed as named list
v <- list(petal = c("Petal.Length", "Petal.Width"), species = "Species")
s <- perm_importance(fit, X = iris, y = "Sepal.Length", v = v, verbose = FALSE)
s
plot(s)

# MODEL 2: Multi-response linear regression
fit <- lm(as.matrix(iris[, 1:2]) ~ Petal.Length + Petal.Width + Species, data = iris)
s <- perm_importance(fit, X = iris[, 3:5], y = iris[, 1:2], normalize = TRUE)
s
plot(s)
plot(s, swap_dim = TRUE, top_m = 2)
```

---

plot.hstats *Plot Method for "hstats" Object*

Description

Plot method for object of class "hstats".

Usage

```
## S3 method for class 'hstats'
plot(
  x,
  which = 1:3,
  normalize = TRUE,
  squared = TRUE,
  sort = TRUE,
  top_m = 15L,
  zero = TRUE,
  fill = getOption("hstats.fill"),
```

```
  viridis_args = getOption("hstats.viridis_args"),
  facet_scales = "free",
  ncol = 2L,
  rotate_x = FALSE,
  ...
)
```

Arguments

x	Object of class "hstats".
which	Which statistic(s) to be shown? Default is 1:3, i.e., show H^2_j (1), H^2_{jk} (2), and H^2_{jkl} (3).
normalize	Should statistics be normalized? Default is TRUE.
squared	Should *squared* statistics be returned? Default is TRUE.
sort	Should results be sorted? Default is TRUE. (Multi-output is sorted by row means.)
top_m	How many rows should be plotted? Inf for all.
zero	Should rows with all 0 be shown? Default is TRUE.
fill	Fill color of ungrouped bars. The default equals the global option hstats.fill = "#fca50a". To change the global option, use options(stats.fill = new value).
viridis_args	List of viridis color scale arguments, see [ggplot2::scale_color_viridis_d()]. The default points to the global option hstats.viridis_args, which corresponds to list(begin = 0.2, end = 0.8, option = "B"). E.g., to switch to a standard viridis scale, you can change the default via options(hstats.viridis_args = list()), or set viridis_args = list().
facet_scales	Value passed as scales argument to [ggplot2::facet_wrap()].
ncol	Passed to [ggplot2::facet_wrap()].
rotate_x	Should x axis labels be rotated by 45 degrees?
...	Passed to ggplot2::geom_bar().

Value

An object of class "ggplot".

See Also

See hstats() for examples.

`plot.hstats_matrix` *Plots "hstats_matrix" Object*

Description

Plot method for objects of class "hstats_matrix".

Usage

```
## S3 method for class 'hstats_matrix'
plot(
  x,
  top_m = 15L,
  fill = getOption("hstats.fill"),
  swap_dim = FALSE,
  viridis_args = getOption("hstats.viridis_args"),
  facet_scales = "fixed",
  ncol = 2L,
  rotate_x = FALSE,
  err_type = c("SE", "SD", "No"),
  ...
)
```

Arguments

`x`	An object of class "hstats_matrix".
`top_m`	How many rows should be plotted? `Inf` for all.
`fill`	Fill color of ungrouped bars. The default equals the global option `hstats.fill = "#fca50a"`. To change the global option, use `options(stats.fill = new value)`.
`swap_dim`	Switches the role of grouping and facetting (default is `FALSE`).
`viridis_args`	List of viridis color scale arguments, see `[ggplot2::scale_color_viridis_d()]`. The default points to the global option `hstats.viridis_args`, which corresponds to `list(begin = 0.2, end = 0.8, option = "B")`. E.g., to switch to a standard viridis scale, you can change the default via `options(hstats.viridis_args = list())`, or set `viridis_args = list()`.
`facet_scales`	Value passed as `scales` argument to `[ggplot2::facet_wrap()]`.
`ncol`	Passed to `[ggplot2::facet_wrap()]`.
`rotate_x`	Should x axis labels be rotated by 45 degrees?
`err_type`	The error type to show, by default "SE" (standard errors). Set to "SD" for standard deviations (SE * sqrt(m_rep)), or "No" for no bars. Currently, supported only for `perm_importance()`.
`...`	Passed to `ggplot2::geom_bar()`.

Value

An object of class "ggplot".

plot.ice *Plots "ice" Object*

Description

Plot method for objects of class "ice".

Usage

```
## S3 method for class 'ice'
plot(
  x,
  center = FALSE,
  alpha = 0.2,
  color = getOption("hstats.color"),
  swap_dim = FALSE,
  viridis_args = getOption("hstats.viridis_args"),
  facet_scales = "fixed",
  rotate_x = FALSE,
  ...
)
```

Arguments

x	An object of class "ice".
center	Should curves be centered? Default is FALSE.
alpha	Transparency passed to ggplot2::geom_line().
color	Color of lines and points (in case there is no color/fill aesthetic). The default equals the global option hstats.color = "#3b528b". To change the global option, use options(stats.color = new value).
swap_dim	Swaps between color groups and facets. Default is FALSE.
viridis_args	List of viridis color scale arguments, see [ggplot2::scale_color_viridis_d()]. The default points to the global option hstats.viridis_args, which corresponds to list(begin = 0.2, end = 0.8, option = "B"). E.g., to switch to a standard viridis scale, you can change the default via options(hstats.viridis_args = list()), or set viridis_args = list().
facet_scales	Value passed as scales argument to [ggplot2::facet_wrap()].
rotate_x	Should x axis labels be rotated by 45 degrees?
...	Passed to ggplot2::geom_bar().

Value

An object of class "ggplot".

See Also

See ice() for examples.

---

plot.partial_dep *Plots "partial_dep" Object*

Description

Plot method for objects of class "partial_dep". Can do (grouped) line plots or heatmaps.

Usage

```
## S3 method for class 'partial_dep'
plot(
  x,
  color = getOption("hstats.color"),
  swap_dim = FALSE,
  viridis_args = getOption("hstats.viridis_args"),
  facet_scales = "fixed",
  rotate_x = FALSE,
  show_points = TRUE,
  d2_geom = c("tile", "point", "line"),
  ...
)
```

Arguments

x	An object of class "partial_dep".
color	Color of lines and points (in case there is no color/fill aesthetic). The default equals the global option hstats.color = "#3b528b". To change the global option, use options(stats.color = new value).
swap_dim	Switches the role of grouping and facetting (default is FALSE). Exception: For the 2D PDP with d2_geom = "line", it swaps the role of the two variables in v.
viridis_args	List of viridis color scale arguments, see [ggplot2::scale_color_viridis_d()]. The default points to the global option hstats.viridis_args, which corresponds to list(begin = 0.2, end = 0.8, option = "B"). E.g., to switch to a standard viridis scale, you can change the default via options(hstats.viridis_args = list()), or set viridis_args = list().
facet_scales	Value passed as scales argument to [ggplot2::facet_wrap()].
rotate_x	Should x axis labels be rotated by 45 degrees?
show_points	Logical flag indicating whether to show points (default) or not. No effect for 2D PDPs.
d2_geom	The geometry used for 2D PDPs, by default "tile". Option "point" is useful, e.g., when the grid represents spatial points. Option "line" produces lines grouped by the second variable.
...	Arguments passed to geometries.

Value

An object of class "ggplot".

See Also

See `partial_dep()` for examples.

---

print.hstats *Print Method*

Description

Print method for object of class "hstats". Shows H^2.

Usage

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

Arguments

`x`	An object of class "hstats".
`...`	Further arguments passed from other methods.

Value

Invisibly, the input is returned.

See Also

See `hstats()` for examples.

---

print.hstats_matrix *Prints "hstats_matrix" Object*

Description

Print method for object of class "hstats_matrix".

Usage

```
## S3 method for class 'hstats_matrix'
print(x, top_m = Inf, ...)
```

Arguments

x	An object of class "hstats_matrix".
top_m	Number of rows to print.
...	Currently not used.

Value

Invisibly, the input is returned.

---

print.hstats_summary *Print Method*

Description

Print method for object of class "hstats_summary".

Usage

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

Arguments

x	An object of class "hstats_summary".
...	Further arguments passed from other methods.

Value

Invisibly, the input is returned.

See Also

See hstats() for examples.

print.ice *Prints "ice" Object*

Description

Print method for object of class "ice".

Usage

```
## S3 method for class 'ice'
print(x, n = 3L, ...)
```

Arguments

x	An object of class "ice".
n	Number of rows to print.
...	Further arguments passed from other methods.

Value

Invisibly, the input is returned.

See Also

See `ice()` for examples.

print.partial_dep *Prints "partial_dep" Object*

Description

Print method for object of class "partial_dep".

Usage

```
## S3 method for class 'partial_dep'
print(x, n = 3L, ...)
```

Arguments

x	An object of class "partial_dep".
n	Number of rows to print.
...	Further arguments passed from other methods.

Value

Invisibly, the input is returned.

See Also

See `partial_dep()` for examples.

`summary.hstats` *Summary Method*

Description

Summary method for "hstats" object. Note that only the top 4 overall, the top 3 pairwise and the top 1 three-way statistics are shown.

Usage

```
## S3 method for class 'hstats'
summary(
  object,
  normalize = TRUE,
  squared = TRUE,
  sort = TRUE,
  zero = TRUE,
  ...
)
```

Arguments

`object`	Object of class "hstats".
`normalize`	Should statistics be normalized? Default is `TRUE`.
`squared`	Should *squared* statistics be returned? Default is `TRUE`.
`sort`	Should results be sorted? Default is `TRUE`. (Multi-output is sorted by row means.)
`zero`	Should rows with all 0 be shown? Default is `TRUE`.
`...`	Currently not used.

Value

An object of class "summary_hstats" representing a named list with statistics "h2", "h2_overall", "h2_pairwise", "h2_threeway", all of class "hstats_matrix".

See Also

See `hstats()` for examples.

`univariate_grid` *Univariate Grid*

Description

Creates evaluation grid for any numeric or non-numeric vector `z`.

For discrete `z` (non-numeric, or numeric with at most `grid_size` unique values), this is simply `sort(unique(z))`.

Otherwise, if `strategy = "uniform"` (default), the evaluation points form a regular grid over the trimmed range of `z`. By trimmed range we mean the range of `z` after removing values outside `trim[1]` and `trim[2]` quantiles. Set `trim = 0:1` for no trimming.

If `strategy = "quantile"`, the evaluation points are quantiles over a regular grid of probabilities from `trim[1]` to `trim[2]`.

Quantiles are calculated via the inverse of the ECDF, i.e., via `stats::quantile(..., type = 1)`.

Usage

```
univariate_grid(
  z,
  grid_size = 49L,
  trim = c(0.01, 0.99),
  strategy = c("uniform", "quantile"),
  na.rm = TRUE
)
```

Arguments

`z`	A vector or factor.
`grid_size`	Approximate grid size.
`trim`	The default `c(0.01, 0.99)` means that values outside the 1% and 99% quantiles of non-discrete numeric columns are removed before calculation of grid values. Set to `0:1` for no trimming.
`strategy`	How to find grid values of non-discrete numeric columns? Either "uniform" or "quantile", see description of `univariate_grid()`.
`na.rm`	Should missing values be dropped from the grid? Default is `TRUE`.

Value

A vector or factor of evaluation points.

See Also

`multivariate_grid()`

Examples

```
univariate_grid(iris$Species)
univariate_grid(rev(iris$Species))                      # Same

x <- iris$Sepal.Width
univariate_grid(x, grid_size = 5)                       # Uniform binning
univariate_grid(x, grid_size = 5, strategy = "quantile")  # Quantile
```

[.hstats_matrix *Subsets "hstats_matrix" Object*

Description

Use standard square bracket subsetting to select rows and/or columns of statistics "M" (and "SE" in case of permutation importance statistics). Implies `head()` and `tail()`.

Usage

```
## S3 method for class 'hstats_matrix'
x[i, j, ...]
```

Arguments

`x`	An object of class "hstats_matrix".
`i`	Row subsetting.
`j`	Column subsetting.
`...`	Currently unused.

Value

A new object of class "hstats_matrix".

Examples

```
fit <- lm(as.matrix(iris[1:2]) ~ Petal.Length + Petal.Width * Species, data = iris)
imp <- perm_importance(fit, X = iris, y = c("Sepal.Length", "Sepal.Width"))
head(imp, 1)
tail(imp, 2)
imp[1, "Sepal.Length"]
imp[1]
imp[, "Sepal.Width"]$SE
plot(imp[, "Sepal.Width"])
```

Index