Package 'vimp'

January 9, 2021

Type Package

Title Perform Inference on Algorithm-Agnostic Variable Importance

Version 2.1.6

Description Calculate point estimates of and valid confidence intervals for nonparametric, algorithm-agnostic variable importance measures in high and low dimensions, using flexible estimators of the underlying regression functions. For more information about the methods, please see Williamson et al. (Biometrics, 2020), Williamson et al. (arXiv, 2020+) <arXiv:2004.03683>, and Williamson and Feng (ICML, 2020).

Depends R (>= 3.1.0)

Imports SuperLearner, stats, dplyr, magrittr, ROCR, tibble, rlang, MASS

Suggests knitr, rmarkdown, gam, xgboost, glmnet, ranger, polspline, quadprog, covr, testthat, ggplot2, cowplot, RCurl

License MIT + file LICENSE

URL https://bdwilliamson.github.io/vimp/, https://github.com/bdwilliamson/vimp

BugReports https://github.com/bdwilliamson/vimp/issues

LazyData TRUE

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Brian D. Williamson [aut, cre] (<https://orcid.org/0000-0002-7024-548X>), Jean Feng [ctb], Noah Simon [ths] (<https://orcid.org/0000-0002-8985-2474>), Marco Carone [ths] (<https://orcid.org/0000-0003-2106-0953>)

Maintainer Brian D. Williamson <bwillia2@fredhutch.org>

Repository CRAN

Date/Publication 2021-01-09 02:20:02 UTC

R topics documented:

`average_vim` *Average multiple independent importance estimates*

Description

Average the output from multiple calls to `vimp_regression`, for different independent groups, into a single estimate with a corresponding standard error and confidence interval.

Usage

```
average_vim(..., weights = rep(1/length(list(...)), length(list(...))))
```

Arguments

...	an arbitrary number of vim objects.
weights	how to average the vims together, and must sum to 1; defaults to 1/(number of vims) for each vim, corresponding to the arithmetic mean

Value

an object of class vim containing the (weighted) average of the individual importance estimates, as well as the appropriate standard error and confidence interval. This results in a list containing:

- s - a list of the column(s) to calculate variable importance for
- SL.library - a list of the libraries of learners passed to SuperLearner
- full_fit - a list of the fitted values of the chosen method fit to the full data
- red_fit - a list of the fitted values of the chosen method fit to the reduced data
- est- a vector with the corrected estimates
- naive- a vector with the naive estimates
- update- a list with the influence curve-based updates
- mat - a matrix with the estimated variable importance, the standard error, and the $(1-\alpha)\times 100\%$ confidence interval
- full_mod - a list of the objects returned by the estimation procedure for the full data regression (if applicable)
- red_mod - a list of the objects returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- y - a list of the outcomes

Examples

```
# generate the data
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -5, 5)))

# apply the function to the x's
smooth <- (x[,1]/5)^2*(x[,1]+7)/5 + (x[,2]/3)^2

# generate Y ~ Normal (smooth, 1)
y <- smooth + stats::rnorm(n, 0, 1)

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# get estimates on independent splits of the data
samp <- sample(1:n, n/2, replace = FALSE)

# using Super Learner (with a small number of folds, for illustration only)
```

```
est_2 <- vimp_regression(Y = y[samp], X = x[samp, ], indx = 2, V = 2,
           run_regression = TRUE, alpha = 0.05,
           SL.library = learners, cvControl = list(V = 2))

est_1 <- vimp_regression(Y = y[-samp], X = x[-samp, ], indx = 2, V = 2,
           run_regression = TRUE, alpha = 0.05,
           SL.library = learners, cvControl = list(V = 2))

ests <- average_vim(est_1, est_2, weights = c(1/2, 1/2))
```

`cv_vim` *Nonparametric Variable Importance Estimates and Inference using Cross-fitting*

Description

Compute estimates and confidence intervals using cross-fitting for nonparametric variable importance based on the population-level contrast between the oracle predictiveness using the feature(s) of interest versus not.

Usage

```
cv_vim(
  Y = NULL,
  X = NULL,
  f1 = NULL,
  f2 = NULL,
  indx = 1,
  V = length(unique(folds)),
  folds = NULL,
  stratified = FALSE,
  type = "r_squared",
  run_regression = TRUE,
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  alpha = 0.05,
  delta = 0,
  scale = "identity",
  na.rm = FALSE,
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  ...
)
```

Arguments

Y	the outcome.
X	the covariates.
f1	the predicted values on validation data from a flexible estimation technique regressing Y on X in the training data; a list of length V, where each object is a set of predictions on the validation data.
f2	the predicted values on validation data from a flexible estimation technique regressing the fitted values in f1 on X withholding the columns in indx; a list of length V, where each object is a set of predictions on the validation data.
indx	the indices of the covariate(s) to calculate variable importance for; defaults to 1.
V	the number of folds for cross-fitting, defaults to 10.
folds	the folds to use, if f1 and f2 are supplied. A list of length two; the first element provides the outer folds (for hypothesis testing), while the second element is a list providing the inner folds (for cross-fitting).
stratified	if run_regression = TRUE, then should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-fitting folds)
type	the type of parameter (e.g., ANOVA-based is "anova").
run_regression	if outcome Y and covariates X are passed to cv_vim, and run_regression is TRUE, then Super Learner will be used; otherwise, variable importance will be computed using the inputted fitted values.
SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
scale	should CIs be computed on original ("identity") or logit ("logit") scale?
na.rm	should we remove NA's in the outcome and fitted values in computation? (defaults to FALSE)
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character vector specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings). Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
...	other arguments to the estimation tool, see "See also".

Details

We define the population variable importance measure (VIM) for the group of features (or single feature) s with respect to the predictiveness measure V by

$$\psi_{0,s} := V(f_0, P_0) - V(f_{0,s}, P_0),$$

where f_0 is the population predictiveness maximizing function, $f_{0,s}$ is the population predictiveness maximizing function that is only allowed to access the features with index not in s, and P_0 is the true data-generating distribution. Cross-fitted VIM estimates are obtained by first splitting the data into K folds; then using each fold in turn as a hold-out set, constructing estimators $f_{n,k}$ and $f_{n,k,s}$ of f_0 and $f_{0,s}$, respectively on the training data and estimator $P_{n,k}$ of P_0 using the test data; and finally, computing

$$\psi_{n,s} := K^{(-1)} \sum_{k=1}^{K} \{V(f_{n,k}, P_{n,k}) - V(f_{n,k,s}, P_{n,k})\}$$

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind the `cv_vim` function, and the validity of the confidence intervals.

In the interest of transparency, we return most of the calculations within the `vim` object. This results in a list including:

- s - the column(s) to calculate variable importance for
- SL.library - the library of learners passed to `SuperLearner`
- full_fit - the fitted values of the chosen method fit to the full data (a list, for train and test data)
- red_fit - the fitted values of the chosen method fit to the reduced data (a list, for train and test data)
- est - the estimated variable importance
- naive - the naive estimator of variable importance
- naives - the naive estimator on each fold
- eif- the estimated influence function
- all_eifs - the estimated influence curve for each fold
- se - the standard error for the estimated variable importance
- ci - the $(1 - \alpha) \times 100\%$ confidence interval for the variable importance estimate
- test - a decision to either reject (TRUE) or not reject (FALSE) the null hypothesis, based on a conservative test
- p_value - a p-value based on the same test as `test`
- full_mod - the object returned by the estimation procedure for the full data regression (if applicable)
- red_mod - the object returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- folds - the folds used for hypothesis testing and cross-fitting
- y - the outcome
- ipc_weights - the weights
- mat- a tibble with the estimate, SE, CI, hypothesis testing decision, and p-value

Value

An object of class `vim`. See Details for more information.

See Also

SuperLearner for specific usage of the SuperLearner function and package.

Examples

```
n <- 100
p <- 2
# generate the data
x <- data.frame(replicate(p, stats::runif(n, -5, 5)))

# apply the function to the x's
smooth <- (x[,1]/5)^2*(x[,1]+7)/5 + (x[,2]/3)^2

# generate Y ~ Normal (smooth, 1)
y <- as.matrix(smooth + stats::rnorm(n, 0, 1))

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# -----------------------------------------
# using Super Learner (with a small number of folds, for illustration only)
# -----------------------------------------
set.seed(4747)
est <- cv_vim(Y = y, X = x, indx = 2, V = 2,
type = "r_squared", run_regression = TRUE,
SL.library = learners, cvControl = list(V = 2), alpha = 0.05)

# -----------------------------------------
# doing things by hand, and plugging them in
# (with a small number of folds, for illustration only)
# -----------------------------------------
# set up the folds
indx <- 2
V <- 2
set.seed(4747)
outer_folds <- sample(rep(seq_len(2), length = n))
inner_folds_1 <- sample(rep(seq_len(V), length = sum(outer_folds == 1)))
inner_folds_2 <- sample(rep(seq_len(V), length = sum(outer_folds == 2)))
y_1 <- y[outer_folds == 1, , drop = FALSE]
x_1 <- x[outer_folds == 1, , drop = FALSE]
y_2 <- y[outer_folds == 2, , drop = FALSE]
x_2 <- x[outer_folds == 2, , drop = FALSE]
# get the fitted values by fitting the super learner on each pair
fhat_ful <- list()
fhat_red <- list()
for (v in 1:V) {
    # fit super learner
    fit <- SuperLearner::SuperLearner(Y = y_1[inner_folds_1 != v, , drop = FALSE],
     X = x_1[inner_folds_1 != v, , drop = FALSE],
     SL.library = learners, cvControl = list(V = V))
    fitted_v <- SuperLearner::predict.SuperLearner(fit)$pred
```

```
    # get predictions on the validation fold
    fhat_ful[[v]] <- SuperLearner::predict.SuperLearner(fit,
     newdata = x_1[inner_folds_1 == v, , drop = FALSE])$pred
    # fit the super learner on the reduced covariates
    red <- SuperLearner::SuperLearner(Y = y_2[inner_folds_2 != v, , drop = FALSE],
     X = x_2[inner_folds_2 != v, -indx, drop = FALSE],
     SL.library = learners, cvControl = list(V = V))
    # get predictions on the validation fold
    fhat_red[[v]] <- SuperLearner::predict.SuperLearner(red,
     newdata = x_2[inner_folds_2 == v, -indx, drop = FALSE])$pred
}
est <- cv_vim(Y = y, f1 = fhat_ful, f2 = fhat_red, indx = 2,
V = V, folds = list(outer_folds = outer_folds,
inner_folds = list(inner_folds_1, inner_folds_2)),
type = "r_squared", run_regression = FALSE, alpha = 0.05)
```

`est_predictiveness` *Estimate a nonparametric predictiveness functional*

Description

Compute nonparametric estimates of the chosen measure of predictiveness.

Usage

```
est_predictiveness(
  fitted_values,
  y,
  type = "r_squared",
  C = rep(1, length(y)),
  Z = NULL,
  ipc_weights = rep(1, length(C)),
  ipc_fit_type = "external",
  ipc_eif_preds = rep(1, length(C)),
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

`fitted_values`	fitted values from a regression function using the observed data.
`y`	the observed outcome.
`type`	which parameter are you estimating (defaults to `r_squared`, for R-squared-based variable importance)?
`C`	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).

Z	either NULL (if no coarsening) or a matrix-like object containing the fully observed data.
ipc_weights	weights for inverse probability of coarsening (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
ipc_fit_type	if "external", then use ipc_eif_preds; if "SL", fit a SuperLearner to determine the correction to the efficient influence function
ipc_eif_preds	if ipc_fit_type = "external", the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
scale	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
na.rm	logical; should NA's be removed in computation? (defaults to FALSE)
...	other arguments to SuperLearner, if ipc_fit_type = "SL".

Details

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind this function and the definition of the parameter of interest.

Value

A list, with: the estimated predictiveness; the estimated efficient influence function; and the predictions of the EIF based on inverse probability of censoring.

---

est_predictiveness_cv *Estimate a nonparametric predictiveness functional using cross-validation*

Description

Compute nonparametric estimates of the chosen measure of predictiveness.

Usage

```
est_predictiveness_cv(
  fitted_values,
  y,
  folds,
  type = "r_squared",
  C = rep(1, length(y)),
  Z = NULL,
  folds_Z = folds,
  ipc_weights = rep(1, length(C)),
  ipc_fit_type = "external",
  ipc_eif_preds = rep(1, length(C)),
```

```
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

fitted_values	fitted values from a regression function using the observed data; a list of length V, where each object is a set of predictions on the validation data.
y	the observed outcome.
folds	the cross-validation folds for the observed data
type	which parameter are you estimating (defaults to r_squared, for R-squared-based variable importance)?
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either NULL (if no coarsening) or a matrix-like object containing the fully observed data.
folds_Z	either the cross-validation folds for the observed data (no coarsening) or a vector of folds for the fully observed data Z.
ipc_weights	weights for inverse probability of coarsening (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
ipc_fit_type	if "external", then use ipc_eif_preds; if "SL", fit a SuperLearner to determine the correction to the efficient influence function
ipc_eif_preds	if ipc_fit_type = "external", the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
scale	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
na.rm	logical; should NA's be removed in computation? (defaults to FALSE)
...	other arguments to SuperLearner, if ipc_fit_type = "SL".

Details

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind this function and the definition of the parameter of interest.

Value

The estimated measure of predictiveness.

format.vim *Format a* vim *object*

Description

Nicely formats the output from a vim object for printing.

Usage

```
## S3 method for class 'vim'
format(x, ...)
```

Arguments

x	the vim object of interest.
...	other options, see the generic format function.

measure_accuracy *Estimate the classification accuracy*

Description

Compute nonparametric estimate of classification accuracy.

Usage

```
measure_accuracy(
  fitted_values,
  y,
  C = rep(1, length(y)),
  Z = NULL,
  ipc_weights = rep(1, length(y)),
  ipc_fit_type = "external",
  ipc_eif_preds = rep(1, length(y)),
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

fitted_values	fitted values from a regression function using the observed data.
y	the observed outcome.
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either NULL (if no coarsening) or a matrix-like object containing the fully observed data.
ipc_weights	weights for inverse probability of coarsening (IPC) (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
ipc_fit_type	if "external", then use ipc_eif_preds; if "SL", fit a SuperLearner to determine the IPC correction to the efficient influence function
ipc_eif_preds	if ipc_fit_type = "external", the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
scale	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
na.rm	logical; should NAs be removed in computation? (defaults to FALSE)
...	other arguments to SuperLearner, if ipc_fit_type = "SL".

Value

A named list of: (1) the estimated classification accuracy of the fitted regression function; (2) the estimated influence function; and (3) the IPC EIF predictions.

---

measure_anova *Estimate ANOVA decomposition-based variable importance.*

Description

Estimate ANOVA decomposition-based variable importance.

Usage

```
measure_anova(
  full,
  reduced,
  y,
  C = rep(1, length(y)),
  Z = NULL,
  ipc_weights = rep(1, length(y)),
  ipc_fit_type = "external",
  ipc_eif_preds = rep(1, length(y)),
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

full	fitted values from a regression function of the observed outcome on the full set of covariates.
reduced	fitted values from a regression on the reduced set of observed covariates.
y	the observed outcome.
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either NULL (if no coarsening) or a matrix-like object containing the fully observed data.
ipc_weights	weights for inverse probability of coarsening (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
ipc_fit_type	if "external", then use ipc_eif_preds; if "SL", fit a SuperLearner to determine the correction to the efficient influence function
ipc_eif_preds	if ipc_fit_type = "external", the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
scale	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
na.rm	logical; should NAs be removed in computation? (defaults to FALSE)
...	other arguments to SuperLearner, if ipc_fit_type = "SL".

Value

A named list of: (1) the estimated ANOVA (based on a one-step correction) of the fitted regression functions; (2) the estimated influence function; (3) the naive ANOVA estimate; and (4) the IPC EIF predictions.

---

measure_auc *Estimate area under the receiver operating characteristic curve (AUC)*

Description

Compute nonparametric estimate of AUC.

Usage

```
measure_auc(
  fitted_values,
  y,
  C = rep(1, length(y)),
  Z = NULL,
  ipc_weights = rep(1, length(y)),
  ipc_fit_type = "external",
```

```
  ipc_eif_preds = rep(1, length(y)),
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

fitted_values	fitted values from a regression function using the observed data.
y	the observed outcome.
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either NULL (if no coarsening) or a matrix-like object containing the fully observed data.
ipc_weights	weights for inverse probability of coarsening (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
ipc_fit_type	if "external", then use ipc_eif_preds; if "SL", fit a SuperLearner to determine the correction to the efficient influence function
ipc_eif_preds	if ipc_fit_type = "external", the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
scale	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
na.rm	logical; should NAs be removed in computation? (defaults to FALSE)
...	other arguments to SuperLearner, if ipc_fit_type = "SL".

Value

A named list of: (1) the estimated AUC of the fitted regression function; (2) the estimated influence function; and (3) the IPC EIF predictions.

---

measure_cross_entropy *Estimate the cross-entropy*

Description

Compute nonparametric estimate of cross-entropy.

Usage

```
measure_cross_entropy(
  fitted_values,
  y,
  C = rep(1, length(y)),
  Z = NULL,
```

```
  ipc_weights = rep(1, length(y)),
  ipc_fit_type = "external",
  ipc_eif_preds = rep(1, length(y)),
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

fitted_values	fitted values from a regression function using the observed data.
y	the observed outcome.
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either NULL (if no coarsening) or a matrix-like object containing the fully observed data.
ipc_weights	weights for inverse probability of coarsening (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
ipc_fit_type	if "external", then use ipc_eif_preds; if "SL", fit a SuperLearner to determine the correction to the efficient influence function
ipc_eif_preds	if ipc_fit_type = "external", the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
scale	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
na.rm	logical; should NAs be removed in computation? (defaults to FALSE)
...	other arguments to SuperLearner, if ipc_fit_type = "SL".

Value

A named list of: (1) the estimated cross-entropy of the fitted regression function; (2) the estimated influence function; and (3) the IPC EIF predictions.

---

measure_deviance *Estimate the deviance*

Description

Compute nonparametric estimate of deviance.

Usage

```
measure_deviance(
  fitted_values,
  y,
  C = rep(1, length(y)),
  Z = NULL,
  ipc_weights = rep(1, length(y)),
  ipc_fit_type = "external",
  ipc_eif_preds = rep(1, length(y)),
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

fitted_values	fitted values from a regression function using the observed data.
y	the opbserved outcome.
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either NULL (if no coarsening) or a matrix-like object containing the fully observed data.
ipc_weights	weights for inverse probability of coarsening (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
ipc_fit_type	if "external", then use ipc_eif_preds; if "SL", fit a SuperLearner to determine the correction to the efficient influence function
ipc_eif_preds	if ipc_fit_type = "external", the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
scale	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
na.rm	logical; should NAs be removed in computation? (defaults to FALSE)
...	other arguments to SuperLearner, if ipc_fit_type = "SL".

Value

A named list of: (1) the estimated deviance of the fitted regression function; (2) the estimated influence function; and (3) the IPC EIF predictions.

`measure_mse` *Estimate mean squared error*

Description

Compute nonparametric estimate of mean squared error.

Usage

```
measure_mse(
  fitted_values,
  y,
  C = rep(1, length(y)),
  Z = NULL,
  ipc_weights = rep(1, length(y)),
  ipc_fit_type = "external",
  ipc_eif_preds = rep(1, length(y)),
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

`fitted_values`	fitted values from a regression function using the observed data.
`y`	the observed outcome.
`C`	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
`Z`	either `NULL` (if no coarsening) or a matrix-like object containing the fully observed data.
`ipc_weights`	weights for inverse probability of coarsening (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
`ipc_fit_type`	if "external", then use `ipc_eif_preds`; if "SL", fit a SuperLearner to determine the correction to the efficient influence function
`ipc_eif_preds`	if `ipc_fit_type = "external"`, the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
`scale`	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
`na.rm`	logical; should `NA`s be removed in computation? (defaults to `FALSE`)
`...`	other arguments to SuperLearner, if `ipc_fit_type = "SL"`.

Value

A named list of: (1) the estimated mean squared error of the fitted regression function; (2) the estimated influence function; and (3) the IPC EIF predictions.

`measure_r_squared` *Estimate R-squared*

Description

Estimate R-squared

Usage

```
measure_r_squared(
  fitted_values,
  y,
  C = rep(1, length(y)),
  Z = NULL,
  ipc_weights = rep(1, length(y)),
  ipc_fit_type = "external",
  ipc_eif_preds = rep(1, length(y)),
  scale = "identity",
  na.rm = FALSE,
  ...
)
```

Arguments

`fitted_values`	fitted values from a regression function using the observed data.
`y`	the observed outcome.
`C`	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
`Z`	either `NULL` (if no coarsening) or a matrix-like object containing the fully observed data.
`ipc_weights`	weights for inverse probability of coarsening (e.g., inverse weights from a two-phase sample) weighted estimation. Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
`ipc_fit_type`	if "external", then use `ipc_eif_preds`; if "SL", fit a SuperLearner to determine the correction to the efficient influence function
`ipc_eif_preds`	if `ipc_fit_type = "external"`, the fitted values from a regression of the full-data EIF on the fully observed covariates/outcome; otherwise, not used.
`scale`	if doing an IPC correction, then the scale that the correction should be computed on (e.g., "identity"; or "logit" to logit-transform, apply the correction, and back-transform)
`na.rm`	logical; should `NA`s be removed in computation? (defaults to `FALSE`)
`...`	other arguments to SuperLearner, if `ipc_fit_type = "SL"`.

Value

A named list of: (1) the estimated R-squared of the fitted regression function; (2) the estimated influence function; and (3) the IPC EIF predictions.

`merge_vim` *Merge multiple* `vim` *objects into one*

Description

Take the output from multiple different calls to `vimp_regression` and merge into a single `vim` object; mostly used for plotting results.

Usage

```
merge_vim(...)
```

Arguments

`...`	an arbitrary number of `vim` objects, separated by commas.

Value

an object of class `vim` containing all of the output from the individual `vim` objects. This results in a list containing:

- s - a list of the column(s) to calculate variable importance for
- SL.library - a list of the libraries of learners passed to `SuperLearner`
- full_fit - a list of the fitted values of the chosen method fit to the full data
- red_fit - a list of the fitted values of the chosen method fit to the reduced data
- est- a vector with the corrected estimates
- naive- a vector with the naive estimates
- eif- a list with the influence curve-based updates
- se- a vector with the standard errors
- ci- a matrix with the CIs
- mat - a tibble with the estimated variable importance, the standard errors, and the $(1 - \alpha) \times 100\%$ confidence intervals
- full_mod - a list of the objects returned by the estimation procedure for the full data regression (if applicable)
- red_mod - a list of the objects returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - a list of the levels, for confidence interval calculation

Examples

```
# generate the data
# generate X
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -5, 5)))

# apply the function to the x's
smooth <- (x[,1]/5)^2*(x[,1]+7)/5 + (x[,2]/3)^2

# generate Y ~ Normal (smooth, 1)
y <- smooth + stats::rnorm(n, 0, 1)

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# using Super Learner (with a small number of folds, for illustration only)
est_2 <- vimp_regression(Y = y, X = x, indx = 2, V = 2,
           run_regression = TRUE, alpha = 0.05,
           SL.library = learners, cvControl = list(V = 2))

est_1 <- vimp_regression(Y = y, X = x, indx = 1, V = 2,
           run_regression = TRUE, alpha = 0.05,
           SL.library = learners, cvControl = list(V = 2))

ests <- merge_vim(est_1, est_2)
```

---

print.vim *Print a* vim *object*

Description

Prints out the table of estimates, confidence intervals, and standard errors for a vim object.

Usage

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

Arguments

x	the vim object of interest.
...	other options, see the generic print function.

sample_subsets *Create necessary objects for SPVIMs*

Description

Creates the Z and W matrices and a list of sampled subsets, S, for SPVIM estimation.

Usage

```
sample_subsets(p, gamma, n)
```

Arguments

p	the number of covariates
gamma	the fraction of the sample size to sample (e.g., gamma = 1 means sample n subsets)
n	the sample size

Value

a list, with elements Z (the matrix encoding presence/absence of each feature in the uniquely sampled subsets), S (the list of unique sampled subsets), W (the matrix of weights), and z_counts (the number of times each subset was sampled)

Examples

```
p <- 10
gamma <- 1
n <- 100
set.seed(100)
subset_lst <- sample_subsets(p, gamma, n)
```

spvim_ics *Influence function estimates for SPVIMs*

Description

Compute the influence functions for the contribution from sampling observations and subsets.

Usage

```
spvim_ics(Z, z_counts, W, v, psi, G, c_n, ics, measure)
```

Arguments

Z	the matrix of presence/absence of each feature (columns) in each sampled subset (rows)
z_counts	the number of times each unique subset was sampled
W	the matrix of weights
v	the estimated predictiveness measures
psi	the estimated SPVIM values
G	the constraint matrix
c_n	the constraint values
ics	a matrix of influence function values for each predictiveness measure
measure	the type of measure (e.g., "r_squared" or "auc")

Details

The processes for sampling observations and sampling subsets are independent. Thus, we can compute the influence function separately for each sampling process. For further details, see the paper by Williamson and Feng (2020).

Value

a named list of length 2; `contrib_v` is the contribution from estimating V, while `contrib_s` is the contribution from sampling subsets.

---

spvim_se *Standard error estimate for SPVIM values*

Description

Compute standard error estimates based on the estimated influence function for a SPVIM value of interest.

Usage

```
spvim_se(ics, idx = 1, gamma = 1, na_rm = FALSE)
```

Arguments

ics	the influence function estimates based on the contributions from sampling observations and sampling subsets: a list of length two resulting from a call to `spvim_ics`.
idx	the index of interest
gamma	the proportion of the sample size used when sampling subsets
na_rm	remove NAs?

Details

Since the processes for sampling observations and subsets are independent, the variance for a given SPVIM estimator is simply the sum of the vairances based on sampling observations and on sampling subsets.

Value

The standard error estimate for the desired SPVIM value

See Also

spvim_ics for how the influence functions are estimated.

---

sp_vim *Shapley Population Variable Importance Measure (SPVIM) Estimates and Inference*

Description

Compute estimates and confidence intervals for the SPVIMs, using cross-fitting.

Usage

```
sp_vim(
  Y = NULL,
  X = NULL,
  V = 5,
  type = "r_squared",
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  univariate_SL.library = NULL,
  gamma = 1,
  alpha = 0.05,
  delta = 0,
  na.rm = FALSE,
  stratified = FALSE,
  verbose = FALSE,
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  scale = "identity",
  ...
)
```

Arguments

Y	the outcome.
X	the covariates.
V	the number of folds for cross-fitting, defaults to 10.
type	the type of parameter (e.g., R-squared-based is "r_squared"). Note that type = 'anova' is not allowed for SPVIMs.
SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
univariate_SL.library	(optional) a character vector of learners to pass to SuperLearner for estimating univariate regression functions. Defaults to SL.polymars
gamma	the fraction of the sample size to use when sampling subsets (e.g., gamma = 1 samples the same number of subsets as the sample size)
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
na.rm	should we remove NA's in the outcome and fitted values in computation? (defaults to FALSE)
stratified	should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-fitting folds)?
verbose	should sp_vim and SuperLearner print out progress? (defaults to FALSE)
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character vector specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings). Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
scale	should CIs be computed on original ("identity") or logit ("logit") scale?
...	other arguments to the estimation tool, see "See also".

Details

We define the SPVIM as the weighted average of the population difference in predictiveness over all subsets of features not containing feature j.

This is equivalent to finding the solution to a population weighted least squares problem. This key fact allows us to estimate the SPVIM using weighted least squares, where we first sample subsets from the power set of all possible features using the Shapley sampling distribution; then use cross-fitting to obtain estimators of the predictiveness of each sampled subset; and finally, solve the least squares problem given in Williamson and Feng (2020).

See the paper by Williamson and Feng (2020) for more details on the mathematics behind this function, and the validity of the confidence intervals. The function works by estimating In the interest of transparency, we return most of the calculations within the vim object. This results in a list containing:

- SL.library - the library of learners passed to SuperLearner
- v- the estimated predictiveness measure for each sampled subset
- preds_lst - the predicted values from the chosen method for each sampled subset
- est - the estimated SPVIM value for each feature
- ic_lst - the influence functions for each sampled subset
- ic- a list of the SPVIM influence function contributions
- se - the standard errors for the estimated variable importance
- ci - the $(1-\alpha) \times 100\%$ confidence intervals based on the variable importance estimates
- gamma- the fraction of the sample size used when sampling subsets
- alpha - the level, for confidence interval calculation
- delta- the delta value used for hypothesis testing
- y - the outcome
- ipc_weights - the weights
- mat- a tibble with the estimates, SEs, CIs, hypothesis testing decisions, and p-values

Value

An object of class vim. See Details for more information.

See Also

SuperLearner for specific usage of the SuperLearner function and package.

Examples

```
n <- 100
p <- 2
# generate the data
x <- data.frame(replicate(p, stats::runif(n, -5, 5)))

# apply the function to the x's
smooth <- (x[,1]/5)^2*(x[,1]+7)/5 + (x[,2]/3)^2

# generate Y ~ Normal (smooth, 1)
y <- as.matrix(smooth + stats::rnorm(n, 0, 1))

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# ----------------------------------------
# using Super Learner (with a small number of CV folds,
# for illustration only)
# ----------------------------------------
set.seed(4747)
est <- sp_vim(Y = y, X = x, V = 2, type = "r_squared",
SL.library = learners, alpha = 0.05)
```

vim *Nonparametric Variable Importance Estimates and Inference*

Description

Compute estimates of and confidence intervals for nonparametric variable importance based on the population-level contrast between the oracle predictiveness using the feature(s) of interest versus not.

Usage

```
vim(
  Y = NULL,
  X = NULL,
  f1 = NULL,
  f2 = NULL,
  indx = 1,
  type = "r_squared",
  run_regression = TRUE,
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  alpha = 0.05,
  delta = 0,
  scale = "identity",
  na.rm = FALSE,
  folds = NULL,
  stratified = FALSE,
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  ...
)
```

Arguments

Y	the outcome.
X	the covariates.
f1	the fitted values from a flexible estimation technique regressing Y on X.
f2	the fitted values from a flexible estimation technique regressing Y on X withholding the columns in indx.
indx	the indices of the covariate(s) to calculate variable importance for; defaults to 1.
type	the type of importance to compute; defaults to r_squared, but other supported options are auc, accuracy, deviance, and anova.
run_regression	if outcome Y and covariates X are passed to vimp_accuracy, and run_regression is TRUE, then Super Learner will be used; otherwise, variable importance will be computed using the inputted fitted values.

SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
scale	should CIs be computed on original ("identity") or logit ("logit") scale?
na.rm	should we remove NAs in the outcome and fitted values in computation? (defaults to FALSE)
folds	the folds used for f1 and f2; assumed to be 1 for the observations used in f1 and 2 for the observations used in f2. If there is only a single fold passed in, then hypothesis testing is not done.
stratified	if run_regression = TRUE, then should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-validation folds)
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character vector specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings). Assumed to be already inverted (i.e., ipc_weights = 1 / [estimated probability weights]).
...	other arguments to the estimation tool, see "See also".

Details

We define the population variable importance measure (VIM) for the group of features (or single feature) s with respect to the predictiveness measure V by

$$\psi_{0,s} := V(f_0, P_0) - V(f_{0,s}, P_0),$$

where f_0 is the population predictiveness maximizing function, $f_{0,s}$ is the population predictiveness maximizing function that is only allowed to access the features with index not in s, and P_0 is the true data-generating distribution. VIM estimates are obtained by obtaining estimators f_n and $f_{n,s}$ of f_0 and $f_{0,s}$, respectively; obtaining an estimator P_n of P_0; and finally, setting $\psi_{n,s} := V(f_n, P_n) - V(f_{n,s}, P_n)$.

In the interest of transparency, we return most of the calculations within the vim object. This results in a list including:

- s - the column(s) to calculate variable importance for
- SL.library - the library of learners passed to SuperLearner
- type - the type of risk-based variable importance measured
- full_fit - the fitted values of the chosen method fit to the full data
- red_fit - the fitted values of the chosen method fit to the reduced data
- est - the estimated variable importance
- naive - the naive estimator of variable importance (only used if type = "anova")

- eif - the estimated influence curve
- se - the standard error for the estimated variable importance
- ci - the $(1 - \alpha) \times 100\%$ confidence interval for the variable importance estimate
- test - a decision to either reject (TRUE) or not reject (FALSE) the null hypothesis, based on a conservative test
- p_value - a p-value based on the same test as `test`
- full_mod - the object returned by the estimation procedure for the full data regression (if applicable)
- red_mod - the object returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- folds - the folds used for hypothesis testing
- y - the outcome
- ipc_weights - the weights
- mat- a tibble with the estimate, SE, CI, hypothesis testing decision, and p-value

Value

An object of classes `vim` and the type of risk-based measure. See Details for more information.

See Also

`SuperLearner` for specific usage of the `SuperLearner` function and package.

Examples

```
# generate the data
# generate X
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -1, 1)))

# apply the function to the x's
f <- function(x) 0.5 + 0.3*x[1] + 0.2*x[2]
smooth <- apply(x, 1, function(z) f(z))

# generate Y ~ Normal (smooth, 1)
y <- matrix(rbinom(n, size = 1, prob = smooth))

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# using Y and X; use class-balanced folds
folds_1 <- sample(rep(seq_len(2), length = sum(y == 1)))
folds_0 <- sample(rep(seq_len(2), length = sum(y == 0)))
folds <- vector("numeric", length(y))
folds[y == 1] <- folds_1
```

```
folds[y == 0] <- folds_0
est <- vim(y, x, indx = 2, type = "r_squared",
           alpha = 0.05, run_regression = TRUE,
           SL.library = learners, cvControl = list(V = 2),
           folds = folds)

# using pre-computed fitted values
full <- SuperLearner::SuperLearner(Y = y[folds == 1], X = x[folds == 1, ],
SL.library = learners, cvControl = list(V = 2))
full.fit <- SuperLearner::predict.SuperLearner(full)$pred
reduced <- SuperLearner::SuperLearner(Y = y[folds == 2],
X = x[folds == 2, -2, drop = FALSE],
SL.library = learners, cvControl = list(V = 2))
red.fit <- SuperLearner::predict.SuperLearner(reduced)$pred

est <- vim(Y = y, f1 = full.fit, f2 = red.fit,
            indx = 2, run_regression = FALSE, alpha = 0.05, folds = folds,
            type = "accuracy")
```

`vimp` *vimp: Perform Inference on Algorithm-Agnostic Variable Importance*

Description

A unified framework for valid statistical inference on algorithm-agnostic measures of variable importance. You provide the data, a method for estimating the conditional mean of the outcome given the covariates, choose a variable importance measure, and specify variable(s) of interest; 'vimp' takes care of the rest.

Author(s)

Maintainer: Brian Williamson https://bdwilliamson.github.io/

Methodology authors:

- Brian D. Williamson
- Jean Feng
- Peter B. Gilbert
- Noah R. Simon
- Marco Carone

See Also

Manuscripts:

- doi: 10.1111/biom.13392 (R-squared-based variable importance)
- doi: 10.1111/biom.13389 (Rejoinder to discussion on R-squared-based variable importance article)

- http://proceedings.mlr.press/v119/williamson20a.html (general Shapley-based variable importance)

Preprints:

- https://arxiv.org/abs/2004.03683 (general variable importance)

Other useful links:

- https://bdwilliamson.github.io/vimp/
- https://github.com/bdwilliamson/vimp
- Report bugs at https://github.com/bdwilliamson/vimp/issues

Imports

The packages that we import either make the internal code nice (dplyr, magrittr, tibble, rlang, MASS), are directly relevant to estimating the conditional mean (SuperLearner) or predictiveness measures (ROCR), or are necessary for hypothesis testing (stats).

We suggest several other packages: xgboost, ranger, gam, glmnet, polspline, and quadprog allow a flexible library of candidate learners in the Super Learner; ggplot2, cowplot, and forcats help with plotting variable importance estimates; testthat and covr help with unit tests; and knitr, rmarkdown, and RCurl help with the vignettes and examples.

`vimp_accuracy` *Nonparametric Variable Importance Estimates: Classification accuracy*

Description

Compute estimates of and confidence intervals for nonparametric difference in classification accuracy-based variable importance. This is a wrapper function for `cv_vim`, with `type = "accuracy"`.

Usage

```
vimp_accuracy(
  Y = NULL,
  X = NULL,
  f1 = NULL,
  f2 = NULL,
  indx = 1,
  V = 10,
  run_regression = TRUE,
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  alpha = 0.05,
  delta = 0,
  na.rm = FALSE,
  folds = NULL,
  stratified = TRUE,
```

```
  scale = "identity",
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  ...
)
```

Arguments

Y	the outcome.
X	the covariates.
f1	the predicted values on validation data from a flexible estimation technique regressing Y on X in the training data; a list of length V, where each object is a set of predictions on the validation data.
f2	the predicted values on validation data from a flexible estimation technique regressing the fitted values in f1 on X withholding the columns in indx; a list of length V, where each object is a set of predictions on the validation data.
indx	the indices of the covariate(s) to calculate variable importance for; defaults to 1.
V	the number of folds for cross-validation, defaults to 10.
run_regression	if outcome Y and covariates X are passed to cv_vim, and run_regression is TRUE, then Super Learner will be used; otherwise, variable importance will be computed using the inputted fitted values.
SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
na.rm	should we remove NA's in the outcome and fitted values in computation? (defaults to FALSE)
folds	the folds to use, if f1 and f2 are supplied.
stratified	if run_regression = TRUE, then should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-validation folds)
scale	scale should CIs be computed on original ("identity") or logit ("logit") scale? (defaults to "identity")
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character list specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings)
...	other arguments to the estimation tool, see "See also".

Details

In the interest of transparency, we return most of the calculations within the `vim` object. This results in a list containing:

- s - the column(s) to calculate variable importance for
- SL.library - the library of learners passed to `SuperLearner`
- full_fit - the fitted values of the chosen method fit to the full data
- red_fit - the fitted values of the chosen method fit to the reduced data
- est - the estimated variable importance
- naive - the naive estimator of variable importance
- update - the influence curve-based update
- se - the standard error for the estimated variable importance
- ci - the $(1 - \alpha) \times 100\%$ confidence interval for the variable importance estimate
- full_mod - the object returned by the estimation procedure for the full data regression (if applicable)
- red_mod - the object returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- y - the outcome

Value

An object of classes `vim` and `vim_accuracy`. See Details for more information.

See Also

`SuperLearner` for specific usage of the `SuperLearner` function and package.

Examples

```
# generate the data
# generate X
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -1, 1)))

# apply the function to the x's
f <- function(x) 0.5 + 0.3*x[1] + 0.2*x[2]
smooth <- apply(x, 1, function(z) f(z))

# generate Y ~ Normal (smooth, 1)
y <- matrix(rbinom(n, size = 1, prob = smooth))

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")
```

```
# estimate (with a small number of folds, for illustration only)
est <- vimp_accuracy(y, x, indx = 2,
           alpha = 0.05, run_regression = TRUE,
           SL.library = learners, V = 2, cvControl = list(V = 2))
```

---

vimp_anova *Nonparametric Variable Importance Estimates: ANOVA*

Description

Compute estimates of and confidence intervals for nonparametric difference in classification accuracy-based variable importance. This is a wrapper function for cv_vim, with type = "anova".

Usage

```
vimp_anova(
  Y = NULL,
  X = NULL,
  f1 = NULL,
  f2 = NULL,
  indx = 1,
  V = 10,
  run_regression = TRUE,
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  alpha = 0.05,
  delta = 0,
  na.rm = FALSE,
  scale = "identity",
  folds,
  stratified = FALSE,
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  ...
)
```

Arguments

Y	the outcome.
X	the covariates.
f1	the predicted values on validation data from a flexible estimation technique regressing Y on X in the training data; a list of length V, where each object is a set of predictions on the validation data.
f2	the predicted values on validation data from a flexible estimation technique regressing the fitted values in f1 on X withholding the columns in indx; a list of length V, where each object is a set of predictions on the validation data.

indx	the indices of the covariate(s) to calculate variable importance for; defaults to 1.
V	the number of folds for cross-validation, defaults to 10.
run_regression	if outcome Y and covariates X are passed to cv_vim, and run_regression is TRUE, then Super Learner will be used; otherwise, variable importance will be computed using the inputted fitted values.
SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
na.rm	should we remove NA's in the outcome and fitted values in computation? (defaults to FALSE)
scale	scale should CIs be computed on original ("identity") or logit ("logit") scale? (defaults to "identity")
folds	the folds to use, if f1 and f2 are supplied.
stratified	if run_regression = TRUE, then should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-validation folds)
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character list specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings)
...	other arguments to the estimation tool, see "See also".

Details

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind this function, and the validity of the confidence intervals. In the interest of transparency, we return most of the calculations within the vim object. This results in a list containing:

- s - the column(s) to calculate variable importance for
- SL.library - the library of learners passed to SuperLearner
- full_fit - the fitted values of the chosen method fit to the full data
- red_fit - the fitted values of the chosen method fit to the reduced data
- est - the estimated variable importance
- naive - the naive estimator of variable importance
- update - the influence curve-based update
- se - the standard error for the estimated variable importance
- ci - the $(1 - \alpha) \times 100\%$ confidence interval for the variable importance estimate
- full_mod - the object returned by the estimation procedure for the full data regression (if applicable)

- red_mod - the object returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- y - the outcome

Value

An object of classes `vim` and `vim_regression`. See Details for more information.

See Also

`SuperLearner` for specific usage of the `SuperLearner` function and package.

Examples

```
# generate the data
# generate X
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -5, 5)))

# apply the function to the x's
smooth <- (x[,1]/5)^2*(x[,1]+7)/5 + (x[,2]/3)^2

# generate Y ~ Normal (smooth, 1)
y <- smooth + stats::rnorm(n, 0, 1)

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# estimate (with a small number of folds, for illustration only)
est <- vimp_anova(y, x, indx = 2,
           alpha = 0.05, run_regression = TRUE,
           SL.library = learners, V = 2, cvControl = list(V = 2))
```

`vimp_auc` *Nonparametric Variable Importance Estimates: AUC*

Description

Compute estimates of and confidence intervals for nonparametric difference in AUC-based variable importance. This is a wrapper function for `cv_vim`, with `type = "auc"`.

Usage

```
vimp_auc(
  Y = NULL,
  X = NULL,
  f1 = NULL,
  f2 = NULL,
  indx = 1,
  V = 10,
  run_regression = TRUE,
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  alpha = 0.05,
  delta = 0,
  na.rm = FALSE,
  folds = NULL,
  stratified = TRUE,
  scale = "identity",
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  ...
)
```

Arguments

Y	the outcome.
X	the covariates.
f1	the predicted values on validation data from a flexible estimation technique regressing Y on X in the training data; a list of length V, where each object is a set of predictions on the validation data.
f2	the predicted values on validation data from a flexible estimation technique regressing the fitted values in f1 on X withholding the columns in indx; a list of length V, where each object is a set of predictions on the validation data.
indx	the indices of the covariate(s) to calculate variable importance for; defaults to 1.
V	the number of folds for cross-validation, defaults to 10.
run_regression	if outcome Y and covariates X are passed to cv_vim, and run_regression is TRUE, then Super Learner will be used; otherwise, variable importance will be computed using the inputted fitted values.
SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
na.rm	should we remove NA's in the outcome and fitted values in computation? (defaults to FALSE)
folds	the folds to use, if f1 and f2 are supplied.

stratified	if run_regression = TRUE, then should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-validation folds)
scale	scale should CIs be computed on original ("identity") or logit ("logit") scale? (defaults to "identity")
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character list specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings)
...	other arguments to the estimation tool, see "See also".

Details

AUC for each regression (full and reduced) is computed using performance. In the interest of transparency, we return most of the calculations within the vim object. This results in a list containing:

- s - the column(s) to calculate variable importance for
- SL.library - the library of learners passed to SuperLearner
- full_fit - the fitted values of the chosen method fit to the full data
- red_fit - the fitted values of the chosen method fit to the reduced data
- est - the estimated variable importance
- naive - the naive estimator of variable importance
- update - the influence curve-based update
- se - the standard error for the estimated variable importance
- ci - the $(1-\alpha) \times 100\%$ confidence interval for the variable importance estimate
- full_mod - the object returned by the estimation procedure for the full data regression (if applicable)
- red_mod - the object returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- y - the outcome

Value

An object of classes vim and vim_auc. See Details for more information.

See Also

SuperLearner for specific usage of the SuperLearner function and package, and performance for specific usage of the ROCR package.

Examples

```
# generate the data
# generate X
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -1, 1)))

# apply the function to the x's
f <- function(x) 0.5 + 0.3*x[1] + 0.2*x[2]
smooth <- apply(x, 1, function(z) f(z))

# generate Y ~ Normal (smooth, 1)
y <- matrix(rbinom(n, size = 1, prob = smooth))

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# estimate (with a small number of folds, for illustration only)
est <- vimp_auc(y, x, indx = 2,
           alpha = 0.05, run_regression = TRUE,
           SL.library = learners, V = 2, cvControl = list(V = 2))
```

---

vimp_ci *Confidence intervals for variable importance*

Description

Compute confidence intervals for the true variable importance parameter.

Usage

```
vimp_ci(est, se, scale = "identity", level = 0.95)
```

Arguments

est	estimate of variable importance, e.g., from a call to vimp_point_est.
se	estimate of the standard error of est, e.g., from a call to vimp_se.
scale	scale to compute interval estimate on (defaults to "identity": compute Wald-type CI).
level	confidence interval type (defaults to 0.95).

Details

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind this function and the definition of the parameter of interest.

Value

The Wald-based confidence interval for the true importance of the given group of left-out covariates.

`vimp_deviance` *Nonparametric Variable Importance Estimates: Deviance*

Description

Compute estimates of and confidence intervals for nonparametric deviance-based variable importance. This is a wrapper function for `cv_vim`, with `type = "deviance"`.

Usage

```
vimp_deviance(
  Y = NULL,
  X = NULL,
  f1 = NULL,
  f2 = NULL,
  indx = 1,
  V = 10,
  run_regression = TRUE,
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  alpha = 0.05,
  delta = 0,
  na.rm = FALSE,
  folds = NULL,
  stratified = TRUE,
  scale = "identity",
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  ...
)
```

Arguments

`Y`	the outcome.
`X`	the covariates.
`f1`	the predicted values on validation data from a flexible estimation technique regressing Y on X in the training data; a list of length V, where each object is a set of predictions on the validation data.
`f2`	the predicted values on validation data from a flexible estimation technique regressing the fitted values in `f1` on X withholding the columns in `indx`; a list of length V, where each object is a set of predictions on the validation data.
`indx`	the indices of the covariate(s) to calculate variable importance for; defaults to 1.

V	the number of folds for cross-validation, defaults to 10.
run_regression	if outcome Y and covariates X are passed to cv_vim, and run_regression is TRUE, then Super Learner will be used; otherwise, variable importance will be computed using the inputted fitted values.
SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
na.rm	should we remove NA's in the outcome and fitted values in computation? (defaults to FALSE)
folds	the folds to use, if f1 and f2 are supplied.
stratified	if run_regression = TRUE, then should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-validation folds)
scale	scale should CIs be computed on original ("identity") or logit ("logit") scale? (defaults to "identity")
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character list specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings)
...	other arguments to the estimation tool, see "See also".

Details

In the interest of transparency, we return most of the calculations within the vim object. This results in a list containing:

- s - the column(s) to calculate variable importance for
- SL.library - the library of learners passed to SuperLearner
- full_fit - the fitted values of the chosen method fit to the full data
- red_fit - the fitted values of the chosen method fit to the reduced data
- est - the estimated variable importance
- naive - the naive estimator of variable importance
- update - the influence curve-based update
- se - the standard error for the estimated variable importance
- ci - the $(1 - \alpha) \times 100\%$ confidence interval for the variable importance estimate
- full_mod - the object returned by the estimation procedure for the full data regression (if applicable)
- red_mod - the object returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- y - the outcome

Value

An object of classes vim and vim_deviance. See Details for more information.

See Also

SuperLearner for specific usage of the SuperLearner function and package.

Examples

```
# generate the data
# generate X
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -1, 1)))

# apply the function to the x's
f <- function(x) 0.5 + 0.3*x[1] + 0.2*x[2]
smooth <- apply(x, 1, function(z) f(z))

# generate Y ~ Normal (smooth, 1)
y <- matrix(stats::rbinom(n, size = 1, prob = smooth))

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# estimate (with a small number of folds, for illustration only)
est <- vimp_deviance(y, x, indx = 2,
           alpha = 0.05, run_regression = TRUE,
           SL.library = learners, V = 2, cvControl = list(V = 2))
```

---

vimp_hypothesis_test *Perform a hypothesis test against the null hypothesis of δ importance*

Description

Perform a hypothesis test against the null hypothesis of zero importance by: (i) for a user-specified level α, compute a $(1-\alpha) \times 100\%$ confidence interval around the predictiveness for both the full and reduced regression functions (these must be estimated on independent splits of the data); (ii) if the intervals do not overlap, reject the null hypothesis.

Usage

```
vimp_hypothesis_test(
  predictiveness_full,
  predictiveness_reduced,
  se_full,
  se_reduced,
```

```
  delta = 0,
  alpha = 0.05
)
```

Arguments

predictiveness_full	the estimated predictiveness of the regression including the covariate(s) of interest.
predictiveness_reduced	the estimated predictiveness of the regression excluding the covariate(s) of interest.
se_full	the estimated standard error for the predictiveness of the regression including the covariate(s) of interest.
se_reduced	the estimated standard error for the predictiveness of the regression excluding the covariate(s) of interest.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
alpha	the desired type I error rate (defaults to 0.05).

Details

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind this function and the definition of the parameter of interest.

Value

a list, with: the hypothesis testing decision (TRUE if the null hypothesis is rejected, FALSE otherwise); the p-value from the hypothesis test; and the test statistic from the hypothesis test.

---

vimp_regression *Nonparametric Variable Importance Estimates*

Description

Compute estimates of and confidence intervals for nonparametric ANOVA-based variable importance. This is a wrapper function for cv_vim, with type = "anova". This function is deprecated in vimp version 2.0.0.

Usage

```
vimp_regression(
  Y = NULL,
  X = NULL,
  f1 = NULL,
  f2 = NULL,
  indx = 1,
```

```
  V = 10,
  run_regression = TRUE,
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  alpha = 0.05,
  delta = 0,
  na.rm = FALSE,
  folds,
  stratified = FALSE,
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  ...
)
```

Arguments

Y	the outcome.
X	the covariates.
f1	the predicted values on validation data from a flexible estimation technique regressing Y on X in the training data; a list of length V, where each object is a set of predictions on the validation data.
f2	the predicted values on validation data from a flexible estimation technique regressing the fitted values in f1 on X withholding the columns in indx; a list of length V, where each object is a set of predictions on the validation data.
indx	the indices of the covariate(s) to calculate variable importance for; defaults to 1.
V	the number of folds for cross-validation, defaults to 10.
run_regression	if outcome Y and covariates X are passed to cv_vim, and run_regression is TRUE, then Super Learner will be used; otherwise, variable importance will be computed using the inputted fitted values.
SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
na.rm	should we remove NA's in the outcome and fitted values in computation? (defaults to FALSE)
folds	the folds to use, if f1 and f2 are supplied.
stratified	if run_regression = TRUE, then should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-validation folds)
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character list specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings)
...	other arguments to the estimation tool, see "See also".

Details

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind this function, and the validity of the confidence intervals. In the interest of transparency, we return most of the calculations within the `vim` object. This results in a list containing:

- s - the column(s) to calculate variable importance for
- SL.library - the library of learners passed to `SuperLearner`
- full_fit - the fitted values of the chosen method fit to the full data
- red_fit - the fitted values of the chosen method fit to the reduced data
- est - the estimated variable importance
- naive - the naive estimator of variable importance
- update - the influence curve-based update
- se - the standard error for the estimated variable importance
- ci - the $(1 - \alpha) \times 100\%$ confidence interval for the variable importance estimate
- full_mod - the object returned by the estimation procedure for the full data regression (if applicable)
- red_mod - the object returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- y - the outcome

Value

An object of classes `vim` and `vim_regression`. See Details for more information.

See Also

`SuperLearner` for specific usage of the `SuperLearner` function and package.

Examples

```
# generate the data
# generate X
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -5, 5)))

# apply the function to the x's
smooth <- (x[,1]/5)^2*(x[,1]+7)/5 + (x[,2]/3)^2

# generate Y ~ Normal (smooth, 1)
y <- smooth + stats::rnorm(n, 0, 1)

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")
```

```
# estimate (with a small number of folds, for illustration only)
est <- vimp_regression(y, x, indx = 2,
           alpha = 0.05, run_regression = TRUE,
           SL.library = learners, V = 2, cvControl = list(V = 2))
```

---

vimp_rsquared *Nonparametric Variable Importance Estimates: R-squared*

Description

Compute estimates of and confidence intervals for nonparametric R^2-based variable importance. This is a wrapper function for cv_vim, with type = "r_squared".

Usage

```
vimp_rsquared(
  Y = NULL,
  X = NULL,
  f1 = NULL,
  f2 = NULL,
  indx = 1,
  V = 10,
  run_regression = TRUE,
  SL.library = c("SL.glmnet", "SL.xgboost", "SL.mean"),
  alpha = 0.05,
  delta = 0,
  na.rm = FALSE,
  folds = NULL,
  stratified = FALSE,
  C = rep(1, length(Y)),
  Z = NULL,
  ipc_weights = rep(1, length(Y)),
  ...
)
```

Arguments

Y	the outcome.
X	the covariates.
f1	the predicted values on validation data from a flexible estimation technique regressing Y on X in the training data; a list of length V, where each object is a set of predictions on the validation data.
f2	the predicted values on validation data from a flexible estimation technique regressing the fitted values in f1 on X withholding the columns in indx; a list of length V, where each object is a set of predictions on the validation data.

indx	the indices of the covariate(s) to calculate variable importance for; defaults to 1.
V	the number of folds for cross-validation, defaults to 10.
run_regression	if outcome Y and covariates X are passed to cv_vim, and run_regression is TRUE, then Super Learner will be used; otherwise, variable importance will be computed using the inputted fitted values.
SL.library	a character vector of learners to pass to SuperLearner, if f1 and f2 are Y and X, respectively. Defaults to SL.glmnet, SL.xgboost, and SL.mean.
alpha	the level to compute the confidence interval at. Defaults to 0.05, corresponding to a 95% confidence interval.
delta	the value of the δ-null (i.e., testing if importance < δ); defaults to 0.
na.rm	should we remove NA's in the outcome and fitted values in computation? (defaults to FALSE)
folds	the folds to use, if f1 and f2 are supplied.
stratified	if run_regression = TRUE, then should the generated folds be stratified based on the outcome (helps to ensure class balance across cross-validation folds)
C	the indicator of coarsening (1 denotes observed, 0 denotes unobserved).
Z	either (i) NULL (the default, in which case the argument C above must be all ones), or (ii) a character list specifying the variable(s) among Y and X that are thought to play a role in the coarsening mechanism.
ipc_weights	weights for the computed influence curve (i.e., inverse probability weights for coarsened-at-random settings)
...	other arguments to the estimation tool, see "See also".

Details

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind this function, and the validity of the confidence intervals. In the interest of transparency, we return most of the calculations within the vim object. This results in a list containing:

- s - the column(s) to calculate variable importance for
- SL.library - the library of learners passed to SuperLearner
- full_fit - the fitted values of the chosen method fit to the full data
- red_fit - the fitted values of the chosen method fit to the reduced data
- est - the estimated variable importance
- naive - the naive estimator of variable importance
- update - the influence curve-based update
- se - the standard error for the estimated variable importance
- ci - the $(1 - \alpha) \times 100\%$ confidence interval for the variable importance estimate
- full_mod - the object returned by the estimation procedure for the full data regression (if applicable)
- red_mod - the object returned by the estimation procedure for the reduced data regression (if applicable)
- alpha - the level, for confidence interval calculation
- y - the outcome

Value

An object of classes `vim` and `vim_rsquared`. See Details for more information.

See Also

`SuperLearner` for specific usage of the `SuperLearner` function and package.

Examples

```
# generate the data
# generate X
p <- 2
n <- 100
x <- data.frame(replicate(p, stats::runif(n, -5, 5)))

# apply the function to the x's
smooth <- (x[,1]/5)^2*(x[,1]+7)/5 + (x[,2]/3)^2

# generate Y ~ Normal (smooth, 1)
y <- smooth + stats::rnorm(n, 0, 1)

# set up a library for SuperLearner; note simple library for speed
library("SuperLearner")
learners <- c("SL.glm", "SL.mean")

# estimate (with a small number of folds, for illustration only)
est <- vimp_rsquared(y, x, indx = 2,
           alpha = 0.05, run_regression = TRUE,
           SL.library = learners, V = 2, cvControl = list(V = 2))
```

---

vimp_se *Estimate standard errors*

Description

Compute standard error estimates for estimates of variable importance.

Usage

```
vimp_se(est, eif, n = length(eif), na.rm = FALSE)
```

Arguments

`est`	the estimate of variable importance.
`eif`	the estimated efficient influence curve.
`n`	the sample size.
`na.rm`	logical; should NA's be removed in computation? (defaults to `FALSE`).

Details

See the paper by Williamson, Gilbert, Simon, and Carone for more details on the mathematics behind this function and the definition of the parameter of interest.

Value

The standard error for the estimated variable importance for the given group of left-out covariates.

Index