

Package ‘flevr’

May 8, 2026

Title Flexible, Ensemble-Based Variable Selection with Potentially Missing Data

Version 0.0.5

Description Perform variable selection in settings with possibly missing data based on extrinsic (algorithm-specific) and intrinsic (population-level) variable importance. Uses a Super Learner ensemble to estimate the underlying prediction functions that give rise to estimates of variable importance. For more information about the methods, please see Williamson and Huang (2024) <[doi:10.1515/ijb-2023-0059](https://doi.org/10.1515/ijb-2023-0059)>.

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 3.1.0)

Imports SuperLearner, dplyr, magrittr, tibble, caret, mvtnorm, kernlab, rlang, ranger

Suggests vimp, stabs, testthat, knitr, rmarkdown, mice, xgboost, glmnet, polspline

URL <https://github.com/bdwilliamson/flevr>

BugReports <https://github.com/bdwilliamson/flevr/issues>

VignetteBuilder knitr

License MIT + file LICENSE

NeedsCompilation no

Author Brian D. Williamson [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-7024-548X>>)

Maintainer Brian D. Williamson <brian.d.williamson@kp.org>

Repository CRAN

Date/Publication 2025-12-06 14:30:02 UTC

Contents

biomarkers	2
extract_importance_glm	3
extract_importance_glmnet	4
extract_importance_mean	5
extract_importance_polymars	6
extract_importance_ranger	7
extract_importance_SL	8
extract_importance_SL_learner	9
extract_importance_svm	10
extract_importance_xgboost	11
extrinsic_selection	12
get_augmented_set	13
get_base_set	15
intrinsic_control	16
intrinsic_selection	17
pool_selected_sets	18
pool_spvims	20
SL.ranger.imp	21
SL_stabs_fitfun	23
spvim_vcov	24
Index	25

biomarkers	<i>Example biomarker data</i>
------------	-------------------------------

Description

A dataset inspired by data collected by the Early Detection Research Network (EDRN). Biomarkers developed at six "labs" are validated at at least one of four "validation sites" on 306 cysts. The data also include two binary outcome variables: whether or not the cyst was classified as mucinous, and whether or not the cyst was determined to have high malignant potential.

Usage

biomarkers

Format

biomarkers: a tibble with 306 rows and 24 columns, where the first column is the validation site, the next two columns are the possible outcomes, and the remaining columns are the biomarkers:

institution the validation site

mucinous a binary indicator of whether the cyst was classified as mucinous

high_malignancy a binary indicator of whether the cyst was classified as having high malignant potential

lab1_actb a biomarker
lab1_molecules_score a biomarker
lab1_telomerase_score a biomarker
lab2_fluorescence_score a biomarker
lab3_muc3ac_score a biomarker
lab3_muc5ac_score a biomarker
lab4_areg_score a biomarker
lab4_glucose_score a biomarker
lab5_mucinous_call a biomarker (binary)
lab5_neoplasia_v1_call a biomarker (binary)
lab5_neoplasia_v2_call a biomarker (binary)
lab6_ab_score a biomarker
cea a biomarker
lab1_molecules_neoplasia_call binary indicator of whether lab1_molecules_score > 25
lab1_telomerase_neoplasia_call binary indicator of whether lab1_telomerase_score > 730
lab2_fluorescence_mucinous_call binary indicator of whether lab2_fluorescence_score > 1.23
lab4_areg_mucinous_call binary indicator of whether lab4_areg_score > 112
lab4_glucose_mucinous_call binary indicator of whether lab4_glucose_score < 50
lab4_combined_mucinous_call binary indicator of whether lab4_areg_score > 112 and lab4_glucose_score < 50
lab6_ab_neoplasia_call binary indicator of whether lab6_ab_score > 0.104
cea_call binary indicator of whether cea > 192

Source

Inspired by data collected by the EDRN <https://edrn.nci.nih.gov/>.

extract_importance_glm

Extract the learner-specific importance from a glm object

Description

Extract the individual-algorithm extrinsic importance from a glm object, along with the importance rank.

Usage

```
extract_importance_glm(fit = NULL, feature_names = "", coef = 0)
```

Arguments

fit the glm object.
 feature_names the feature names
 coef the Super Learner coefficient associated with the learner.

Value

a tibble, with columns algorithm (the fitted algorithm), feature (the feature), importance (the algorithm-specific extrinsic importance of the feature), rank (the feature importance rank, with 1 indicating the most important feature), and weight (the algorithm's weight in the Super Learner)

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# get the fit
fit <- stats::glm(y ~ ., family = "binomial", data = data.frame(y = y, x))
# extract importance
importance <- extract_importance_glm(fit = fit, feature_names = feature_nms)
importance
```

extract_importance_glmnet

Extract the learner-specific importance from a glmnet object

Description

Extract the individual-algorithm extrinsic importance from a glmnet object, along with the importance rank.

Usage

```
extract_importance_glmnet(fit = NULL, feature_names = "", coef = 0)
```

Arguments

fit the glmnet or cv.glmnet object
 feature_names the feature names
 coef the Super Learner coefficient associated with the learner.

Value

a tibble, with columns `algorithm` (the fitted algorithm), `feature` (the feature), `importance` (the algorithm-specific extrinsic importance of the feature), `rank` (the feature importance rank, with 1 indicating the most important feature), and `weight` (the algorithm's weight in the Super Learner)

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# get the fit (using only 3 CV folds for illustration only)
set.seed(20231129)
fit <- glmnet::cv.glmnet(x = as.matrix(x), y = y,
                        family = "binomial", nfolds = 3)
# extract importance
importance <- extract_importance_glmnet(fit = fit, feature_names = feature_nms)
importance
```

extract_importance_mean

Extract the learner-specific importance from a mean object

Description

Extract the individual-algorithm extrinsic importance from a mean object, along with the importance rank.

Usage

```
extract_importance_mean(fit = NULL, feature_names = "", coef = 0)
```

Arguments

<code>fit</code>	the mean object.
<code>feature_names</code>	the feature names
<code>coef</code>	the Super Learner coefficient associated with the learner.

Value

a tibble, with columns `algorithm` (the fitted algorithm), `feature` (the feature), `importance` (the algorithm-specific extrinsic importance of the feature), `rank` (the feature importance rank, with 1 indicating the most important feature), and `weight` (the algorithm's weight in the Super Learner)

Examples

```

data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# get the mean outcome
fit <- mean(y)
# extract importance
importance <- extract_importance_mean(fit = fit, feature_names = feature_nms)
importance

```

```
extract_importance_polymars
```

Extract the learner-specific importance from a polymars object

Description

Extract the individual-algorithm extrinsic importance from a polymars object, along with the importance rank.

Usage

```
extract_importance_polymars(fit = NULL, feature_names = "", coef = 0)
```

Arguments

fit	the polymars object.
feature_names	the feature names
coef	the Super Learner coefficient associated with the learner.

Value

a tibble, with columns algorithm (the fitted algorithm), feature (the feature), importance (the algorithm-specific extrinsic importance of the feature), rank (the feature importance rank, with 1 indicating the most important feature), and weight (the algorithm's weight in the Super Learner)

Examples

```

data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome

```

```

y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
x_mat <- as.matrix(x)
# get the fit
set.seed(20231129)
fit <- polyspline::polyclass(y, x_mat)
# extract importance
importance <- extract_importance_polymars(fit = fit, feature_names = feature_nms)
importance

```

extract_importance_ranger

Extract the learner-specific importance from a ranger object

Description

Extract the individual-algorithm extrinsic importance from a ranger object, along with the importance rank.

Usage

```
extract_importance_ranger(fit = NULL, feature_names = "", coef = 0)
```

Arguments

fit	the ranger object.
feature_names	the feature names
coef	the Super Learner coefficient associated with the learner.

Value

a tibble, with columns algorithm (the fitted algorithm), feature (the feature), importance (the algorithm-specific extrinsic importance of the feature), rank (the feature importance rank, with 1 indicating the most important feature), and weight (the algorithm's weight in the Super Learner)

Examples

```

data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# get the fit
set.seed(20231129)

```

```
fit <- ranger::ranger(y ~ ., data = data.frame(y = y, x), importance = "impurity")
# extract importance
importance <- extract_importance_ranger(fit = fit, feature_names = feature_nms)
importance
```

extract_importance_SL *Extract extrinsic importance from a Super Learner object*

Description

Extract the individual-algorithm extrinsic importance from each fitted algorithm within the Super Learner; compute the average weighted rank of the importance scores, with weights specified by each algorithm's weight in the Super Learner.

Usage

```
extract_importance_SL(fit, feature_names, import_type = "all", ...)
```

Arguments

fit	the fitted Super Learner ensemble
feature_names	the names of the features
import_type	the level of granularity for importance: "all" is the importance based on the weighted average of ranks across algorithms (weights are SL coeffs); "best" is the importance based on the algorithm with highest weight. Defaults to "all".
...	other arguments to pass to individual-algorithm extractors.

Value

a tibble, with columns feature (the feature) and rank (the weighted feature importance rank, with 1 indicating the most important feature).

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# get the fit (using a simple library and 2 folds for illustration only)
set.seed(20231129)
library("SuperLearner")
fit <- SuperLearner::SuperLearner(Y = y, X = x, SL.library = c("SL.glm", "SL.mean"),
```

```

                                cvControl = list(V = 2))
# extract importance using all learners
importance <- extract_importance_SL(fit = fit, feature_names = feature_nms)
importance
# extract importance of best learner
best_importance <- extract_importance_SL(fit = fit, feature_names = feature_nms,
                                         import_type = "best")
best_importance

```

```
extract_importance_SL_learner
```

Extract the learner-specific importance from a fitted SuperLearner algorithm

Description

Extract the individual-algorithm extrinsic importance from one fitted algorithm within the Super Learner, along with the importance rank.

Usage

```
extract_importance_SL_learner(fit = NULL, coef = 0, feature_names = "", ...)
```

Arguments

fit	the specific learner (e.g., from the Super Learner's fitLibrary list).
coef	the Super Learner coefficient associated with the learner.
feature_names	the feature names
...	other arguments to pass to algorithm-specific importance extractors.

Value

a tibble, with columns algorithm (the fitted algorithm), feature (the feature), importance (the algorithm-specific extrinsic importance of the feature), rank (the feature importance rank, with 1 indicating the most important feature), and weight (the algorithm's weight in the Super Learner)

Examples

```

data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# get the fit (using a simple library and 2 folds for illustration only)

```

```

library("SuperLearner")
set.seed(20231129)
fit <- SuperLearner::SuperLearner(Y = y, X = x, SL.library = c("SL.glm", "SL.mean"),
                                cvControl = list(V = 2))

# extract importance
importance <- extract_importance_SL_learner(fit = fit$fitLibrary[[1]]$object,
                                             feature_names = feature_nms, coef = fit$coef[1])

importance

```

```
extract_importance_svm
```

Extract the learner-specific importance from an svm object

Description

Extract the individual-algorithm extrinsic importance from a glm object, along with the importance rank.

Usage

```

extract_importance_svm(
  fit = NULL,
  feature_names = "",
  coef = 0,
  x = NULL,
  y = NULL,
  K = 10
)

```

Arguments

<code>fit</code>	the svm object.
<code>feature_names</code>	the feature names
<code>coef</code>	the Super Learner coefficient associated with the learner.
<code>x</code>	the features
<code>y</code>	the outcome
<code>K</code>	the number of cross-validation folds

Value

a tibble, with columns `algorithm` (the fitted algorithm), `feature` (the feature), `importance` (the algorithm-specific extrinsic importance of the feature), `rank` (the feature importance rank, with 1 indicating the most important feature), and `weight` (the algorithm's weight in the Super Learner)

Examples

```

data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- as.data.frame(dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))])
x_mat <- as.matrix(x)
feature_nms <- names(x)
# get the fit
set.seed(20231129)
fit <- kernlab::ksvm(x_mat, y)
# extract importance
importance <- extract_importance_svm(fit = fit, feature_names = feature_nms, x = x, y = y)
importance

```

```
extract_importance_xgboost
```

Extract the learner-specific importance from an xgboost object

Description

Extract the individual-algorithm extrinsic importance from an xgboost object, along with the importance rank.

Usage

```
extract_importance_xgboost(fit = NULL, feature_names = "", coef = 0)
```

Arguments

fit	the xgboost object.
feature_names	the feature names
coef	the Super Learner coefficient associated with the learner.

Value

a tibble, with columns algorithm (the fitted algorithm), feature (the feature), importance (the algorithm-specific extrinsic importance of the feature), rank (the feature importance rank, with 1 indicating the most important feature), and weight (the algorithm's weight in the Super Learner)

Examples

```

data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- as.matrix(dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))])
feature_nms <- names(x)
set.seed(20231129)
xgbmat <- xgboost::xgb.DMatrix(data = x, label = y)
# get the fit, using a small number of rounds for illustration only
fit <- xgboost::xgb.train(
  data = xgbmat, nrounds = 10,
  params = list("objective" = "binary:logistic",
                "nthread" = 1, "max_depth" = 1)
)
# extract importance
importance <- extract_importance_xgboost(fit = fit, feature_names = feature_nms)
importance

```

extrinsic_selection *Perform extrinsic, ensemble-based variable selection*

Description

Based on a fitted Super Learner ensemble, extract extrinsic variable importance estimates, rank them, and do variable selection using the specified rank threshold.

Usage

```

extrinsic_selection(
  fit = NULL,
  feature_names = "",
  threshold = 20,
  import_type = "all",
  ...
)

```

Arguments

<code>fit</code>	the fitted Super Learner ensemble.
<code>feature_names</code>	the names of the features (a character vector of length <code>p</code> (the total number of features)); only used if the fitted Super Learner ensemble was fit on a matrix rather than on a data.frame, tibble, etc.

threshold	the threshold for selection based on ranked variable importance; rank 1 is the most important. Defaults to 20 (though this is arbitrary, and really should be specified for the task at hand).
import_type	the type of extrinsic importance (either "all", the default, for a weighted combination of the individual-algorithm importance; or "best", for the importance from the algorithm with the highest weight in the Super Learner).
...	other arguments to pass to algorithm-specific importance extractors.

Value

a tibble with the estimated extrinsic variable importance, the corresponding variable importance ranks, and the selected variables.

See Also

[SuperLearner](#) for specific usage of the SuperLearner function and package.

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# get the fit (using a simple library and 2 folds for illustration only)
library("SuperLearner")
set.seed(20231129)
fit <- SuperLearner::SuperLearner(Y = y, X = x, SL.library = c("SL.glm", "SL.mean"),
                                cvControl = list(V = 2))

# extract importance
importance <- extrinsic_selection(fit = fit, feature_names = feature_nms, threshold = 1.5,
                                import_type = "all")

importance
```

get_augmented_set

Get an augmented set based on the next-most significant variables

Description

Based on the adjusted p-values from a FWER-controlling procedure and a more general error rate for which control is desired (e.g., generalized FWER, proportion of false positives, or FDR), augment the set based on FWER control with the next-most significant variables.

Usage

```
get_augmented_set(
  p_values = NULL,
  num_rejected = 0,
  alpha = 0.05,
  quantity = "gFWER",
  q = 0.05,
  k = 1
)
```

Arguments

p_values	the adjusted p-values.
num_rejected	the number of rejected null hypotheses from the base FWER-controlling procedure.
alpha	the significance level.
quantity	the quantity to control (i.e., "gFWER", "PFP", or "FDR").
q	the proportion for FDR or PFP control.
k	the number of false positives for gFWER control.

Value

a list of the variables selected into the augmentation set. Contains the following values:

- set, a numeric vector where 1 denotes that the variable was selected and 0 otherwise
- k, the value of k used
- q_star, the value of q-star used

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# estimate SPVIMs (using simple library and V = 2 for illustration only)
set.seed(20231129)
library("SuperLearner")
est <- vimp::sp_vim(Y = y, X = x, V = 2, type = "auc", SL.library = "SL.glm",
  cvControl = list(V = 2))

# get base set
base_set <- get_base_set(test_statistics = est$test_statistic, p_values = est$p_value,
  alpha = 0.2, method = "Holm")

# get augmented set
augmented_set <- get_augmented_set(p_values = base_set$p_values,
```

```

num_rejected = sum(base_set$decision), alpha = 0.2,
quantity = "gFWER", k = 1)
augmented_set$set

```

get_base_set	<i>Get an initial selected set based on intrinsic importance and a base method</i>
--------------	--

Description

Using the estimated intrinsic importance and a base method designed to control the family-wise error rate (e.g., Holm), obtain an initial selected set.

Usage

```

get_base_set(
  test_statistics = NULL,
  p_values = NULL,
  alpha = 0.05,
  method = "maxT",
  B = 10000,
  Sigma = diag(1, nrow = length(test_statistics)),
  q = NULL
)

```

Arguments

test_statistics	the test statistics (used with "maxT")
p_values	(used with "minP" or "Holm")
alpha	the alpha level
method	the method (one of "none", "BY", "maxT", "minP", or "Holm")
B	the number of resamples (for minP or maxT)
Sigma	the estimated covariance matrix for the test statistics
q	the false discovery rate (for method = "BY")

Value

the initial selected set, a list of the following:

- decision, a numeric vector with 1 indicating that the variable was selected and 0 otherwise
- p_values, the p-values used to make the decision

Examples

```

data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# estimate SPVIMs (using simple library and V = 2 for illustration only)
set.seed(20231129)
library("SuperLearner")
est <- vimp::sp_vim(Y = y, X = x, V = 2, type = "auc", SL.library = "SL.glm",
                   cvControl = list(V = 2))

# get base set
base_set <- get_base_set(test_statistics = est$test_statistic, p_values = est$p_value,
                        alpha = 0.2, method = "Holm")

base_set$decision

```

intrinsic_control

Control parameters for intrinsic variable selection

Description

Control parameters for SPVIM-based intrinsic variable selection.

Usage

```

intrinsic_control(
  quantity = "gFWER",
  base_method = "Holm",
  fdr_method = "Holm",
  q = 0.2,
  k = 5
)

```

Arguments

quantity	the desired quantity for error-rate control: possible values are "gFWER" (the generalized family-wise error rate), "PFP" (the proportion of false positives), and "FDR" (the false discovery rate).
base_method	the family-wise error rate controlling method to use for obtaining the initial set of selected variables. Possible values are "maxT" and "minP" (for step-down procedures based on the test statistics ranked from largest to smallest or the p-values ranked from smallest to largest, respectively) or "Holm" for a procedure based on Holm-adjusted p-values.

fdr_method	the method for controlling the FDR (if quantity = "FDR"); possible values are "BY" (for Benjamini-Yekutieli) or one of the base_methods.
q	the desired proportion of false positives (only used if quantity = "PFP" or "FDR"; a fraction between 0 and 1).
k	the desired number of family-wise errors (an integer, greater than or equal to zero.)

Value

a list with the control parameters.

Examples

```
control <- intrinsic_control(quantity = "gFWER", base_method = "Holm", fdr_method = "Holm",
                             k = 1)
control
```

intrinsic_selection *Perform intrinsic, ensemble-based variable selection*

Description

Based on estimated SPVIM values, do variable selection using the specified error-controlling method.

Usage

```
intrinsic_selection(
  spvim_ests = NULL,
  sample_size = NULL,
  feature_names = "",
  alpha = 0.05,
  control = list(quantity = "gFWER", base_method = "Holm", fdr_method = NULL, q = NULL, k
                 = NULL)
)
```

Arguments

spvim_ests	the estimated SPVIM values (an object of class vim, resulting from a call to vimp::sp_vim). Can also be a list of estimated SPVIMs, if multiple imputation was used to handle missing data; in this case, Rubin's rules will be used to combine the estimated SPVIMs, and then selection will be based on the combined SPVIMs.
sample_size	the number of independent observations used to estimate the SPVIM values.
feature_names	the names of the features (a character vector of length p (the total number of features)); only used if the fitted Super Learner ensemble was fit on a matrix rather than on a data.frame, tibble, etc.

alpha	the nominal generalized family-wise error rate, proportion of false positives, or false discovery rate level to control at (e.g., 0.05).
control	a list of parameters to control the variable selection process. Parameters include quantity, base_method, q, and k. See intrinsic_control for details.

Value

a tibble with the estimated intrinsic variable importance, the corresponding variable importance ranks, and the selected variables.

See Also

[sp_vim](#) for specific usage of the `sp_vim` function and the `vimp` package for estimating intrinsic variable importance.

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# estimate SPVIMs (using simple library and V = 2 for illustration only)
set.seed(20231129)
library("SuperLearner")
est <- vimp::sp_vim(Y = y, X = x, V = 2, type = "auc", SL.library = "SL.glm",
                   cvControl = list(V = 2))
# do intrinsic selection
intrinsic_set <- intrinsic_selection(spvim_est = est, sample_size = nrow(dat_cc), alpha = 0.2,
                                   feature_names = feature_nms,
                                   control = list(quantity = "gFWER", base_method = "Holm",
                                                k = 1))

intrinsic_set
```

pool_selected_sets *Pool selected sets from multiply-imputed data*

Description

Pool the selected sets from multiply-imputed or bootstrap + imputed data. Uses the "stability" of the variables over the multiple selected sets to select variables that are stable across the sets, where stability is determined by presence in a certain fraction of the selected sets (and the fraction must be above the specified threshold to be "stable").

Usage

```
pool_selected_sets(sets = list(), threshold = 0.8)
```

Arguments

sets a list of sets of selected variables from the multiply-imputed datasets. Expects each set of selected variables to be a binary vector, where 1 denotes that the variable was selected.

threshold a numeric threshold between 0 and 1 determining the "stability" of a feature; only features with stability above the threshold after pooling will be in the final selected set of variables.

Value

a vector denoting the final set of selected variables (1 denotes selected, 0 denotes not selected)

Examples

```
data("biomarkers")
x <- biomarkers[, !(names(biomarkers) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
library("dplyr")
library("SuperLearner")
# do multiple imputation (with a small number for illustration only)
library("mice")
n_imp <- 2
set.seed(20231129)
mi_biomarkers <- mice::mice(data = biomarkers, m = n_imp, printFlag = FALSE)
imputed_biomarkers <- mice::complete(mi_biomarkers, action = "long") %>%
  rename(imp = .imp, id = .id)
# set up a list to collect selected sets
all_selected_vars <- vector("list", length = 5)
for (i in 1:n_imp) {
  # fit a Super Learner using simple library for illustration only
  these_data <- imputed_biomarkers %>%
    filter(imp == i)
  this_y <- these_data$mucinous
  this_x <- these_data %>%
    select(starts_with("lab"), starts_with("cea"))
  this_x_df <- as.data.frame(this_x)
  fit <- SuperLearner::SuperLearner(Y = this_y, X = this_x_df,
    SL.library = "SL.glm",
    cvControl = list(V = 2),
    family = "binomial")

  # do extrinsic selection
  all_selected_vars[[i]] <- extrinsic_selection(
    fit = fit, feature_names = feature_nms, threshold = 5, import_type = "all"
  )$selected
}
# perform extrinsic variable selection
selected_vars <- pool_selected_sets(sets = all_selected_vars, threshold = 1 / n_imp)
```

```
feature_nms[selected_vars]
```

```
pool_spvims
```

```
Pool SPVIM Estimates Using Rubin's Rules
```

Description

If multiple imputation was used due to the presence of missing data, pool SPVIM estimates from individual imputed datasets using Rubin's rules. Results in point estimates averaged over the imputations, along with within-imputation variance estimates and across-imputation variance estimates; and test statistics and p-values for hypothesis testing.

Usage

```
pool_spvims(spvim_est = NULL)
```

Arguments

```
spvim_est      a list of estimated SPVIMs (of class vim)
```

Value

a list of results containing the following:

- `est`, the average SPVIM estimate over the multiply-imputed datasets
- `se`, the average of the within-imputation SPVIM variance estimates
- `test_statistics`, the test statistics for hypothesis tests of zero importance, using the Rubin's rules standard error estimator and average SPVIM estimate
- `p_values`, p-values computed using the above test statistics
- `tau_n`, the across-imputation variance estimates
- `vcov`, the overall variance-covariance matrix

Examples

```
data("biomarkers")
library("dplyr")
# do multiple imputation (with a small number for illustration only)
library("mice")
n_imp <- 2
set.seed(20231129)
mi_biomarkers <- mice::mice(data = biomarkers, m = n_imp, printFlag = FALSE)
imputed_biomarkers <- mice::complete(mi_biomarkers, action = "long") %>%
  rename(imp = .imp, id = .id)
# estimate SPVIMs for each imputed dataset, using simple library for illustration only
library("SuperLearner")
est_lst <- lapply(as.list(1:n_imp), function(l) {
  this_x <- imputed_biomarkers %>%
```

```

    filter(imp == 1) %>%
    select(starts_with("lab"), starts_with("cea"))
this_y <- biomarkers$mucinous
suppressWarnings(
  vimp::sp_vim(Y = this_y, X = this_x, V = 2, type = "auc",
              SL.library = "SL.glm", gamma = 0.1, alpha = 0.05, delta = 0,
              cvControl = list(V = 2), env = environment())
)
})
# pool the SPVIMs using Rubin's rules
pooled_spvims <- pool_spvims(spvim_estimates = est_lst)
pooled_spvims

```

SL.ranger.imp

Super Learner wrapper for a ranger object with variable importance

Description

Super Learner wrapper for a ranger object with variable importance

Usage

```

SL.ranger.imp(
  Y,
  X,
  newX,
  family,
  obsWeights = rep(1, length(Y)),
  num.trees = 500,
  mtry = floor(sqrt(ncol(X))),
  write.forest = TRUE,
  probability = family$family == "binomial",
  min.node.size = ifelse(family$family == "gaussian", 5, 1),
  replace = TRUE,
  sample.fraction = ifelse(replace, 1, 0.632),
  num.threads = 1,
  verbose = FALSE,
  importance = "impurity",
  ...
)

```

Arguments

Y	Outcome variable
X	Training dataframe
newX	Test dataframe

family	Gaussian or binomial
obsWeights	Observation-level weights
num.trees	Number of trees.
mtry	Number of variables to possibly split at in each node. Default is the (rounded down) square root of the number variables.
write.forest	Save ranger.forest object, required for prediction. Set to FALSE to reduce memory usage if no prediction intended.
probability	Grow a probability forest as in Malley et al. (2012).
min.node.size	Minimal node size. Default 1 for classification, 5 for regression, 3 for survival, and 10 for probability.
replace	Sample with replacement.
sample.fraction	Fraction of observations to sample. Default is 1 for sampling with replacement and 0.632 for sampling without replacement.
num.threads	Number of threads to use.
verbose	If TRUE, display additional output during execution.
importance	Variable importance mode, one of 'none', 'impurity', 'impurity_corrected', 'permutation'. The 'impurity' measure is the Gini index for classification, the variance of the responses for regression and the sum of test statistics (see <code>splitrule</code>) for survival.
...	Any additional arguments, not currently used.

Value

a named list with elements `pred` (predictions on `newX`) and `fit` (the fitted ranger object).

References

- Breiman, L. (2001). Random forests. *Machine learning* 45:5-32.
- Wright, M. N. & Ziegler, A. (2016). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software*, in press. <http://arxiv.org/abs/1508.04409>.

See Also

[SL.ranger](#) [ranger](#) [predict.ranger](#)

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
```

```
# get the fit
set.seed(20231129)
fit <- SL.ranger.imp(Y = y, X = x, newX = x, family = binomial())
fit
```

SL_stabs_fitfun	<i>Wrapper for using Super Learner-based extrinsic selection within stability selection</i>
-----------------	---

Description

A wrapper function for Super Learner-based extrinsic variable selection within stability selection, using the stabs package.

Usage

```
SL_stabs_fitfun(x, y, q, ...)
```

Arguments

x	the features.
y	the outcome of interest.
q	the number of features to select on average.
...	other arguments to pass to SuperLearner.

Value

a named list, with elements: selected (a logical vector indicating whether or not each variable was selected); and path (a logical matrix indicating which variable was selected at each step).

See Also

[stabsel](#) for general usage of stability selection.

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# use stability selection with SL (using small number of folds for CV,
# small SL library and small number of bootstrap replicates for illustration only)
set.seed(20231129)
library("SuperLearner")
```

```
sl_stabs <- stabs::stabsel(x = x, y = y,
                        fitfun = SL_stabs_fitfun,
                        args.fitfun = list(SL.library = "SL.glm", cvControl = list(V = 2)),
                        q = 2, B = 5, PFER = 5)

sl_stabs
```

spvim_vcov

Extract a Variance-Covariance Matrix for SPVIM Estimates

Description

Extract a variance-covariance matrix based on the efficient influence function for each of the estimated SPVIMs.

Usage

```
spvim_vcov(spvim_ests = NULL)
```

Arguments

spvim_ests estimated SPVIMs

Value

a variance-covariance matrix

Examples

```
data("biomarkers")
# subset to complete cases for illustration
cc <- complete.cases(biomarkers)
dat_cc <- biomarkers[cc, ]
# use only the mucinous outcome, not the high-malignancy outcome
y <- dat_cc$mucinous
x <- dat_cc[, !(names(dat_cc) %in% c("mucinous", "high_malignancy"))]
feature_nms <- names(x)
# estimate SPVIMs (using simple library and V = 2 for illustration only)
set.seed(20231129)
library("SuperLearner")
est <- vimp::sp_vim(Y = y, X = x, V = 2, type = "auc", SL.library = "SL.glm",
                  cvControl = list(V = 2))
# get variance-covariance matrix
vcov <- spvim_vcov(spvim_ests = est)
```

Index

* datasets

biomarkers, [2](#)

biomarkers, [2](#)

extract_importance_glm, [3](#)

extract_importance_glmnet, [4](#)

extract_importance_mean, [5](#)

extract_importance_polymars, [6](#)

extract_importance_ranger, [7](#)

extract_importance_SL, [8](#)

extract_importance_SL_learner, [9](#)

extract_importance_svm, [10](#)

extract_importance_xgboost, [11](#)

extrinsic_selection, [12](#)

get_augmented_set, [13](#)

get_base_set, [15](#)

intrinsic_control, [16](#), [18](#)

intrinsic_selection, [17](#)

pool_selected_sets, [18](#)

pool_spvims, [20](#)

predict.ranger, [22](#)

ranger, [22](#)

SL.ranger, [22](#)

SL.ranger.imp, [21](#)

SL_stabs_fitfun, [23](#)

sp_vim, [18](#)

spvim_vcov, [24](#)

stabsel, [23](#)

SuperLearner, [13](#)