Package 'mossyforest'

November 20, 2026

Title Network Aware Stability Selection with Random Forests

Version 0.1.0

Description An interpretable, network aware random forest feature selection framework for high dimensional and correlated data. It combines module stratified recursive feature elimination, shadow stability selection, and TreeSHAP (Tree SHapley Additive exPlanations) importance, with modules either provided by the user or derived with WGCNA (weighted gene co expression network analysis). The default backend requires a 'Python' environment with 'scikit-learn', 'shap', and 'numpy', which can be configured with mf_setup(). A pure 'R' backend based on 'ranger' is also available.

Imports grDevices, reticulate (>= 1.35), ranger (>= 0.16.0), shapviz (>= 0.9.4), ggplot2 (>= 3.4.0), dplyr (>= 1.1.0), scales (>= 1.3.0), stats, utils, lifecycle, withr

License GPL (>= 3)

URL https://github.com/timothyzshen/mossyforest, https://github.com/timothyzshen/mossyforest-py, https://timothyzshen.github.io/mossyforest/

BugReports https://github.com/timothyzshen/mossyforest/issues

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

SystemRequirements Python (>= 3.9) with scikit-learn, shap, and numpy

LazyData true

Suggests WGCNA, treeshap, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation no

Author Timothy Shen [aut, cre, cph], Christina Ramirez [aut]

Maintainer Timothy Shen <tzshen@g.ucla.edu>

Repository CRAN

Date/Publication 2026-11-20 11:30:13 UTC

Contents

detect_interaction 2
leukemia 4
mf 5
mf_setup 7
plot_decisions 8
plot_elbow 9
plot_force 10
plot_importance 12
plot_modules 13
plot_potential_interactions 14
plot_signed_importance 15
plot_stability 17
plot_stability_elbow 18
plot_waterfall 20
predict.mossy_forest 21
print.mossy_forest 22
screen_control 23
select_control 24
wdi 26
WGCNA_control 27
wmf 28

Index **30**

`detect_interaction` *Detect Pairwise Interactions from a Mossy Forest Object*

Description

Returns TreeSHAP pairwise interaction strengths for all stable features. Interaction values are computed during model fitting and stored in the object, either by the Python backend (`TreeExplainer.shap_interaction_valu` or by the pure R backend with `r_shap = "treeshap"`. They are not available with `r_shap = "permutation"` or when `compute_interactions = FALSE`.

Usage

```
detect_interaction(object, thresh = 0, all = FALSE, verbose = TRUE, ...)
```

Arguments

`object`	A `mossy_forest` object.
`thresh`	Interaction strength threshold. Only pairs whose mean $\lvert\phi_{jk}\rvert$ exceeds `thresh` are returned (those above the threshold). Set to `0` to return all pairs.
`all`	If `TRUE`, return every off diagonal pair (upper triangle) regardless of `thresh`. Default `FALSE`.
`verbose`	If `TRUE`, print the table of above threshold pairs (or a message if none are found). Default `TRUE`.
`...`	Ignored.

Details

The $p \times p$ **interaction matrix** stored in `object$shap_obj$interaction_matrix` is defined as:

$$M_{jk} = \frac{1}{n} \sum_{i=1}^{n} |\phi_{jk}(x_i)|$$

where $\phi_{jk}(x_i)$ is the TreeSHAP interaction value for the pair (j, k) at observation i (Lundberg et al., 2018). Diagonal entries are main effects and off diagonal entries are pairwise interactions.

Value

A data frame with columns `FeatureA`, `FeatureB`, and `Interaction_Strength` (mean absolute TreeSHAP interaction), sorted by interaction strength descending. Invisibly when `all = TRUE`.

References

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S.-I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2, 56-67.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", r_shap = "treeshap", verbose = 0, seed = 1)
detect_interaction(fit)
```

`leukemia` *ALL Leukemia Gene Expression Data*

Description

A list containing the top 500 most variable probe sets from the Acute Lymphoblastic Leukemia (ALL) microarray dataset (Chiaretti et al., 2004), along with the corresponding molecular subtype outcome.

Usage

```
data(leukemia)
```

Format

A list with two elements:

X A numeric matrix with 79 rows (patients) and 500 columns (Affymetrix HGU95Av2 probe sets, already $\log_2$-scaled by the original authors).

y An integer vector of length 79: `1` (BCR/ABL subtype) or `0` (NEG, cytogenetically normal), derived from the `mol.biol` column of the original ALL phenotype data.

Details

Starting from the `ALL` Bioconductor package (128 patients, 12 625 probe sets), samples are restricted to the B lineage cases carrying one of two balanced molecular subtypes which were the BCR/ABL (Philadelphia chromosome, Ph+) fusion versus cytogenetically normal (NEG). This creates 79 patients (37 BCR/ABL, 42 NEG).Features were then ranked by across sample variance and the top 500 are retained. See `data-raw/prep_leukemia.R` for the exact preprocessing steps.

Using `y` as a classification outcome

`y` is stored as an integer vector, so `mf()` treats it as a regression outcome. For classification, convert it to a factor first. The first level is the class that SHAP values and directions refer to:

```
y <- factor(leukemia$y, levels = c(1, 0), labels = c("BCR/ABL", "NEG"))
```

Source

The Bioconductor `ALL` package, which is distributed under the Artistic-2.0 license. This dataset is a subset of its samples and probe sets; the expression values are unchanged.

References

Chiaretti, S. et al. (2004). Gene expression profile of adult T cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. *Blood*, 103(7), 2771-2778. doi:10.1182/blood2003093243

Li, X. *ALL: A data package*. Bioconductor. doi:10.18129/B9.bioc.ALL

mf *Mossy Forest Feature Selection*

Description

Runs the Mossy Forest algorithm.

Usage

```
mf(
  X,
  y,
  module_membership,
  screen_params = screen_control(),
  select_params = select_control(),
  final_ntree = 500L,
  nodesize = NULL,
  num_processors = 1L,
  max_modsize = 5000L,
  verbose = 1L,
  seed = as.integer(Sys.time()),
  compute_interactions = TRUE,
  backend = "python",
  r_shap = "permutation"
)
```

Arguments

X	A data frame where each column is a feature.
y	Response vector (numeric for regression, factor for classification).
module_membership	A named character vector assigning each feature in X to a module. Names must match colnames(X). Modules named in select_control(unassigned_modules) form the unassigned pool, which is selected with its own elbow based threshold; all other modules use pi_thr.
screen_params	Screening parameters. See screen_control.
select_params	Selection parameters. See select_control.
final_ntree	Number of trees in the final random forest. Default 500.
nodesize	Minimum terminal node size. Defaults to 1 for classification and 5 for regression.
num_processors	Number of parallel threads for random forests. Default 1.
max_modsize	Maximum module size before blockwise pre reduction is applied (Python backend only). Default 5000.
verbose	Verbose level. 0 = silent; 1 = progress messages; 2 = detailed output. Default 1.

seed Integer random seed for reproducibility. Defaults to current system time.

compute_interactions
Logical. If TRUE (default), the TreeSHAP interaction matrix is computed by the Python backend and by the pure R backend with r_shap = "treeshap". Set FALSE to skip it (faster); detect_interaction() and plot_potential_interactions() then have nothing to use. It has no effect with r_shap = "permutation", which never computes interactions.

backend Computation backend. "python" (default) uses a Python scikit-learn / SHAP engine via **reticulate** and provides exact TreeSHAP values plus an interaction matrix. "R" uses a R **ranger** implementation, no Python installation required. Choose the final importance method with r_shap.

r_shap Final SHAP / importance method used when backend = "R". Ignored for the Python backend. One of:

"permutation" Ranger permutation variable importance (default). No extra packages needed. Returns a scalar VIM per feature; per observation SHAP values and SHAP based plots are unavailable.

"treeshap" Exact TreeSHAP values via the R **treeshap** package. Requires **treeshap**. Also computes a $p \times p$ interaction matrix, enabling detect_interaction() and plot_potential_interactions().

Value

An object of class mossy_forest.

References

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., & Robins, J. (2018). Double/cross-fit residualed machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1), C1-C68.

Lundberg, S. M., & Lee, S. I. (2017). A unified model for interpreting predictions. *NeurIPS*, 30.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", verbose = 0, seed = 1)
print(fit)
```

```
head(fit$final_SHAP)

## Not run:
# The Python backend (exact TreeSHAP) needs a configured Python environment
mf_setup(condaenv = "sfenv")
fit_py <- mf(X, y, module_membership = membership, backend = "python")
plot_importance(fit_py)

## End(Not run)
```

`mf_setup` *Configure the Python environment for Mossy Forest*

Description

Tells Mossy Forest which Python installation to use for its backend. Call this once at the start of each R session before using `mf` with `backend = "python"`. The pure R backend does not need it.

Usage

```
mf_setup(python = NULL, condaenv = NULL, virtualenv = NULL)
```

Arguments

`python`	Path to a Python binary. Passed to `reticulate::use_python()`.
`condaenv`	Name of a conda environment. Passed to `reticulate::use_condaenv()`.
`virtualenv`	Path to a virtual environment. Passed to `reticulate::use_virtualenv()`.

Details

The Python environment must have `scikit-learn`, `shap`, and `numpy` installed. Example setup:

```
conda create -n sfenv python=3.11
conda activate sfenv
pip install scikit-learn shap numpy
```

Value

Invisible `NULL`.

Examples

```
## Not run:
  mf_setup(condaenv = "sfenv")
  mf_setup(python = "/usr/local/bin/python3")

## End(Not run)
```

plot_decisions *Plot Decision Plot for a Mossy Forest Object*

Description

Creates a cumulative SHAP decision plot adapted from the Python shap package.

Usage

```
plot_decisions(object, ...)

## S3 method for class 'mossy_forest'
plot_decisions(
  object,
  highlight = NULL,
  just_shap = FALSE,
  plot_title = "Decision Plot",
  geom_point = FALSE,
  gradient = c("blue", "red"),
  ...
)
```

Arguments

object	A mossy_forest object. If just_shap = TRUE, pass a data frame of SHAP values instead.
...	Ignored.
highlight	Row indices or feature names to highlight. NULL shows all observations. Default NULL.
just_shap	If TRUE, object is treated as a plain SHAP data frame. Default FALSE.
plot_title	Plot title. Default "Decision Plot".
geom_point	Add a point at each feature step? Default FALSE.
gradient	Length 2 color vector for the prediction gradient. Default c("blue", "red").

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_decisions(mossy_forest): Decision plot for a mossy_forest object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", r_shap = "treeshap", verbose = 0, seed = 1)
plot_decisions(fit)
```

---

plot_elbow *Plot Elbow Plot of Absolute SHAP Values for a Mossy Forest Object*

Description

Ranks stable features by their importance score and displays an elbow (scree) plot, optionally colored by module membership. The score is mean |SHAP| for the Python backend and for the pure R backend with `r_shap = "treeshap"`, and ranger permutation importance for `r_shap = "permutation"`.

Usage

```
plot_elbow(object, ...)

## S3 method for class 'mossy_forest'
plot_elbow(
  object,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  colorName = NULL,
  color = TRUE,
  module_labels = NULL,
  ...
)
```

Arguments

`object`	A `mossy_forest` object.
`...`	Ignored.

main	Plot title. Default "Elbow Plot of Absolute SHAP Values" ("Elbow Plot of Variable Importance" when the fit has no SHAP scores).
xlab	X axis label. Default "Features".
ylab	Y axis label. Default "Absolute SHAP Values" ("Variable Importance" when the fit has no SHAP scores).
colorName	Legend title for module color. Default "Module".
color	Color points by module membership? Default TRUE.
module_labels	Optional two column data frame of label mappings. Default NULL.

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_elbow(mossy_forest): Elbow plot for a mossy_forest object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", verbose = 0, seed = 1)
plot_elbow(fit)
```

---

plot_force *Plot SHAP Force Plot for Specific Observations in a Mossy Forest Object*

Description

Generates a SHAP force plot for one or more observations. Plots are built via shapviz.

Usage

```
plot_force(object, ...)

## S3 method for class 'mossy_forest'
plot_force(
  object,
  row_id,
  row_name = NULL,
  max_display = NULL,
  fill_colors = c("#59c46b", "#3b528b"),
  contrast = TRUE,
  bar_label_size = 3.2,
  show_annotation = TRUE,
  ...
)
```

Arguments

object	A mossy_forest object.
...	Ignored.
row_id	Row index (or indices) for the observation(s) to plot.
row_name	Optional name(s) for the observation(s).
max_display	Maximum number of features to display. NULL shows all. Default NULL.
fill_colors	Length 2 vector of colors. Default c("#59c46b", "#3b528b").
contrast	Apply text/arrow contrast? Default TRUE.
bar_label_size	Label size on bars. Default 3.2.
show_annotation	Annotate bars. Default TRUE.

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_force(mossy_forest): Force plot for a mossy_forest object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
```

```
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", r_shap = "treeshap", verbose = 0, seed = 1)
plot_force(fit, row_id = 1)
```

`plot_importance` *Plot Feature Importance for a Mossy Forest Object*

Description

Generates a feature importance plot from a `mossy_forest` object. Three plot types of bar, beeswarm, and both can be used. Plots are built via shapviz.

Usage

```
plot_importance(object, ...)

## S3 method for class 'mossy_forest'
plot_importance(
  object,
  kind = "beeswarm",
  color_bar_title = "Feature Value",
  max_display = NULL,
  fill = "#3e568a",
  sort_features = TRUE,
  show_numbers = TRUE,
  viridis_args = list(begin = 0.25, end = 0.75, option = "viridis"),
  ...
)
```

Arguments

`object`	A `mossy_forest` object.
`...`	Ignored.
`kind`	Plot type: `"bar"`, `"beeswarm"`, or `"both"`. Default `"beeswarm"`.
`color_bar_title`	Title for the color bar in the beeswarm plot. Default `"Feature Value"`.
`max_display`	Maximum number of features to display. `NULL` shows all. Default `NULL`.
`fill`	Bar/point color Any valid `ggplot2` color or hex code. Default `"#3e568a"`.
`sort_features`	Sort features by importance? Default `TRUE`.
`show_numbers`	Show numeric SHAP values on the bar plot? Default `TRUE`.
`viridis_args`	Named list of arguments forwarded to `viridis` for the beeswarm color scale. Default `list(begin = 0.25, end = 0.75, option = "viridis")`.

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_importance(mossy_forest): Importance plot for a mossy_forest object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", r_shap = "treeshap", verbose = 0, seed = 1)
plot_importance(fit)
```

---

plot_modules *Plot Module Membership Distribution for a Mossy Forest Object*

Description

Shows the proportion of features per module that were selected as stable, colored by selection status.

Usage

```
plot_modules(object, ...)

## S3 method for class 'mossy_forest'
plot_modules(
  object,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  module_labels = NULL,
  ...
)
```

Arguments

object	A mossy_forest object.
...	Ignored.
main	Plot title. Default "Module Membership Distribution".
xlab	X axis label. Default "Module".
ylab	Y axis label. Default "Proportion of features".
module_labels	Optional two column data frame mapping old module labels to new display labels. Default NULL.

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_modules(mossy_forest): Module membership plot for a mossy_forest object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", verbose = 0, seed = 1)
plot_modules(fit)
```

---

plot_potential_interactions

Plot Potential Interaction Matrix for a Mossy Forest Object

Description

Generates a heatmap of pairwise SHAP based potential interaction strengths for all stable features.

Usage

```
plot_potential_interactions(object, ...)

## S3 method for class 'mossy_forest'
plot_potential_interactions(object, ...)
```

Arguments

object	A mossy_forest object.
...	Ignored.

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_potential_interactions(mossy_forest): Interaction heatmap for a mossy_forest object (uses exact TreeSHAP interaction values).

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", r_shap = "treeshap", verbose = 0, seed = 1)
plot_potential_interactions(fit)
```

---

plot_signed_importance

Plot Directional (Signed) Feature Importance for a Mossy Forest Object

---

Description

Bar chart of signed importance = direction $\times$ mean|SHAP|. Magnitude is mean|SHAP| and the sign is the global effect direction taken from the Spearman correlation between each feature's value and its SHAP value (`mean(signed SHAP)` ≈ 0 because contributions cancel, so it is unusable for direction). Bars to the right increase the average predicted outcome, and bars to the left decrease it. These are associations in the fitted model, not causal effects of changing a feature. Features whose direction is ambiguous (non monotone: $|\rho| < 0.1$) are greyed. For binary classification, "the outcome" means the probability of `levels(y)[1]`, the first level of the factor passed as `y` to `mf`. Relevel `y` first if you want a different reference class.

Usage

```
plot_signed_importance(object, ...)

## S3 method for class 'mossy_forest'
plot_signed_importance(
  object,
  top_n = 20L,
  labels = NULL,
  pos_color = "#4393c3",
  neg_color = "#d6604d",
  amb_color = "#bbbbbb",
  pos_label = "increase",
  neg_label = "decrease",
  amb_label = "ambiguous",
  main = "Directional feature importance",
  ...
)
```

Arguments

`object`	A `mossy_forest` object.
`...`	Ignored.
`top_n`	Show only the `top_n` features with the largest |signed importance|. Default `20L`; use `NULL` or `Inf` to plot every stable feature.
`labels`	Optional named character vector mapping feature names (as in `final_SHAP$feature_name`) to readable labels for the axis, e.g. a gene symbol or indicator name lookup. Names not found are left unchanged. Default `NULL` (show the raw feature names).
`pos_color`	Bar color for positive direction. Default `"#4393c3"`.
`neg_color`	Bar color for negative direction. Default `"#d6604d"`.
`amb_color`	Bar color for ambiguous features. Default `"#bbbbbb"`.
`pos_label`	Legend text for the positive direction group. Default `"increase"`.
`neg_label`	Legend text for the negative direction group. Default `"decrease"`.
`amb_label`	Legend text for the ambiguous group. Default `"ambiguous"`.
`main`	Plot title. Default `"Directional feature importance"`.

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_signed_importance(mossy_forest): Signed importance plot for a mossy_forest object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", r_shap = "treeshap", verbose = 0, seed = 1)
plot_signed_importance(fit)
```

---

plot_stability *Plot Shadow Stability Frequencies for a Mossy Forest Object*

Description

Displays the subsample shadow stability selection frequency for each stable feature as a horizontal bar chart. Features are sorted by frequency.

Usage

```
plot_stability(object, ...)

## S3 method for class 'mossy_forest'
plot_stability(
  object,
  fill = "#3e568a",
  main = "Shadow Stability Frequencies",
  ...
)
```

Arguments

object	A mossy_forest object.
...	Ignored.
fill	Bar color. Default "#3e568a".
main	Plot title. Default "Shadow Stability Frequencies".

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_stability(mossy_forest): Stability frequency plot for a mossy_forest object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", verbose = 0, seed = 1)
plot_stability(fit)
```

---

plot_stability_elbow *Plot Shadow Stability Selection Curves for a Mossy Forest Object*

Description

Displays the shadow subsample stability frequency curve for each selection pool with a horizontal threshold line and colored points indicating which features were selected.

Usage

```
plot_stability_elbow(object, ...)

## S3 method for class 'mossy_forest'
plot_stability_elbow(
```

```
  object,
  col_selected = "#3e568a",
  col_unselected = "grey65",
  col_threshold = "tomato",
  col_elbow = "#e07b00",
  show_labels = FALSE,
  ncol = NULL,
  main = "Shadow stability selection",
  ...
)
```

Arguments

object	A mossy_forest object.
...	Ignored.
col_selected	Point color for selected features. Default "#3e568a".
col_unselected	Point color for unselected features. Default "grey65".
col_threshold	Color of the horizontal threshold line. Default "tomato".
col_elbow	Color of the vertical elbow rank line. Default "#e07b00".
show_labels	Show feature names on points. Default FALSE. Set TRUE only for small pools ($p \leq 20$).
ncol	Number of columns in the facet grid. NULL lets ggplot choose. Default NULL.
main	Plot title. Default "Shadow stability selection".

Details

For shadow_mode = "split", there is one panel for the correlated pool and one for the unassigned pool (elbow or fixed threshold). For shadow_mode = "within_module" one panel is shown per module.

The dashed red line marks the stability threshold applied. For unassigned pools with elbow detection the elbow rank is marked with a vertical dotted line.

Value

A ggplot2 object (invisibly).

Methods (by class)

- plot_stability_elbow(mossy_forest): Stability elbow plot for a mossy_forest object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
```

```
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", verbose = 0, seed = 1)
plot_stability_elbow(fit)
```

`plot_waterfall` *Plot Waterfall for Specific Observations in a Mossy Forest Object*

Description

Generates a SHAP waterfall plot for one or more observations. Plots are built via shapviz.

Usage

```
plot_waterfall(object, ...)

## S3 method for class 'mossy_forest'
plot_waterfall(
  object,
  row_id,
  row_name = NULL,
  max_display = NULL,
  order_fun = function(s) order(abs(s)),
  fill_colors = c("#59c46b", "#3b528b"),
  contrast = TRUE,
  show_connection = TRUE,
  show_annotation = TRUE,
  ...
)
```

Arguments

`object`	A `mossy_forest` object.
`...`	Ignored.
`row_id`	Row index (or indices) for the observation(s) to plot. When multiple indices are given, SHAP values are averaged.
`row_name`	Optional name(s) for the observation(s).
`max_display`	Maximum number of features to display. `NULL` shows all. Default `NULL`.
`order_fun`	Function used to order features. Default orders by absolute SHAP value.

`fill_colors` Length 2 vector of colors for positive/negative bars. Default `c("#59c46b", "#3b528b")`.

`contrast` Apply text/arrow contrast? Default `TRUE`.

`show_connection` Show connections between bars? Default `TRUE`.

`show_annotation` Annotate bars? Default `TRUE`.

Value

A ggplot2 object (invisibly).

Methods (by class)

- `plot_waterfall(mossy_forest)`: Waterfall plot for a `mossy_forest` object.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", r_shap = "treeshap", verbose = 0, seed = 1)
plot_waterfall(fit, row_id = 1)
```

---

predict.mossy_forest *Predict from a Mossy Forest Object*

Description

Generates predictions from the final random forest fitted on the stable feature set.

Usage

```
## S3 method for class 'mossy_forest'
predict(object, new_data, ...)
```

Arguments

object	A mossy_forest object.
new_data	A matrix or data frame of new observations. Column names must match the training features.
...	Additional arguments passed to predict.ranger.

Value

A ranger prediction object. Predictions always come from the ranger forest refit on the stable features, whichever backend selected them.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", verbose = 0, seed = 1)
predict(fit, X[1:5, ])$predictions
```

---

print.mossy_forest *Print a Mossy Forest Object*

Description

Prints the stable feature set with final feature importance scores and shadow stability selection frequencies.

Usage

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

Arguments

x	A mossy_forest object.
...	Ignored.

Value

Invisibly returns `x`.

Examples

```
# A small simulated regression problem with two modules of features
set.seed(1)
n <- 60
z <- rnorm(n)
X <- data.frame(a1 = z + rnorm(n, sd = 0.3), a2 = z + rnorm(n, sd = 0.3),
                a3 = z + rnorm(n, sd = 0.3), b1 = rnorm(n), b2 = rnorm(n),
                b3 = rnorm(n))
y <- z + rnorm(n, sd = 0.3)
membership <- c(a1 = "A", a2 = "A", a3 = "A", b1 = "B", b2 = "B", b3 = "B")

# The pure R backend needs no Python installation
fit <- mf(X, y, module_membership = membership,
          screen_params = screen_control(min_ntree = 50),
          select_params = select_control(n_reps = 10, min_ntree = 50),
          final_ntree = 50, backend = "R", verbose = 0, seed = 1)
print(fit)
```

---

screen_control *Screening Step Parameters*

Description

Creates a parameter object controlling the screening (recursive feature elimination) step of `mf`.

Usage

```
screen_control(
  drop_fraction = 0.25,
  keep_fraction = 0.5,
  mtry_factor = 1,
  mtry_rule = "auto",
  min_ntree = 500L,
  ntree_factor = 1
)
```

Arguments

`drop_fraction`	Fraction of features dropped at each RFE iteration. Default `0.25`.
`keep_fraction`	Fraction of features retained per module after screening. Default `0.50`.
`mtry_factor`	Multiplier applied to the `mtry_rule`'s base value to derive `mtry` for each screening random forest (`mtry = ceiling(mtry_factor * base)`, constrained to `[1, p]`). Default `1`.

mtry_rule	How mtry is derived from the feature count p. "auto" (default) uses sqrt(p) for classification and p/3 for regression. "sqrt" uses sqrt(p) for classification AND regression. "p_over_3" uses p/3 for classification AND regression.
min_ntree	Minimum number of trees per screening forest. Default 500.
ntree_factor	Trees per forest = max(p * ntree_factor, min_ntree). Default 1.

Value

An object of class screen_control.

See Also

select_control, mf

Examples

```
screen_control()
screen_control(drop_fraction = 0.1, keep_fraction = 0.3, min_ntree = 200)
```

---

select_control *Selection Step Parameters*

Description

Creates a parameter object controlling the shadow stability selection step of mf.

Usage

```
select_control(
  mtry_factor = 1,
  mtry_rule = "auto",
  mtry_on_real = FALSE,
  min_ntree = 500L,
  ntree_factor = 1,
  shadow_mode = "split",
  n_reps = 50L,
  pi_thr = 0.6,
  pi_thr_unassigned = NULL,
  threshold_mode = "pi_thr",
  shadow_percentile = 95,
  unassigned_modules = NULL,
  use_cfres = FALSE,
  cfres_n_folds = 5L,
  cfres_ntree = 300L,
  cfres_scope = "unassigned",
  early_stop = FALSE,
  early_stop_tol = 0.01,
  early_stop_check_every = 10L
)
```

Arguments

mtry_factor	mtry multiplier for random forests. Default 1.
mtry_rule	How mtry is derived from the feature count. One of "auto" (default), "sqrt", "p_over_3". See screen_control. Unlike screening, a single stability selection run can fit BOTH classification and regression forests even though y is one fixed type. For instance, when use_cfres = TRUE, any pool that gets cfRes residualized (see cfres_scope) turns continuous for that pool's own forest.
mtry_on_real	Logical. If TRUE, mtry on the augmented $[real\|shadow]$ matrix is 2 * rule(p_real) so real feature sampling is not diluted by the shadows. Default FALSE.
min_ntree	Minimum trees per forest. Default 500.
ntree_factor	Trees per forest scaling factor. Default 1.
shadow_mode	Shadow stability pooling strategy. One of "split" (correlated and unassigned pools evaluated separately) or "within_module" (one shadow run per module). Default "split".
n_reps	Number of subsample replicates for shadow stability (each draws half the rows without replacement). Default 50.
pi_thr	Stability frequency threshold for correlated pool features. A feature is selected if its shadow win rate across replicates exceeds pi_thr. Default 0.60.
pi_thr_unassigned	Stability threshold for the unassigned pool. NULL (default) uses an automatic elbow detection rule. We recommend keeping this NULL.
threshold_mode	Selection cutoff rule. "pi_thr" (default) uses the fixed pi_thr for the correlated pool and elbow detection for the unassigned pool. "elbow" applies the adaptive elbow rule to *both* pools.
shadow_percentile	Percentile of the shadow feature distribution used as the importance threshold within each replicate. Default 95.
unassigned_modules	Character vector of module names treated as the unassigned pool (e.g. "M10_unassigned"). Names must match values in module_membership; a name that matches no module gives a warning. NULL disables split pool logic.
use_cfres	Logical. If TRUE, a cross fitted random forest is used to residualize y before the unassigned pool stability step, reducing confounding from correlated modules. Default FALSE.
cfres_n_folds	Number of cross fitting folds for cfRes residualization. Default 5.
cfres_ntree	Trees per fold forest in cfRes residualization. Default 300.
cfres_scope	Scope of cfRes residualization. "unassigned" (default) applies cfRes only to the unassigned pool. "all_modules" applies per module cfRes to every module (removes the contribution of all other modules before that module's stability run).
early_stop	Logical. If TRUE, stop the replicate loop once selection frequencies converge. Default FALSE.
early_stop_tol	Convergence tolerance for early_stop: stop when the largest change in any feature's frequency between batches falls below this. Default 0.01.

`early_stop_check_every`
: Replicates per batch. Convergence is checked between batches. Default `10`.

Value

An object of class `select_control`.

See Also

`screen_control`, `mf`

Examples

```
select_control()
# cross fitted residualization of the outcome, with a stricter threshold
select_control(n_reps = 20, pi_thr = 0.7, use_cfres = TRUE)
```

---

wdi *World Development Indicators (2020)*

Description

A list containing World Bank development indicators and life expectancy for 181 countries in 2020.

Usage

```
data(wdi)
```

Format

A list with three elements:

X A numeric matrix with 181 rows (countries) and 501 columns of World Bank development indicators for 2020, filtered to those present for at least 80% of countries and with residual gaps median imputed. Row names are country short names.

y A numeric vector of length 181 of life expectancy at birth (total, in years). All life table derived series (life expectancy, mortality, survival, and death counts) were removed from `X`.

labels A named character vector mapping each indicator code in `colnames(X)` to its human readable World Bank indicator name.

License

World Bank datasets are published under the Creative Commons Attribution 4.0 International license (CC BY 4.0, https://datacatalog.worldbank.org/public-licenses). Credit: World Bank, World Development Indicators. Changes made: the data were subset, filtered and median imputed as described above. Some indicators are compiled by the World Bank from other agencies, so check the original source before redistributing them separately.

Source

The World Development Indicators bulk database (https://databank.worldbank.org/). See `data-raw/prep_wdi.R` for the exact download and preprocessing steps.

References

World Bank. *World Development Indicators*. Washington, DC: The World Bank Group.

`WGCNA_control` *WGCNA Parameter Organization*

Description

Creates a parameter object for the WGCNA co expression network step used by `wmf`.

Usage

```
WGCNA_control(power = 6, min_features = 20, ...)
```

Arguments

`power`	Soft thresholding power passed to `blockwiseModules`. Default `6`.
`min_features`	Minimum module size, passed to `blockwiseModules` as `minModuleSize`. Features in clusters smaller than this are left unassigned. Default `20`.
`...`	Additional arguments forwarded to `blockwiseModules`.

Value

An object of class `WGCNA_control`.

Examples

```
WGCNA_control()
# extra arguments are passed on to WGCNA::blockwiseModules()
WGCNA_control(power = 8, min_features = 30, deepSplit = 2)
```

wmf *Mossy Forest with WGCNA Module Detection*

Description

Runs a Weighted Gene Co expression Network Analysis (WGCNA) to derive module membership, then passes the result to `mf`.

Usage

```
wmf(
  X,
  y,
  WGCNA_params = WGCNA_control(power = 6),
  screen_params = screen_control(),
  select_params = select_control(),
  final_ntree = 500L,
  nodesize = NULL,
  num_processors = 1L,
  max_modsize = 5000L,
  verbose = 1L,
  seed = as.integer(Sys.time()),
  backend = "python",
  r_shap = "permutation",
  compute_interactions = TRUE
)
```

Arguments

`X`	A data frame where each column is a feature.
`y`	Response vector (numeric for regression, factor for classification).
`WGCNA_params`	WGCNA parameters. See `WGCNA_control`.
`screen_params`	Screening parameters. See `screen_control`.
`select_params`	Selection parameters. See `select_control`.
`final_ntree`	Number of trees in the final random forest. Default `500`.
`nodesize`	Minimum terminal node size. Defaults to `1` for classification and `5` for regression.
`num_processors`	Number of parallel threads for random forests. Default `1`.
`max_modsize`	Maximum module size before blockwise pre reduction is applied (Python backend only). Default `5000`.
`verbose`	Verbose level. `0` = silent; `1` = progress messages; `2` = detailed output. Default `1`.
`seed`	Integer random seed for reproducibility. Defaults to current system time.

backend Computation backend. "python" (default) uses a Python scikit-learn / SHAP engine via **reticulate** and provides exact TreeSHAP values plus an interaction matrix. "R" uses a R **ranger** implementation, no Python installation required. Choose the final importance method with r_shap.

r_shap Final SHAP / importance method used when backend = "R". Ignored for the Python backend. One of:

"permutation" Ranger permutation variable importance (default). No extra packages needed. Returns a scalar VIM per feature; per observation SHAP values and SHAP based plots are unavailable.

"treeshap" Exact TreeSHAP values via the R **treeshap** package. Requires **treeshap**. Also computes a $p \times p$ interaction matrix, enabling detect_interaction() and plot_potential_interactions().

compute_interactions Logical. If TRUE (default), the TreeSHAP interaction matrix is computed by the Python backend and by the pure R backend with r_shap = "treeshap". Set FALSE to skip it (faster); detect_interaction() and plot_potential_interactions() then have nothing to use. It has no effect with r_shap = "permutation", which never computes interactions.

Value

An object of class mossy_forest with an additional $WGCNA_object slot containing the raw WGCNA output.

References

Zhang, B., & Horvath, S. (2005). A general framework for weighted gene co expression network analysis. *Statistical Applications in Genetics and Molecular Biology*, 4(1), Article 17.

Examples

```
# Two blocks of correlated features; WGCNA finds the modules
set.seed(1)
n <- 30
z1 <- rnorm(n); z2 <- rnorm(n)
block <- function(k, zz) as.data.frame(replicate(k, zz + rnorm(n, sd = 0.4)))
X <- cbind(block(10, z1), block(10, z2))
colnames(X) <- paste0("g", seq_len(ncol(X)))
y <- z1 - z2 + rnorm(n, sd = 0.3)
fit <- wmf(X, y, WGCNA_params = WGCNA_control(power = 6, min_features = 6),
           screen_params = screen_control(min_ntree = 30),
           select_params = select_control(n_reps = 3, min_ntree = 30),
           final_ntree = 30, backend = "R", verbose = 0, seed = 1)
table(fit$WGCNA_object$colors)
```

Index

∗ **datasets**
leukemia, 4
wdi, 26

blockwiseModules, *27*

detect_interaction, 2

leukemia, 4

mf, 5, *7*, *16*, *23, 24*, *26*, *28*
mf_setup, 7
mossy_forest, *6*, *29*

plot_decisions, 8
plot_elbow, 9
plot_force, 10
plot_importance, 12
plot_modules, 13
plot_potential_interactions, 14
plot_signed_importance, 15
plot_stability, 17
plot_stability_elbow, 18
plot_waterfall, 20
predict.mossy_forest, 21
predict.ranger, *22*
print.mossy_forest, 22

screen_control, *5*, 23, *25, 26*, *28*
select_control, *5*, *24*, 24, *28*
shapviz, *10*, *12*, *20*

wdi, 26
WGCNA_control, 27, *28*
wmf, *27*, 28