Package: n4m (via r-universe)

November 3, 2026

Type Package

Title Portable Partial Least Squares and NIRS Engine

Version 1.3.14

Date 2026-10-06

Description Implements a portable Partial Least Squares (PLS) and Near-Infrared Spectroscopy (NIRS) engine. Provides fit/predict wrappers for the shipped PLS regression solvers (NIPALS, SIMPLS, SVD, kernel, wide-kernel, orthogonal-scores, power, randomized SVD, PCR), variants (sparse SIMPLS, CPPLS, weighted, robust, ridge, continuum, multi-block, GLM, MIR), adaptive AOM-PLS and POP-PLS operator selection, variable-selection methods (SPA, CARS, GA, random frog, stability selection, VIP), diagnostics (Hotelling T2, Q residuals, DModX), and calibration transfer (PDS, DS). The 'C++17' implementation is vendored and compiled from source at install time; no external system libraries are required.

License CeCILL-2.1 | AGPL (>= 3)

URL https://github.com/GBeurier/nirs4all-methods

BugReports https://github.com/GBeurier/nirs4all-methods/issues

SystemRequirements C++17, GNU make

NeedsCompilation yes

Encoding UTF-8

Imports stats

Suggests testthat (>= 3.0.0), jsonlite, knitr, rmarkdown, markdown, pls

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs make

Repository https://gbeurier.r-universe.dev

Date/Publication 2026-11-03 01:36:28 UTC

RemoteUrl https://github.com/GBeurier/nirs4all-methods

RemoteRef main

RemoteSha 6a6ba8cf5e4930953d2fc10c80748547805fc87d

RemoteSubdir bindings/r/n4m

Contents

aom_pls . . . 5
aom_preprocess . . . 6
approximate_press . . . 6
area_normalization_transform . . . 7
bagging_pls . . . 8
bagging_pls_fit . . . 8
bipls_select . . . 9
boosting_pls . . . 9
boosting_pls_fit . . . 10
bve_select . . . 11
cars_select . . . 11
coef.n4m_fit . . . 12
coefficient_select . . . 12
continuum_regression . . . 13
continuum_regression_fit . . . 13
cppls . . . 14
cppls_fit . . . 15
detrend_transform . . . 15
di_pls . . . 16
di_pls_fit . . . 16
ds_fit . . . 17
ecr . . . 17
ecr_fit . . . 18
emcuve_select . . . 19
emsc_fit . . . 19
emsc_transform . . . 20
fused_sparse_pls_fit . . . 20
ga_select . . . 21
gpr_pls_fit . . . 22
group_sparse_pls_fit . . . 22
interval_select . . . 23
ipw_select . . . 23
irf_select . . . 24
iriv_select . . . 25
kennard_stone_split . . . 26
kernel_pls_fit . . . 26
local_snv_transform . . . 27
lw_pls_fit . . . 28

mb_pls . . . 28
mb_pls_fit . . . 29
mir_pls . . . 29
mir_pls_fit . . . 30
missing_aware_nipals . . . 30
missing_aware_nipals_fit . . . 31
msc_fit . . . 31
msc_transform . . . 32
n_pls_fit . . . 32
n4m_abi_version . . . 33
n4m_affine_fit . . . 33
n4m_affine_model_export . . . 34
n4m_affine_supported_methods . . . 34
n4m_augmentation_apply . . . 35
n4m_estimator_role_constructors . . . 35
n4m_estimator_roles . . . 57
n4m_fit . . . 59
n4m_manifest . . . 60
n4m_method . . . 60
n4m_model_descriptor . . . 62
n4m_model_export . . . 62
n4m_model_import . . . 63
n4m_model_import_linear_predictor . . . 63
n4m_model_inspect . . . 64
n4m_model_pipeline_info . . . 64
n4m_multimodal_classifier . . . 65
n4m_multimodal_pipeline . . . 66
n4m_optimizer . . . 67
n4m_predict . . . 69
n4m_preprocess_fit . . . 70
n4m_procedures . . . 71
n4m_role_pipeline . . . 72
n4m_splitter_run . . . 74
n4m_version . . . 75
o2pls . . . 75
o2pls_fit . . . 76
on_pls_fit . . . 76
one_se_rule . . . 77
opls . . . 77
pds_fit . . . 78
pls . . . 78
pls_cox_fit . . . 80
pls_diagnostics . . . 80
pls_glm . . . 81
pls_glm_fit . . . 81
pls_lda_fit . . . 82
pls_logistic_fit . . . 82
pls_mdatools . . . 83

pls_monitoring . . . 84
pls_qda_fit . . . 85
plsr . . . 85
predict.n4m_fit . . . 88
predict.n4m_method_fit . . . 88
pso_select . . . 89
random_frog_select . . . 90
random_subspace_pls . . . 91
random_subspace_pls_fit . . . 92
randomization_select . . . 92
recursive_pls . . . 93
recursive_pls_fit . . . 94
rep_select . . . 94
ridge_fit . . . 95
ridge_pls . . . 95
ridge_pls_fit . . . 96
robust_pls . . . 97
robust_pls_fit . . . 97
robust_snv_transform . . . 98
rosa_fit . . . 99
scars_select . . . 99
selectivity_ratio_select . . . 100
shaving_select . . . 100
sipls_select . . . 101
snv_transform . . . 102
so_pls_fit . . . 103
spa_select . . . 103
sparse_pls . . . 104
sparse_pls_da_fit . . . 104
sparse_simpls_fit . . . 105
spectral_encoding . . . 105
st_select . . . 106
stability_select . . . 107
t2_select . . . 107
uve_select . . . 108
vip_select . . . 108
vip_spa_select . . . 109
vissa_select . . . 109
weighted_pls . . . 110
weighted_pls_fit . . . 111
wvc_select . . . 111
wvc_threshold_select . . . 112

Index **113**

aom_pls *AOM-PLS and POP-PLS adaptive operator selection*

Description

`aom_pls()` (alias `aompls()`) runs global Adaptive Operator Mixture PLS selection: one preprocessing operator is selected for the whole PLS model. `pop_pls()` (alias `poppls()`) runs per-component operator selection, where each retained PLS component may use a different preprocessing operator.

Usage

```
aom_pls(X, Y, max_components = 3L, n_operators = 9L, cv = 3L)

aompls(X, Y, max_components = 3L, n_operators = 9L, cv = 3L)

pop_pls(X, Y, max_components = 3L, n_operators = 9L, cv = 3L)

poppls(X, Y, max_components = 3L, n_operators = 9L, cv = 3L)
```

Arguments

`X`	Numeric matrix of spectra, with rows as samples and columns as wavelengths.
`Y`	Numeric vector or matrix of responses, with one row per sample.
`max_components`	Maximum number of latent PLS components.
`n_operators`	Number of compact AOM bank operators to expose.
`cv`	Number of contiguous cross-validation folds.

Details

The operator bank mirrors the compact public `nirs4all` AOM bank: identity, Savitzky-Golay smoothers, Savitzky-Golay derivatives, polynomial detrending, and finite difference. The implementation is provided by the native n4m AOM selector ABI.

Value

A named list. Both functions return `predictions`, `operator_kinds`, component-selection diagnostics, cross-validation scores, and selected component metadata.

Examples

```
set.seed(1)
X <- matrix(rnorm(40 * 20), nrow = 40)
Y <- as.numeric(X[, 1] + rnorm(40, sd = 0.1))
fit <- aom_pls(X, Y, max_components = 2L)
dim(fit$predictions)
fit2 <- pop_pls(X, Y, max_components = 2L)
dim(fit2$predictions)
```

aom_preprocess *Adaptive Operator-Mixture preprocessing fit/transform.*

Description

Fits the AOM preprocessing pipeline (operator bank + gating) over ‘X‘ and returns a ‘n4m_method_fit‘ object carrying the selected operators, the per-component gating weights, and the transformed spectra ready to feed into a downstream regression solver.

Usage

```
aom_preprocess(X, Y = NULL, n_operators = 1L, gating_mode = 0L)
```

Arguments

X	numeric matrix of spectra (rows = samples, cols = wavelengths).
Y	optional numeric vector of supervisory targets. When ‘NULL‘, the unsupervised gating path is used (a zero target vector is substituted internally).
n_operators	number of operators in the AOM bank (default ‘1L‘).
gating_mode	integer code selecting the gating strategy: ‘0L‘ = hard-select per component, ‘1L‘ = soft-mixture (default ‘0L‘).

Value

A ‘n4m_method_fit‘ object. Use ‘predict()‘ for inference on new spectra and ‘coef()‘ to extract the gating coefficients.

Examples

```
set.seed(1)
X <- matrix(rnorm(40 * 10), nrow = 40)
Y <- as.numeric(X[, 1] + rnorm(40, sd = 0.1))
fit <- aom_preprocess(X, Y)
class(fit)
```

approximate_press *Approximate-PRESS component selection.*

Description

For each component count k in 1..max_components, fits SIMPLS and approximates PRESS via leverage-inflated in-sample residuals.

Usage

```
approximate_press(X, Y, max_components)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`max_components`	Integer; maximum number of components to scan.

Value

A list with ‘press_per_component‘, ‘rmse_per_component‘, ‘selected_n_components‘ (the argmin of PRESS, 1-based as an integer length 1).

`area_normalization_transform` *Area normalization transform*

Description

Normalize spectra using libn4m’s row area operator.

Usage

```
area_normalization_transform(X, method = "sum")
```

Arguments

`X`	Numeric matrix.
`method`	One of `"sum"`, `"abs_sum"`, or `"trapz"`.

Value

Numeric matrix with the same shape as `X`.

bagging_pls *Bagging PLS — formula entry point.*

Description

Bagging PLS — formula entry point.

Usage

```
bagging_pls(
  formula,
  data,
  ncomp = 2L,
  n_estimators = 50L,
  seed = 0L,
  na.action = stats::na.omit
)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
n_estimators	Integer >= 1. Number of bootstrap / boosting / random-subspace estimators.
seed	Integer. Random seed for reproducibility.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

bagging_pls_fit *Bagging PLS (bootstrap aggregation of PLS regressors).*

Description

Bagging PLS (bootstrap aggregation of PLS regressors).

Usage

```
bagging_pls_fit(X, Y, n_components, n_estimators = 50L, seed = 0L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_estimators	Integer >= 1. Number of bootstrap / boosting / random-subspace estimators.
seed	Integer. Random seed for reproducibility.

---

bipls_select *biPLS — backward interval PLS.*

Description

biPLS — backward interval PLS.

Usage

```
bipls_select(X, Y, n_components, interval_width = 10L, min_intervals = 1L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
interval_width	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
min_intervals	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

boosting_pls *Boosting PLS — formula entry point.*

Description

Boosting PLS — formula entry point.

Usage

```
boosting_pls(
  formula,
  data,
  ncomp = 2L,
  n_estimators = 50L,
  learning_rate = 0.1,
  na.action = stats::na.omit
)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
n_estimators	Integer >= 1. Number of bootstrap / boosting / random-subspace estimators.
learning_rate	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

---

boosting_pls_fit *Boosting PLS (stage-wise refit with learning_rate).*

Description

Boosting PLS (stage-wise refit with learning_rate).

Usage

```
boosting_pls_fit(X, Y, n_components, n_estimators = 50L, learning_rate = 0.1)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_estimators	Integer >= 1. Number of bootstrap / boosting / random-subspace estimators.
learning_rate	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.

`bve_select` *BVE-PLS.*

Description

BVE-PLS.

Usage

```
bve_select(X, Y, n_components, n_steps = 10L, min_features = 5L)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_components`	Integer. Number of latent components.
`n_steps`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
`min_features`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

`cars_select` *CARS — Competitive Adaptive Reweighted Sampling.*

Description

Fit-data selector with a built-in 5-fold validation plan (default C-side fallback). For a custom plan, use the lower-level Python binding or extend r_methods.c.

Usage

```
cars_select(X, Y, n_components, n_iterations = 50L, min_features = 5L)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_components`	Integer. Number of latent components.
`n_iterations`	Number of CARS iterations (typical 50-100).
`min_features`	Lower bound on the final subset size.

Value

A list with 'selected_indices' (1-based) and 'best_rmse'.

`coef.n4m_fit` *Extract the regression coefficients of a [pls()]-fitted model.*

Description

Returns the ‘(p x q)‘ coefficient matrix (‘p‘ predictors by ‘q‘ targets) read from the fitted libn4m model via the ‘n4m_model_get_array‘ C ABI (tag ‘N4M_MODEL_COEFFICIENTS‘). Rows are named after the predictors when their names are available from the model terms.

Usage

```
## S3 method for class 'n4m_fit'
coef(object, ...)
```

Arguments

`object`	A ‘n4m_fit‘ returned by [pls()].
`...`	Ignored (for S3 generic compatibility).

Value

A numeric ‘p x q‘ matrix of regression coefficients.

`coefficient_select` *Coefficient-magnitude ranker.*

Description

Coefficient-magnitude ranker.

Usage

```
coefficient_select(model, X, top_k)
```

Arguments

`model`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
`X`	Numeric matrix used for the fit (re-passed for diagnostics).
`top_k`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.

Value

A list with ‘scores‘ (|coef| sums) and ‘selected_indices‘.

`continuum_regression` *Continuum regression — formula entry point.*

Description

Continuum regression — formula entry point.

Usage

```
continuum_regression(
  formula,
  data,
  ncomp = 2L,
  tau = 0.5,
  na.action = stats::na.omit
)
```

Arguments

`formula`	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
`data`	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
`ncomp`	Integer. Number of latent components.
`tau`	Numeric in [0, 1]. Continuum regression mixing parameter.
`na.action`	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

`continuum_regression_fit` *Continuum regression (tau in [0, 1]).*

Description

Continuum regression (tau in [0, 1]).

Usage

```
continuum_regression_fit(X, Y, n_components, tau = 0.5)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
tau	Numeric in [0, 1]. Continuum regression mixing parameter.

---

cppls *Canonical Powered PLS — formula entry point.*

Description

Canonical Powered PLS — formula entry point.

Usage

```
cppls(formula, data, ncomp = 2L, gamma = 0.5, na.action = stats::na.omit)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
gamma	Numeric. CPPLS / kernel band parameter.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

cppls_fit *Canonical Powered PLS fit (Indahl 2005).*

Description

Canonical Powered PLS fit (Indahl 2005).

Usage

```
cppls_fit(X, Y, n_components, gamma = 0.5)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
gamma	Power exponent in [0, 1]. 0 recovers SIMPLS, 1 is fully power-rescaled.

Value

A list with ‘coefficients‘, ‘predictions‘, ‘x_mean‘, ‘y_mean‘, ‘rmse‘.

detrend_transform *Polynomial detrend transform*

Description

Remove a per-spectrum polynomial baseline with libn4m.

Usage

```
detrend_transform(X, polyorder = 1L)
```

Arguments

X	Numeric matrix.
polyorder	Non-negative baseline polynomial order.

Value

Numeric matrix with the same shape as X.

di_pls *Domain-invariant PLS – formula entry point.*

Description

Domain-invariant PLS – formula entry point.

Usage

```
di_pls(
  formula,
  data,
  ncomp = 2L,
  X_target,
  di_lambda = 1,
  na.action = stats::na.omit
)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
X_target	Numeric matrix for the target domain.
di_lambda	Numeric DI-PLS penalty.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model and in-sample predictions.

di_pls_fit *Domain-Invariant PLS (Nikzad-Langerodi 2018).*

Description

Domain-Invariant PLS (Nikzad-Langerodi 2018).

Usage

```
di_pls_fit(X_source, Y_source, n_components, X_target, di_lambda = 1)
```

Arguments

X_source	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
Y_source	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
n_components	Integer. Number of latent components.
X_target	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
di_lambda	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

ds_fit *Direct Standardization (calibration transfer).*

Description

Direct Standardization (calibration transfer).

Usage

```
ds_fit(X_source, X_target)
```

Arguments

X_source	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
X_target	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

ecr *Elastic Component Regression — formula entry point.*

Description

Elastic Component Regression — formula entry point.

Usage

```
ecr(formula, data, ncomp = 2L, alpha = 0.5, na.action = stats::na.omit)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
alpha	Numeric in [0, 1]. Elastic-net / penalty mixing parameter.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

---

ecr_fit *Elastic Component Regression (Liu 2009/2010).*

Description

Elastic Component Regression (Liu 2009/2010).

Usage

```
ecr_fit(X, Y, n_components, alpha = 0.5)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
alpha	Numeric in [0, 1]. Elastic-net / penalty mixing parameter.

emcuve_select *EMCUVE — ensemble Monte Carlo UVE.*

Description

EMCUVE — ensemble Monte Carlo UVE.

Usage

```
emcuve_select(
  X,
  Y,
  n_components,
  noise_features = NULL,
  noise_seed = 0L,
  n_ensembles = 5L,
  vote_threshold = 0.5
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
noise_features	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
noise_seed	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
n_ensembles	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
vote_threshold	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

emsc_fit *Fit Extended Multiplicative Scatter Correction*

Description

Fits the native n4m EMSC kernel on training spectra. Save the reference and pass it with the same degree to `emsc_transform()` for validation or future samples.

Usage

```
emsc_fit(X, degree = 2L)
```

Arguments

X	Finite numeric training matrix.
degree	Positive polynomial degree; at least degree + 2 features are required.

Value

Numeric fitted reference spectrum of length ncol(X).

---

emsc_transform *Apply fitted Extended Multiplicative Scatter Correction*

Description

Applies native n4m EMSC using a saved training reference.

Usage

```
emsc_transform(X, reference, degree = 2L)
```

Arguments

X	Finite numeric matrix in training feature order.
reference	Reference returned by emsc_fit().
degree	Same polynomial degree used at fit time.

Value

Numeric matrix with the same dimensions as X.

---

fused_sparse_pls_fit *Fused-sparse PLS (L1 + adjacent-coef smoothing).*

Description

Fused-sparse PLS (L1 + adjacent-coef smoothing).

Usage

```
fused_sparse_pls_fit(
  X,
  Y,
  n_components,
  l1_lambda = 0.05,
  fusion_lambda = 0.05
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
l1_lambda	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
fusion_lambda	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

ga_select *GA-PLS — genetic algorithm variable selection.*

Description

GA-PLS — genetic algorithm variable selection.

Usage

```
ga_select(
  X,
  Y,
  n_components,
  n_generations = 50L,
  population_size = 50L,
  min_features = NULL,
  max_features = NULL,
  mutation_rate = 0.01,
  seed = 0L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_generations	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
population_size	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
min_features	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
max_features	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

mutation_rate	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
seed	Integer. Random seed for reproducibility.

---

gpr_pls_fit *Gaussian Process Regression on PLS scores (single-target Y).*

Description

Gaussian Process Regression on PLS scores (single-target Y).

Usage

```
gpr_pls_fit(
  X,
  Y,
  n_components,
  length_scale = 1,
  noise_level = 1e-04,
  seed = 0L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
length_scale	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
noise_level	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
seed	Integer. Random seed for reproducibility.

---

group_sparse_pls_fit *Group-sparse PLS (group L1 across feature groups).*

Description

Group-sparse PLS (group L1 across feature groups).

Usage

```
group_sparse_pls_fit(X, Y, n_components, group_assignment, group_lambda = 0.05)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
group_assignment	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
group_lambda	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

interval_select *Interval selector (iPLS).*

Description

Interval selector (iPLS).

Usage

```
interval_select(X, Y, n_components, interval_width = 10L, step = 1L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
interval_width	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
step	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

ipw_select *IPW-PLS.*

Description

IPW-PLS.

Usage

```
ipw_select(
  X,
  Y,
  n_components,
  n_iterations = 10L,
  top_k = 10L,
  damping = 0.5,
  weight_floor = 1e-06
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_iterations	Integer >= 1. Number of outer-loop iterations.
top_k	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
damping	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
weight_floor	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

irf_select *IRF — Interval Random Frog.*

Description

IRF — Interval Random Frog.

Usage

```
irf_select(
  X,
  Y,
  n_components,
  n_iterations = 100L,
  window_size = 10L,
  initial_intervals = 10L,
  top_k = 5L,
  seed = 0L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_iterations	Integer >= 1. Number of outer-loop iterations.
window_size	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
initial_intervals	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
top_k	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
seed	Integer. Random seed for reproducibility.

---

iriv_select *IRIV — Iteratively Retains Informative Variables.*

Description

IRIV — Iteratively Retains Informative Variables.

Usage

```
iriv_select(X, Y, n_components, max_rounds = 20L, seed = 0L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
max_rounds	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
seed	Integer. Random seed for reproducibility.

kennard_stone_split *Kennard-Stone train/test split*

Description

Thin R wrapper over libn4m's Kennard-Stone splitter C ABI. The R layer performs matrix layout conversion only; all split logic is delegated to libn4m.

Usage

```
kennard_stone_split(X, test_size = 0.25, zero_based = FALSE)
```

Arguments

X	Numeric matrix.
test_size	Fraction of samples assigned to the test set.
zero_based	Logical; return 0-based indices when TRUE. The default FALSE returns idiomatic 1-based R indices.

Value

Named list with integer vectors train and test.

kernel_pls_fit *Non-linear kernel PLS (Rosipal & Trejo 2001).*

Description

Non-linear kernel PLS (Rosipal & Trejo 2001).

Usage

```
kernel_pls_fit(
  X,
  Y,
  n_components,
  kernel_type = 1L,
  gamma = 0,
  coef0 = 1,
  degree = 3L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
kernel_type	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
gamma	Numeric. CPPLS / kernel band parameter.
coef0	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
degree	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

local_snv_transform *Local Standard Normal Variate transform*

Description

Apply libn4m's sliding-window SNV to each spectrum.

Usage

```
local_snv_transform(X, window = 11L, pad_mode = "reflect", constant_value = 0)
```

Arguments

X	Numeric matrix.
window	Odd sliding window length.
pad_mode	One of "reflect", "edge", or "constant".
constant_value	Padding value in constant mode.

Value

Numeric matrix with the same shape as X.

lw_pls_fit *Locally-weighted PLS (Næs & Centner 1998).*

Description

Locally-weighted PLS (Næs & Centner 1998).

Usage

```
lw_pls_fit(X, Y, n_components, n_neighbors = 30L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_neighbors	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

mb_pls *Multi-block PLS — formula entry point.*

Description

Multi-block PLS — formula entry point.

Usage

```
mb_pls(formula, data, ncomp = 2L, block_sizes, na.action = stats::na.omit)
```

Arguments

formula	A two-sided formula (e.g. 'y ~ .' or 'y ~ x1 + x2 + x3'). Response on the left, predictors on the right.
data	A 'data.frame' (or anything 'as.data.frame'-coercible) containing the response and predictor columns referenced by 'formula'.
ncomp	Integer. Number of latent components.
block_sizes	Integer vector summing to the number of predictors.
na.action	What to do with 'NA's. Default: 'na.omit'.

mb_pls_fit *Multi-block PLS (block-weighted SIMPLS).*

Description

Multi-block PLS (block-weighted SIMPLS).

Usage

```
mb_pls_fit(X, Y, n_components, block_sizes)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
block_sizes	Integer vector; must sum to ncol(X).

Value

A list with ‘coefficients‘, ‘predictions‘, ‘x_mean‘, ‘x_scale‘, ‘intercept‘, ‘block_weights‘, ‘rmse‘.

mir_pls *MIR-PLS — formula entry point.*

Description

MIR-PLS — formula entry point.

Usage

```
mir_pls(formula, data, ncomp = 2L, na.action = stats::na.omit)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

mir_pls_fit *MIR-PLS — Multivariate Inverse Regression PLS.*

Description

MIR-PLS — Multivariate Inverse Regression PLS.

Usage

```
mir_pls_fit(X, Y, n_components)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.

Value

A list with ‘coefficients‘, ‘predictions‘, ‘x_mean‘, ‘y_mean‘, ‘rmse‘.

missing_aware_nipals *Missing-aware NIPALS — formula entry point.*

Description

Missing-aware NIPALS — formula entry point.

Usage

```
missing_aware_nipals(formula, data, ncomp = 2L, na.action = stats::na.pass)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

`missing_aware_nipals_fit` *Missing-aware NIPALS PLS (Nelson 1996).*

Description

Missing-aware NIPALS PLS (Nelson 1996).

Usage

```
missing_aware_nipals_fit(X, Y, n_components)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_components`	Integer. Number of latent components.

`msc_fit` *Fit native Multiplicative Scatter Correction*

Description

Fit libn4m MSC on training spectra and return only the learned reference spectrum. Do not refit on validation or prediction samples.

Usage

```
msc_fit(X)
```

Arguments

`X`	Finite numeric training matrix with at least two features.

Value

Numeric fitted reference spectrum, length `ncol(X)`.

`msc_transform` *Apply a fitted MSC reference to spectra*

Description

Restore a libn4m MSC handle from the supplied training reference and transform new spectra without learning from them.

Usage

```
msc_transform(X, reference)
```

Arguments

`X`	Finite numeric matrix with the same feature order as training.
`reference`	Numeric reference from `msc_fit()`.

Value

Numeric matrix with the same shape as `X`.

`n_pls_fit` *N-PLS (3-way tensor) regression. 'X_flat' is the flattened (n, mode_j*mode_k) matrix.*

Description

N-PLS (3-way tensor) regression. 'X_flat' is the flattened (n, mode_j*mode_k) matrix.

Usage

```
n_pls_fit(X_flat, Y, n_components, mode_j, mode_k)
```

Arguments

`X_flat`	Numeric matrix. The flattened 3-way design tensor (rows = samples, cols = mode_j * mode_k).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_components`	Integer. Number of latent components.
`mode_j`	Integer. Size of the first non-sample tensor mode.
`mode_k`	Integer. Size of the second non-sample tensor mode.

n4m_abi_version *Loaded ABI version as an integer vector (major, minor, patch).*

Description

Loaded ABI version as an integer vector (major, minor, patch).

Usage

```
n4m_abi_version()
```

Value

An integer vector of length 3.

n4m_affine_fit *Fit and predict a native affine MethodResult regressor*

Description

Fits a MethodResult through libn4m and converts a natively marked affine result to a native prediction model. Held-out predictions use libn4m.

Usage

```
n4m_affine_fit(method, X, Y, n_components = 2L, params = list())
## S3 method for class 'n4m_affine_fit'
predict(object, newdata, ...)
```

Arguments

method	A qualified n4m MethodResult regression method.
X	A finite numeric samples-by-features matrix.
Y	A finite numeric response vector or samples-by-targets matrix.
n_components	A positive component count; ridge ignores it.
params	A named list of method-specific parameters.
object	An n4m_affine_fit object.
newdata	A finite numeric matrix in the fitted feature order.
...	Unused.

Details

Only methods listed by n4m_affine_supported_methods() are currently qualified for this conversion. The native converter refuses unmarked results.

Value

`n4m_affine_fit()` returns an S3 object with a native prediction model. `predict()` returns a numeric vector for one target or a matrix for multiple targets.

`n4m_affine_model_export` *Export a fitted affine MethodResult predictor*

Description

Exports the fitted native affine model. The bytes preserve prediction, not the original fitting algorithm or its parameters.

Usage

```
n4m_affine_model_export(object)
```

Arguments

`object`	An `n4m_affine_fit` object.

Value

Portable N4MM raw bytes for native prediction.

`n4m_affine_supported_methods` *List MethodResult regressors marked for native affine conversion*

Description

Returns the MethodResult producers currently qualified for conversion to a native held-out prediction model.

Usage

```
n4m_affine_supported_methods()
```

Value

A character vector of method names.

n4m_augmentation_apply

Apply a seeded native X-only training augmentation

Description

One-shot X-only augmentation by an existing seeded libn4m kernel. There is no fitted state, held-out transform, or paired Y output. Mixup and LocalMixup are excluded because their C kernels do not expose pairing or weights needed to keep labels aligned. Use only on training spectra.

Usage

```
n4m_augmentation_apply(kind, X, params = numeric(), seed = 0)
```

Arguments

kind	One of the 22 supported native augmentation names.
X	Finite numeric matrix of training spectra.
params	Finite positional parameter vector in the order of the corresponding libn4m augmenter create function.
seed	Exact nonnegative integer seed up to 2^53-1.

Value

A new numeric matrix with the same shape as X.

n4m_estimator_role_constructors

Constructors of the generic native estimator roles

Description

One constructor per native method of the n4m manifest:

- n4m_aom_chain_fixed_fit: aom_pop.aom_chain_fixed_fit (regressor)
- n4m_aom_chain_ridge_pls: aom_pop.aom_chain_ridge_pls (regressor)
- n4m_aom_chain_sweep: aom_pop.aom_chain_sweep (regressor)
- n4m_aom_pls: aom_pop.aom_pls (regressor)
- n4m_aom_pls_superblock: aom_pop.aom_pls_superblock (regressor)
- n4m_aom_preprocessing: aom_pop.aom_preprocessing (transformer)
- n4m_aom_ridge_pls_superblock: aom_pop.aom_ridge_pls_superblock (regressor)
- n4m_aom_sweep: aom_pop.aom_sweep (regressor)

- n4m_calibration: aom_pop.calibration (regressor)
- n4m_linear_stack_compress: aom_pop.linear_stack_compress (generic)
- n4m_operator_pls_stack: aom_pop.operator_pls_stack (regressor)
- n4m_pop_pls: aom_pop.pop_pls (regressor)
- n4m_ridge_active_superblock: aom_pop.ridge_active_superblock (regressor)
- n4m_ridge_blender: aom_pop.ridge_blender (regressor)
- n4m_ridge_global: aom_pop.ridge_global (regressor)
- n4m_ridge_mkl_superblock: aom_pop.ridge_mkl_superblock (regressor)
- n4m_ridge_superblock: aom_pop.ridge_superblock (regressor)
- n4m_robust_hpo: aom_pop.robust_hpo (regressor)
- n4m_linear_drift: augmentation.drift.linear_drift (augmenter)
- n4m_path_length: augmentation.drift.path_length (augmenter)
- n4m_poly_drift: augmentation.drift.poly_drift (augmenter)
- n4m_detector_rolloff: augmentation.edge_artifacts.detector_rolloff (augmenter)
- n4m_edge_artifacts: augmentation.edge_artifacts.edge_artifacts (augmenter)
- n4m_edge_curvature: augmentation.edge_artifacts.edge_curvature (augmenter)
- n4m_stray_light: augmentation.edge_artifacts.stray_light (augmenter)
- n4m_truncated_peak: augmentation.edge_artifacts.truncated_peak (augmenter)
- n4m_moisture: augmentation.environmental.moisture (augmenter)
- n4m_temperature: augmentation.environmental.temperature (augmenter)
- n4m_local_mixup: augmentation.mixup.local_mixup (augmenter)
- n4m_mixup: augmentation.mixup.mixup (augmenter)
- n4m_gaussian_noise: augmentation.noise.gaussian_noise (augmenter)
- n4m_hetero_noise: augmentation.noise.hetero_noise (augmenter)
- n4m_multiplicative_noise: augmentation.noise.multiplicative_noise (augmenter)
- n4m_spike_noise: augmentation.noise.spike_noise (augmenter)
- n4m_random_x_op: augmentation.random.random_x_op (augmenter)
- n4m_rotate_translate: augmentation.random.rotate_translate (augmenter)
- n4m_batch_effect: augmentation.scattering.batch_effect (augmenter)
- n4m_dead_band: augmentation.scattering.dead_band (augmenter)
- n4m_emsc_distort: augmentation.scattering.emsc_distort (augmenter)
- n4m_instrument_broaden: augmentation.scattering.instrument_broaden (augmenter)
- n4m_particle_size: augmentation.scattering.particle_size (augmenter)
- n4m_scatter_sim_msc: augmentation.scattering.scatter_sim_msc (augmenter)
- n4m_band_mask: augmentation.spectral.band_mask (augmenter)
- n4m_band_perturb: augmentation.spectral.band_perturb (augmenter)
- n4m_channel_dropout: augmentation.spectral.channel_dropout (augmenter)

- n4m_gauss_jitter: augmentation.spectral.gauss_jitter (augmenter)
- n4m_local_clip: augmentation.spectral.local_clip (augmenter)
- n4m_magnitude_warp: augmentation.spectral.magnitude_warp (augmenter)
- n4m_unsharp_mask: augmentation.spectral.unsharp_mask (augmenter)
- n4m_spline_curve_simplification: augmentation.splines.spline_curve_simplification (augmenter)
- n4m_spline_smoothing: augmentation.splines.spline_smoothing (augmenter)
- n4m_spline_x_perturbations: augmentation.splines.spline_x_perturbations (augmenter)
- n4m_spline_x_simplification: augmentation.splines.spline_x_simplification (augmenter)
- n4m_spline_y_perturbations: augmentation.splines.spline_y_perturbations (augmenter)
- n4m_local_warp: augmentation.wavelength.local_warp (augmenter)
- n4m_wavelength_shift: augmentation.wavelength.wavelength_shift (augmenter)
- n4m_wavelength_stretch: augmentation.wavelength.wavelength_stretch (augmenter)
- n4m_approximate_press: diagnostics.approximate_press (generic)
- n4m_local_density: diagnostics.local_density (generic)
- n4m_model_selection: diagnostics.model_selection (generic)
- n4m_pca_reference_qc: diagnostics.pca_reference_qc (generic)
- n4m_pls_diagnostics: diagnostics.pls_diagnostics (generic)
- n4m_pls_monitoring: diagnostics.pls_monitoring (generic)
- n4m_regression_metrics: diagnostics.regression_metrics (generic)
- n4m_spectral_events_qc: diagnostics.spectral_events_qc (generic)
- n4m_spectral_noise_proxy: diagnostics.spectral_noise_proxy (generic)
- n4m_spectral_reference_qc: diagnostics.spectral_reference_qc (generic)
- n4m_correlation: filters.correlation (selector)
- n4m_high_leverage: filters.high_leverage (sample_filter)
- n4m_spectral_quality: filters.spectral_quality (sample_filter)
- n4m_variance: filters.variance (selector)
- n4m_x_outlier: filters.x_outlier (sample_filter)
- n4m_y_outlier: filters.y_outlier (sample_filter)
- n4m_pls_lda: models.classification.pls_lda (classifier)
- n4m_pls_logistic: models.classification.pls_logistic (classifier)
- n4m_pls_qda: models.classification.pls_qda (classifier)
- n4m_bagging_pls: models.ensembles.bagging_pls (regressor)
- n4m_boosting_pls: models.ensembles.boosting_pls (regressor)
- n4m_random_subspace_pls: models.ensembles.random_subspace_pls (regressor)

- n4m_pls_cox: models.heads.pls_cox (regressor)
- n4m_pls_glm: models.heads.pls_glm (regressor)
- n4m_lw_pls: models.local.lw_pls (regressor)
- n4m_mb_pls: models.multiblock.mb_pls (regressor)
- n4m_mir_pls: models.multiblock.mir_pls (regressor)
- n4m_o2pls: models.multiblock.o2pls (regressor)
- n4m_on_pls: models.multiblock.on_pls (transformer)
- n4m_rosa: models.multiblock.rosa (regressor)
- n4m_so_pls: models.multiblock.so_pls (regressor)
- n4m_cppls: models.pls.cppls (regressor)
- n4m_kernel: models.pls.kernel (regressor)
- n4m_pcr: models.pls.pcr (transformer, regressor)
- n4m_pls_fit_simple: models.pls.pls_fit_simple (transformer, regressor)
- n4m_pls_regression: models.pls.pls_regression (transformer, regressor)
- n4m_continuum_regression: models.regularized.continuum_regression (regressor)
- n4m_ridge: models.regularized.ridge (regressor)
- n4m_ridge_pls: models.regularized.ridge_pls (regressor)
- n4m_robust_pls: models.regularized.robust_pls (regressor)
- n4m_weighted_pls: models.regularized.weighted_pls (regressor)
- n4m_fused_sparse_pls: models.sparse.fused_sparse_pls (regressor)
- n4m_group_sparse_pls: models.sparse.group_sparse_pls (regressor)
- n4m_sparse_pls_da: models.sparse.sparse_pls_da (classifier)
- n4m_sparse_simpls: models.sparse.sparse_simpls (regressor)
- n4m_ecr: models.specialized.ecr (regressor)
- n4m_gpr_pls: models.specialized.gpr_pls (regressor)
- n4m_missing_aware_nipals: models.specialized.missing_aware_nipals (regressor)
- n4m_recursive: models.specialized.recursive (transformer, regressor)
- n4m_tensor_pls: models.specialized.tensor_pls (regressor)
- n4m_di_pls: models.transfer.di_pls (regressor)
- n4m_ds: models.transfer.ds (transformer)
- n4m_pds: models.transfer.pds (transformer)
- n4m_cow_align: preprocessing.alignment.cow_align (transformer)
- n4m_dtw_align: preprocessing.alignment.dtw_align (transformer)
- n4m_icoshift_align: preprocessing.alignment.icoshift_align (transformer)
- n4m_xcorr_align: preprocessing.alignment.xcorr_align (transformer)
- n4m_airpls: preprocessing.baselines.airpls (transformer)
- n4m_arpls: preprocessing.baselines.arpls (transformer)

- n4m_asls: preprocessing.baselines.asls (transformer)
- n4m_beads: preprocessing.baselines.beads (transformer)
- n4m_detrend: preprocessing.baselines.detrend (transformer)
- n4m_iasls: preprocessing.baselines.iasls (transformer)
- n4m_imodpoly: preprocessing.baselines.imodpoly (transformer)
- n4m_modpoly: preprocessing.baselines.modpoly (transformer)
- n4m_rolling_ball: preprocessing.baselines.rolling_ball (transformer)
- n4m_saps: preprocessing.baselines.saps (transformer)
- n4m_snip: preprocessing.baselines.snip (transformer)
- n4m_derivate: preprocessing.derivatives.derivate (transformer)
- n4m_first_derivative: preprocessing.derivatives.first_derivative (transformer)
- n4m_norris_williams: preprocessing.derivatives.norris_williams (transformer)
- n4m_savitzky_golay: preprocessing.derivatives.savitzky_golay (transformer)
- n4m_second_derivative: preprocessing.derivatives.second_derivative (transformer)
- n4m_flexible_pca: preprocessing.feature_selection.flexible_pca (transformer)
- n4m_flexible_svd: preprocessing.feature_selection.flexible_svd (transformer)
- n4m_epo: preprocessing.orthogonalization.epo (transformer)
- n4m_osc: preprocessing.orthogonalization.osc (transformer)
- n4m_crop: preprocessing.resampling.crop (transformer)
- n4m_kbins_discretizer: preprocessing.resampling.kbins_discretizer (transformer)
- n4m_range_discretizer: preprocessing.resampling.range_discretizer (transformer)
- n4m_resample_transformer: preprocessing.resampling.resample_transformer (transformer)
- n4m_resampler: preprocessing.resampling.resampler (transformer)
- n4m_baseline: preprocessing.scaling.baseline (transformer)
- n4m_log_transform: preprocessing.scaling.log_transform (transformer)
- n4m_normalize: preprocessing.scaling.normalize (transformer)
- n4m_simple_scale: preprocessing.scaling.simple_scale (transformer)
- n4m_standard_scale: preprocessing.scaling.standard_scale (transformer)
- n4m_area_normalization: preprocessing.scatter.area_normalization (transformer)
- n4m_emsc: preprocessing.scatter.emsc (transformer)
- n4m_local_centering: preprocessing.scatter.local_centering (transformer)
- n4m_local_snv: preprocessing.scatter.local_snv (transformer)
- n4m_localized_msc: preprocessing.scatter.localized_msc (transformer)
- n4m_msc: preprocessing.scatter.msc (transformer)
- n4m_piecewise_msc: preprocessing.scatter.piecewise_msc (transformer)
- n4m_piecewise_snv: preprocessing.scatter.piecewise_snv (transformer)

- n4m_robust_snv: preprocessing.scatter.robust_snv (transformer)
- n4m_snv: preprocessing.scatter.snv (transformer)
- n4m_vsn: preprocessing.scatter.vsn (transformer)
- n4m_weighted_snv: preprocessing.scatter.weighted_snv (transformer)
- n4m_fraction_to_percent: preprocessing.signal_conversion.fraction_to_percent (transformer)
- n4m_from_absorbance: preprocessing.signal_conversion.from_absorbance (transformer)
- n4m_kubelka_munk: preprocessing.signal_conversion.kubelka_munk (transformer)
- n4m_percent_to_fraction: preprocessing.signal_conversion.percent_to_fraction (transformer)
- n4m_to_absorbance: preprocessing.signal_conversion.to_absorbance (transformer)
- n4m_gaussian: preprocessing.smoothing.gaussian (transformer)
- n4m_fck_static: preprocessing.specialized.fck_static (transformer)
- n4m_direct_standardization: preprocessing.transfer.direct_standardization (transformer)
- n4m_piecewise_direct_standardization: preprocessing.transfer.piecewise_direct_standardization (transformer)
- n4m_robust_direct_standardization: preprocessing.transfer.robust_direct_standardization (transformer)
- n4m_slope_bias: preprocessing.transfer.slope_bias (transformer)
- n4m_haar: preprocessing.wavelets.haar (transformer)
- n4m_wavelet: preprocessing.wavelets.wavelet (transformer)
- n4m_wavelet_denoise: preprocessing.wavelets.wavelet_denoise (transformer)
- n4m_wavelet_features: preprocessing.wavelets.wavelet_features (transformer)
- n4m_wavelet_pca: preprocessing.wavelets.wavelet_pca (transformer)
- n4m_wavelet_svd: preprocessing.wavelets.wavelet_svd (transformer)
- n4m_bipls: selection.bipls (selector)
- n4m_bve: selection.bve (selector)
- n4m_cars: selection.cars (selector)
- n4m_emcuve: selection.emcuve (selector)
- n4m_ga: selection.ga (selector)
- n4m_interval: selection.interval (transformer)
- n4m_ipw: selection.ipw (selector)
- n4m_irf: selection.irf (selector)
- n4m_iriv: selection.iriv (selector)
- n4m_pso: selection.pso (selector)
- n4m_random_frog: selection.random_frog (selector)
- n4m_randomization: selection.randomization (selector)

- n4m_rep: selection.rep (selector)
- n4m_scars: selection.scars (selector)
- n4m_shaving: selection.shaving (selector)
- n4m_sipls: selection.sipls (selector)
- n4m_spa: selection.spa (selector)
- n4m_st: selection.st (selector)
- n4m_stability: selection.stability (selector)
- n4m_t2: selection.t2 (selector)
- n4m_uve: selection.uve (selector)
- n4m_variable_select: selection.variable_select (selector)
- n4m_vip_spa: selection.vip_spa (selector)
- n4m_vissa: selection.vissa (selector)
- n4m_wvc: selection.wvc (selector)
- n4m_wvc_threshold: selection.wvc_threshold (selector)
- n4m_binned_strat_group_kfold: splitters.binned_strat_group_kfold (splitter)
- n4m_kbins_stratified: splitters.kbins_stratified (splitter)
- n4m_kennard_stone: splitters.kennard_stone (splitter)
- n4m_kmeans: splitters.kmeans (splitter)
- n4m_split_splitter: splitters.split_splitter (splitter)
- n4m_spxy: splitters.spxy (splitter)
- n4m_spxy_fold: splitters.spxy_fold (splitter)
- n4m_spxy_g_fold: splitters.spxy_g_fold (splitter)
- n4m_systematic_circular: splitters.systematic_circular (splitter)
- n4m_hotelling_t2: utilities.hotelling_t2 (generic)
- n4m_moments: utilities.moments (generic)
- n4m_q_residuals: utilities.q_residuals (generic)
- n4m_ragged_summary: utilities.ragged_summary (generic)
- n4m_signal_type_detector: utilities.signal_type_detector (generic)
- n4m_sweep: utilities.sweep (generic)
- n4m_transfer_metrics: utilities.transfer_metrics (generic)

Usage

```
n4m_aom_chain_fixed_fit(op_kinds = NULL, param_offsets = NULL, chain_params = c(), head = "ridge", para
n4m_aom_chain_ridge_pls(chain_offsets = c(0L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 11L, 13L, 15L), op_kin
n4m_aom_chain_sweep(chain_offsets = NULL, op_kinds = NULL, param_offsets = NULL, chain_params = c(), cv
n4m_aom_pls(max_components = 3L, op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L
n4m_aom_pls_superblock(op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L, 0L, 1L,
n4m_aom_preprocessing(op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L, 0L, 1L, 2
n4m_aom_ridge_pls_superblock(op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L, 0
```

```
n4m_aom_sweep(profile = "compact", cv = 5L, ridge_lambdas = c(0.01, 0.1, 1.0, 10.0), pls_components = c(
n4m_calibration(head = "pls", fast = FALSE, max_components = 25L, alphas = c(), cv = 3L, branches = c(0L,
n4m_linear_stack_compress(base_intercepts = NULL, meta_weights = NULL, meta_intercept = NULL)
n4m_operator_pls_stack(profile = "compact", cv = 5L, components = c(2L, 4L, 8L), alphas = c(0.001, 0.01,
n4m_pop_pls(max_components = 3L, op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L
n4m_ridge_active_superblock(op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L, 0L
n4m_ridge_blender(profile = "compact", cv = 5L, ridge_lambdas = c(0.0001, 0.01, 1.0, 100.0), regularize
n4m_ridge_global(op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L, 0L, 1L, 2L, 4L
n4m_ridge_mkl_superblock(op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L, 0L, 1
n4m_ridge_superblock(op_kinds = c(0L, 7L, 7L, 8L, 8L, 9L, 9L, 10L, 15L), param_offsets = c(0L, 0L, 1L, 2L
n4m_robust_hpo(profile = "compact", cv = 5L, heads = "ridge_pls")
n4m_linear_drift(offset_min = -0.05, offset_max = 0.05, slope_min = -0.01, slope_max = 0.01, seed = NULL
n4m_path_length(path_length_std = 0.05, min_path_length = 0.1, seed = NULL)
n4m_poly_drift(coeff_min = c(-0.01, -0.01, -0.01), coeff_max = c(0.01, 0.01, 0.01), seed = NULL)
n4m_detector_rolloff(detector_model = "generic_nir", effect_strength = 1.0, noise_amplification = 0.02
n4m_edge_artifacts(detector_roll_off = TRUE, stray_light = TRUE, edge_curvature = TRUE, truncated_peak
n4m_edge_curvature(curvature_strength = 0.02, curvature_type = "random", asymmetry = 0.0, edge_focus =
n4m_stray_light(stray_light_fraction = 0.001, edge_enhancement = 2.0, edge_width = 0.1, include_peak_t
n4m_truncated_peak(peak_probability = 0.5, amplitude_min = 0.01, amplitude_max = 0.1, width_min = 50.0,
n4m_moisture(water_activity_delta = 0.0, use_aw_range = FALSE, aw_low = 0.0, aw_high = 1.0, reference_w
n4m_temperature(temperature_delta = 0.0, use_temp_range = FALSE, temp_low = -5.0, temp_high = 5.0, enab
n4m_local_mixup(alpha = 0.2, k_neighbors = 5L, seed = NULL)
n4m_mixup(alpha = 0.2, seed = NULL)
n4m_gaussian_noise(sigma = 0.01, seed = NULL)
n4m_hetero_noise(noise_base = 0.001, noise_signal_dep = 0.01, seed = NULL)
n4m_multiplicative_noise(sigma_gain = 0.01, seed = NULL)
n4m_spike_noise(n_spikes_min = 1L, n_spikes_max = 3L, amplitude_min = -0.1, amplitude_max = 0.1, seed =
n4m_random_x_op(op_kind = "multiply", operator_range_min = 0.97, operator_range_max = 1.03, seed = NULL
n4m_rotate_translate(p_range = 2.0, y_factor = 3.0, seed = NULL)
n4m_batch_effect(offset_std = 0.0, slope_std = 0.0, gain_std = 0.0, variation_scope = "sample", seed = N
n4m_dead_band(n_bands = 1L, width_low = 5L, width_high = 10L, noise_std = 0.05, probability = 0.0, varia
n4m_emsc_distort(mult_low = 0.9, mult_high = 1.1, add_low = -0.05, add_high = 0.05, polynomial_order = 2
n4m_instrument_broaden(fwhm = 5.0, use_fwhm_range = FALSE, fwhm_low = 3.0, fwhm_high = 8.0, variation_s
n4m_particle_size(mean_size_um = 50.0, size_variation_um = 15.0, use_size_range = FALSE, size_range_lo
n4m_scatter_sim_msc(a_low = -0.05, a_high = 0.05, b_low = 0.9, b_high = 1.1, seed = NULL)
n4m_band_mask(n_bands_lo = 1L, n_bands_hi = 3L, bw_lo = 5L, bw_hi = 15L, mode = "zero", seed = NULL)
n4m_band_perturb(n_bands = 3L, bw_lo = 5L, bw_hi = 15L, gain_lo = 0.9, gain_hi = 1.1, offset_lo = -0.01,
n4m_channel_dropout(dropout_prob = 0.05, mode = "zero", seed = NULL)
n4m_gauss_jitter(sigma_lo = 0.5, sigma_hi = 1.5, kernel_width = 9L, seed = NULL)
n4m_local_clip(n_regions = 1L, width_lo = 5L, width_hi = 15L, seed = NULL)
n4m_magnitude_warp(n_control_points = 5L, gain_lo = 0.9, gain_hi = 1.1, seed = NULL)
n4m_unsharp_mask(amount_lo = 0.1, amount_hi = 0.5, sigma = 1.0, kernel_width = 11L, seed = NULL)
n4m_spline_curve_simplification(spline_points = -1L, uniform = FALSE, seed = NULL)
n4m_spline_smoothing(seed = NULL)
n4m_spline_x_perturbations(spline_degree = 3L, perturbation_density = 0.05, perturbation_range_min = -
n4m_spline_x_simplification(spline_points = -1L, uniform = FALSE, seed = NULL)
n4m_spline_y_perturbations(spline_points = -1L, perturbation_intensity = 0.005, seed = NULL)
n4m_local_warp(n_control_points = 5L, max_shift = 1.0, seed = NULL)
```

```
n4m_wavelength_shift(shift_lo = -1.0, shift_hi = 1.0, seed = NULL)
n4m_wavelength_stretch(stretch_lo = 0.99, stretch_hi = 1.01, seed = NULL)
n4m_approximate_press(max_components = 10L)
n4m_local_density(valid_mask = c(), mode = "training", n_neighbors = 20L)
n4m_model_selection()
n4m_pca_reference_qc(valid_mask = c(), n_components = 5L)
n4m_pls_diagnostics(n_components = 2L, center_x = TRUE, scale_x = TRUE, center_y = TRUE, scale_y = TRUE)
n4m_pls_monitoring(n_components = 2L, center_x = TRUE, scale_x = TRUE, center_y = TRUE, scale_y = TRUE,
n4m_regression_metrics()
n4m_spectral_events_qc(valid_mask = c(), axis_basis = "index", axis_values = c(), outlier_threshold = 8
n4m_spectral_noise_proxy(valid_mask = c())
n4m_spectral_reference_qc(valid_mask = c(), reference_mode = "dataset_mean", reference_valid_mask = c(
n4m_correlation(threshold = 0.0, top_k = -1L)
n4m_high_leverage(method = "hat", threshold_multiplier = 2.0, absolute_threshold = NULL, n_components =
n4m_spectral_quality(max_nan_ratio = 0.1, max_zero_ratio = 0.5, min_variance = 1e-08, max_value = NULL,
n4m_variance(threshold = 0.0, top_k = -1L)
n4m_x_outlier(method = "mahalanobis", threshold = NULL, n_components = 0L, contamination = 0.1, seed = N
n4m_y_outlier(method = "iqr", threshold = 1.5, lower_percentile = 1.0, upper_percentile = 99.0)
n4m_pls_lda(n_components = 2L)
n4m_pls_logistic(n_components = 2L, max_iter = 500L)
n4m_pls_qda(n_components = 2L)
n4m_bagging_pls(n_components = 2L, n_estimators = 50L, seed = NULL)
n4m_boosting_pls(n_components = 2L, n_estimators = 50L, learning_rate = 0.1)
n4m_random_subspace_pls(n_components = 2L, n_estimators = 50L, features_per_subspace = 10L, seed = NULL
n4m_pls_cox(n_components = 2L, max_iter = 50L, tol = 1e-10)
n4m_pls_glm(n_components = 2L, family = "gaussian", max_iter = 100L, tol = 1e-10)
n4m_lw_pls(n_components = 2L, n_neighbors = NULL, mode = "weighted")
n4m_mb_pls(n_components = 2L)
n4m_mir_pls(n_components = 2L)
n4m_o2pls(n_predictive = 2L, n_x_orthogonal = 1L, n_y_orthogonal = 1L)
n4m_on_pls(n_joint = 1L, n_unique_per_block = NULL)
n4m_rosa(n_components = 2L)
n4m_so_pls(n_components_per_block = NULL)
n4m_cppls(n_components = 2L, gamma = 0.5)
n4m_kernel(n_components = 2L, kernel = "rbf", gamma = 0.0, coef0 = 1.0, degree = 3L)
n4m_pcr(n_components = 2L, center_x = TRUE, scale_x = TRUE, center_y = TRUE, scale_y = FALSE)
n4m_pls_fit_simple(n_components = 2L)
n4m_pls_regression(n_components = 2L, solver = "nipals", center_x = TRUE, scale_x = TRUE, center_y = TRU
n4m_continuum_regression(n_components = 2L, tau = 0.5)
n4m_ridge(alpha = 1.0, center_x = TRUE, scale_x = TRUE, center_y = TRUE)
n4m_ridge_pls(n_components = 2L, ridge_lambda = 0.1, center_x = TRUE, scale_x = TRUE, center_y = TRUE, s
n4m_robust_pls(n_components = 2L, huber_k = 1.345, max_irls_iter = 5L, center_x = TRUE, scale_x = TRUE,
n4m_weighted_pls(n_components = 2L, center_x = TRUE, scale_x = TRUE, center_y = TRUE, scale_y = TRUE)
n4m_fused_sparse_pls(n_components = 2L, l1_lambda = 0.05, fusion_lambda = 0.05)
n4m_group_sparse_pls(n_components = 2L, group_lambda = 0.05)
n4m_sparse_pls_da(n_components = 2L, sparsity_lambda = 0.05)
n4m_sparse_simpls(n_components = 2L, sparsity_lambda = 0.05)
n4m_ecr(n_components = 2L, alpha = 0.5)
```

```
n4m_gpr_pls(n_components = 2L, length_scale = 1.0, noise_level = 0.001)
n4m_missing_aware_nipals(n_components = 2L)
n4m_recursive(n_components = 2L, window_size = NULL)
n4m_tensor_pls(n_components = 2L, mode_j = NULL, mode_k = NULL)
n4m_di_pls(n_components = 2L, di_lambda = 1.0)
n4m_ds()
n4m_pds(window_half_width = 2L)
n4m_cow_align(reference = c(), interval_size = 32L, max_shift = 5L)
n4m_dtw_align(reference = c())
n4m_icoshift_align(reference = c(), interval_size = 32L, max_shift = 5L)
n4m_xcorr_align(reference = c(), max_shift = 5L)
n4m_airpls(lam = 1000000.0, max_iter = 50L, tol = 0.001)
n4m_arpls(lam = 100000.0, max_iter = 50L, tol = 0.001)
n4m_asls(lam = 1000000.0, p = 0.01, max_iter = 50L, tol = 0.001)
n4m_beads(lam_0 = 100.0, lam_1 = 0.5, lam_2 = 0.5, max_iter = 50L, tol = 0.001)
n4m_detrend(polyorder = 1L)
n4m_iasls(lam = 1000000.0, p = 0.01, lam_1 = 0.0001, polyorder = 2L, diff_order = 2L, max_iter = 50L, tol
n4m_imodpoly(polyorder = 2L, max_iter = 250L, tol = 0.001)
n4m_modpoly(polyorder = 2L, max_iter = 250L, tol = 0.001)
n4m_rolling_ball(half_window = 20L, smooth_half_window = 0L)
n4m_saps(n_components = 5L, score_weight = 1.0, fit_intercept = TRUE, ridge = 0.0)
n4m_snip(max_half_window = 20L)
n4m_derivate(order = 1L, delta = 1.0)
n4m_first_derivative(delta = 1.0, edge_order = 2L)
n4m_norris_williams(segment = 5L, gap = 5L, derivative_order = 1L, delta = 1.0)
n4m_savitzky_golay(window_length = 5L, polyorder = 2L, deriv = 0L, delta = 1.0, mode = "mirror", cval = 0
n4m_second_derivative(delta = 1.0, edge_order = 2L)
n4m_flexible_pca(n_components = 5.0)
n4m_flexible_svd(n_components = 5.0)
n4m_epo(scale = TRUE)
n4m_osc(n_components = 1L, scale = TRUE)
n4m_crop(start = NULL, end = NULL)
n4m_kbins_discretizer(n_bins = 5L, strategy = "uniform")
n4m_range_discretizer(edges = NULL)
n4m_resample_transformer(num_samples = NULL)
n4m_resampler(target_wavelengths = c(), method = "linear", crop_min = 0.0, crop_max = 0.0, use_crop = FA
n4m_baseline()
n4m_log_transform(base = 0.0, offset = 0.0, auto_offset = TRUE, min_value = 1e-08)
n4m_normalize(feature_min = -1.0, feature_max = 1.0)
n4m_simple_scale()
n4m_standard_scale(with_mean = TRUE, with_std = TRUE)
n4m_area_normalization(method = "sum")
n4m_emsc(degree = 2L)
n4m_local_centering()
n4m_local_snv(window = 11L, pad_mode = "reflect", constant_value = 0.0)
n4m_localized_msc(window_size = 32L, reference = c(), eps = 1e-12)
n4m_msc()
n4m_piecewise_msc(window_size = 32L, reference = c(), eps = 1e-12)
```

```
n4m_piecewise_snv(window_size = 32L, ddof = 0L, eps = 1e-12)
n4m_robust_snv(with_center = TRUE, with_scale = TRUE, k = 1.4826)
n4m_snv(with_mean = TRUE, with_std = TRUE, ddof = 0L)
n4m_vsn(eps = 1e-12)
n4m_weighted_snv(weights = c(), ddof = 0L, eps = 1e-12)
n4m_fraction_to_percent()
n4m_from_absorbance(is_percent = FALSE)
n4m_kubelka_munk(is_percent = FALSE, epsilon = 1e-10)
n4m_percent_to_fraction()
n4m_to_absorbance(is_percent = FALSE, epsilon = 1e-10, clip_negative = TRUE)
n4m_gaussian(sigma = 1.0, order = 0L, mode = "reflect", cval = 0.0, truncate = 4.0)
n4m_fck_static(kernel_size = NULL, alphas = NULL, sigmas = NULL)
n4m_direct_standardization(fit_intercept = TRUE, ridge = 0.0)
n4m_piecewise_direct_standardization(window_size = 5L, fit_intercept = TRUE, ridge = 0.0)
n4m_robust_direct_standardization(fit_intercept = TRUE, ridge = 0.0, trim_quantile = 0.9, max_iter = 3L
n4m_slope_bias()
n4m_haar()
n4m_wavelet(family = "haar", mode = "periodization")
n4m_wavelet_denoise(family = "db4", mode = "periodization", level = 3L, threshold_mode = "soft", noise_
n4m_wavelet_features(family = "haar", mode = "periodization", max_level = 3L, entropy = "energy")
n4m_wavelet_pca(family = "haar", mode = "periodization", max_level = 2L, n_components = 5.0)
n4m_wavelet_svd(family = "haar", mode = "periodization", max_level = 2L, n_components = 5.0)
n4m_bipls(n_components = 2L, interval_width = 10L, min_intervals = 2L, cv = 3L)
n4m_bve(n_components = 2L, n_steps = 10L, min_features = 0L, cv = 3L)
n4m_cars(n_components = 2L, n_iterations = 50L, min_features = 0L, cv = 3L)
n4m_emcuve(n_components = 2L, noise_features = 50L, noise_seed = NULL, n_ensembles = 10L, vote_threshol
n4m_ga(n_components = 2L, n_generations = 30L, population_size = 40L, min_features = 0L, max_features =
n4m_interval(interval_size = 32L, step = 0L)
n4m_ipw(top_k = NULL, n_components = 2L, n_iterations = 20L, damping = 0.5, weight_floor = 1e-06, cv = 3L
n4m_irf(top_k = NULL, n_components = 2L, n_iterations = 100L, window_size = 5L, initial_intervals = 5L,
n4m_iriv(n_components = 2L, max_rounds = 5L, cv = 5L, seed = NULL)
n4m_pso(n_components = 2L, n_swarm = 30L, n_iterations = 50L, w = 0.729, c1 = 1.494, c2 = 1.494, v_max = 4
n4m_random_frog(top_k = NULL, n_components = 2L, n_iterations = 100L, initial_size = 20L, min_size = 0L,
n4m_randomization(n_components = 2L, n_permutations = 200L, randomization_seed = NULL, alpha = 0.05)
n4m_rep(n_components = 2L, n_steps = 10L, min_features = 0L, remove_count = 1L, cv = 3L)
n4m_scars(n_components = 2L, n_iterations = 50L, min_features = 0L, sample_fraction = 0.8, cv = 3L, seed
n4m_shaving(n_components = 2L, n_steps = 10L, min_features = 0L, shave_fraction = 0.2, cv = 3L)
n4m_sipls(n_components = 2L, interval_width = 10L, combination_size = 2L, cv = 3L)
n4m_spa(top_k = NULL, n_components = 2L)
n4m_st(thresholds = NULL, n_components = 2L, min_selected = 0L, cv = 3L)
n4m_stability(top_k = NULL, n_components = 2L, cv = 3L)
n4m_t2(alpha_thresholds = NULL, n_components = 2L, min_selected = 0L, cv = 3L)
n4m_uve(n_components = 2L, noise_features = 50L, noise_seed = NULL, min_features = -1L, cv = 3L)
n4m_variable_select(top_k = NULL, n_components = 2L, rank_method = "vip")
n4m_vip_spa(top_k = NULL, n_components = 2L, vip_threshold = 0.3)
n4m_vissa(n_components = 2L, n_iterations = 10L, n_submodels = 60L, ratio_kept = 0.1, threshold = 0.5, f
n4m_wvc(top_k = NULL, n_components = 2L, normalize = TRUE)
n4m_wvc_threshold(n_components = 2L, normalize = TRUE, score_threshold = 0.0, threshold_factor = 1.0, m
```

```
n4m_binned_strat_group_kfold(n_splits = 5L, n_bins = 5L, strategy = "uniform", shuffle = TRUE, seed = NU
n4m_kbins_stratified(test_size = 0.25, seed = NULL, n_bins = 5L, strategy = "uniform")
n4m_kennard_stone(test_size = 0.25)
n4m_kmeans(test_size = 0.25, seed = NULL, max_iter = 100L)
n4m_split_splitter(test_size = 0.25, seed = NULL)
n4m_spxy(test_size = 0.25)
n4m_spxy_fold(n_splits = 5L, y_metric = "euclidean")
n4m_spxy_g_fold(n_splits = 5L, y_metric = "euclidean", aggregation = "mean")
n4m_systematic_circular(test_size = 0.25, seed = NULL)
n4m_hotelling_t2(n_components = 5L, alpha = 0.05)
n4m_moments()
n4m_q_residuals(n_components = 5L, alpha = 0.05)
n4m_ragged_summary(offsets = NULL, presence = c(), time_coordinates = c(), missing_policy = "reject")
n4m_signal_type_detector(confidence_threshold = 0.7)
n4m_sweep(cv = 5L, ridge_lambdas = c(0.01, 0.1, 1.0, 10.0), pls_components = c(1L, 2L, 3L, 4L, 5L), heads
n4m_transfer_metrics(n_components = 10L, k_neighbors = 10L, seed = NULL)
```

Arguments

a_high	Native parameter (double); defaults come from the n4m manifest.
a_low	Native parameter (double); defaults come from the n4m manifest.
absolute_threshold	Native parameter (double); defaults come from the n4m manifest.
active_diversity_threshold	Native parameter (double); defaults come from the n4m manifest.
active_max_per_family	Native parameter (int); defaults come from the n4m manifest.
active_score_method	Native parameter (enum); defaults come from the n4m manifest.
active_top_m	Native parameter (int); defaults come from the n4m manifest.
add_high	Native parameter (double); defaults come from the n4m manifest.
add_low	Native parameter (double); defaults come from the n4m manifest.
aggregation	Native parameter (enum); defaults come from the n4m manifest.
alpha	Native parameter (double); defaults come from the n4m manifest.
alpha_thresholds	Native parameter (double_array); defaults come from the n4m manifest.
alphas	Native parameter (double_array); defaults come from the n4m manifest.
amount_hi	Native parameter (double); defaults come from the n4m manifest.
amount_lo	Native parameter (double); defaults come from the n4m manifest.
amplitude_max	Native parameter (double); defaults come from the n4m manifest.
amplitude_min	Native parameter (double); defaults come from the n4m manifest.
asymmetry	Native parameter (double); defaults come from the n4m manifest.
auto_offset	Native parameter (bool); defaults come from the n4m manifest.

aw_high Native parameter (double); defaults come from the n4m manifest.

aw_low Native parameter (double); defaults come from the n4m manifest.

axis_basis Native parameter (enum); defaults come from the n4m manifest.

axis_values Native parameter (double_array); defaults come from the n4m manifest.

b_high Native parameter (double); defaults come from the n4m manifest.

b_low Native parameter (double); defaults come from the n4m manifest.

base Native parameter (double); defaults come from the n4m manifest.

base_intercepts
Native parameter (double_array); defaults come from the n4m manifest.

block_scaling Native parameter (enum); defaults come from the n4m manifest.

bound_water_shift
Native parameter (double); defaults come from the n4m manifest.

bounds_error Native parameter (bool); defaults come from the n4m manifest.

branches Native parameter (int_array); defaults come from the n4m manifest.

bw_hi Native parameter (int); defaults come from the n4m manifest.

bw_lo Native parameter (int); defaults come from the n4m manifest.

c1 Native parameter (double); defaults come from the n4m manifest.

c2 Native parameter (double); defaults come from the n4m manifest.

center Native parameter (bool); defaults come from the n4m manifest.

center_x Native parameter (bool); defaults come from the n4m manifest.

center_y Native parameter (bool); defaults come from the n4m manifest.

chain_offsets Native parameter (int_array); defaults come from the n4m manifest.

chain_params Native parameter (double_array); defaults come from the n4m manifest.

check_inf Native parameter (bool); defaults come from the n4m manifest.

clip_lower Native parameter (double); defaults come from the n4m manifest.

clip_negative Native parameter (bool); defaults come from the n4m manifest.

clip_tolerance Native parameter (double); defaults come from the n4m manifest.

clip_upper Native parameter (double); defaults come from the n4m manifest.

coef0 Native parameter (double); defaults come from the n4m manifest.

coeff_max Native parameter (double_array); defaults come from the n4m manifest.

coeff_min Native parameter (double_array); defaults come from the n4m manifest.

combination_size
Native parameter (int); defaults come from the n4m manifest.

components Native parameter (int_array); defaults come from the n4m manifest.

confidence_threshold
Native parameter (double); defaults come from the n4m manifest.

constant_value Native parameter (double); defaults come from the n4m manifest.

contamination Native parameter (double); defaults come from the n4m manifest.

`correlation` Native parameter (double); defaults come from the n4m manifest.

`crop_max` Native parameter (double); defaults come from the n4m manifest.

`crop_min` Native parameter (double); defaults come from the n4m manifest.

`curvature_strength` Native parameter (double); defaults come from the n4m manifest.

`curvature_type` Native parameter (enum); defaults come from the n4m manifest.

`cv` Native parameter (int); defaults come from the n4m manifest.

`cval` Native parameter (double); defaults come from the n4m manifest.

`damping` Native parameter (double); defaults come from the n4m manifest.

`ddof` Native parameter (int); defaults come from the n4m manifest.

`degree` Native parameter (int); defaults come from the n4m manifest.

`delta` Native parameter (double); defaults come from the n4m manifest.

`deriv` Native parameter (int); defaults come from the n4m manifest.

`derivative_order` Native parameter (int); defaults come from the n4m manifest.

`detector_model` Native parameter (enum); defaults come from the n4m manifest.

`detector_roll_off` Native parameter (bool); defaults come from the n4m manifest.

`di_lambda` Native parameter (double); defaults come from the n4m manifest.

`diff_order` Native parameter (int); defaults come from the n4m manifest.

`dropout_prob` Native parameter (double); defaults come from the n4m manifest.

`edge_curvature` Native parameter (bool); defaults come from the n4m manifest.

`edge_enhancement` Native parameter (double); defaults come from the n4m manifest.

`edge_focus` Native parameter (double); defaults come from the n4m manifest.

`edge_order` Native parameter (int); defaults come from the n4m manifest.

`edge_width` Native parameter (double); defaults come from the n4m manifest.

`edges` Native parameter (double_array); defaults come from the n4m manifest.

`effect_strength` Native parameter (double); defaults come from the n4m manifest.

`enable_broadening` Native parameter (bool); defaults come from the n4m manifest.

`enable_intensity` Native parameter (bool); defaults come from the n4m manifest.

`enable_shift` Native parameter (bool); defaults come from the n4m manifest.

`end` Native parameter (int); defaults come from the n4m manifest.

`entropy` Native parameter (enum); defaults come from the n4m manifest.

`eps` Native parameter (double); defaults come from the n4m manifest.

`epsilon` Native parameter (double); defaults come from the n4m manifest.

`extrapolate` Native parameter (bool); defaults come from the n4m manifest.

`family` Native parameter (enum); defaults come from the n4m manifest.

`fast` Native parameter (bool); defaults come from the n4m manifest.

`feature_max` Native parameter (double); defaults come from the n4m manifest.

`feature_min` Native parameter (double); defaults come from the n4m manifest.

`features_per_subspace` Native parameter (int); defaults come from the n4m manifest.

`fill_value` Native parameter (double); defaults come from the n4m manifest.

`fit_intercept` Native parameter (bool); defaults come from the n4m manifest.

`floor_probability` Native parameter (double); defaults come from the n4m manifest.

`free_water_fraction` Native parameter (double); defaults come from the n4m manifest.

`fusion_lambda` Native parameter (double); defaults come from the n4m manifest.

`fwhm` Native parameter (double); defaults come from the n4m manifest.

`fwhm_high` Native parameter (double); defaults come from the n4m manifest.

`fwhm_low` Native parameter (double); defaults come from the n4m manifest.

`gain_hi` Native parameter (double); defaults come from the n4m manifest.

`gain_lo` Native parameter (double); defaults come from the n4m manifest.

`gain_std` Native parameter (double); defaults come from the n4m manifest.

`gamma` Native parameter (double); defaults come from the n4m manifest.

`gap` Native parameter (int); defaults come from the n4m manifest.

`gap_penalty` Native parameter (double); defaults come from the n4m manifest.

`gating_mode` Native parameter (enum); defaults come from the n4m manifest.

`group_lambda` Native parameter (double); defaults come from the n4m manifest.

`half_window` Native parameter (int); defaults come from the n4m manifest.

`head` Native parameter (enum); defaults come from the n4m manifest.

`heads` Native parameter (enum); defaults come from the n4m manifest.

`huber_k` Native parameter (double); defaults come from the n4m manifest.

`include_baseline_distortion` Native parameter (bool); defaults come from the n4m manifest.

`include_path_length` Native parameter (bool); defaults come from the n4m manifest.

`include_peak_truncation` Native parameter (bool); defaults come from the n4m manifest.

`initial_intervals` Native parameter (int); defaults come from the n4m manifest.

`initial_size` Native parameter (int); defaults come from the n4m manifest.

`interval_size` Native parameter (int); defaults come from the n4m manifest.

`interval_width` Native parameter (int); defaults come from the n4m manifest.

`is_percent` Native parameter (bool); defaults come from the n4m manifest.

`k` Native parameter (double); defaults come from the n4m manifest.

`k_neighbors` Native parameter (int); defaults come from the n4m manifest.

`keep_identity` Native parameter (bool); defaults come from the n4m manifest.

`kernel` Native parameter (enum); defaults come from the n4m manifest.

`kernel_size` Native parameter (int); defaults come from the n4m manifest.

`kernel_width` Native parameter (int); defaults come from the n4m manifest.

`l1_lambda` Native parameter (double); defaults come from the n4m manifest.

`lam` Native parameter (double); defaults come from the n4m manifest.

`lam_0` Native parameter (double); defaults come from the n4m manifest.

`lam_1` Native parameter (double); defaults come from the n4m manifest.

`lam_2` Native parameter (double); defaults come from the n4m manifest.

`learning_rate` Native parameter (double); defaults come from the n4m manifest.

`left_edge` Native parameter (bool); defaults come from the n4m manifest.

`length_scale` Native parameter (double); defaults come from the n4m manifest.

`level` Native parameter (int); defaults come from the n4m manifest.

`lower_percentile` Native parameter (double); defaults come from the n4m manifest.

`max_components` Native parameter (int); defaults come from the n4m manifest.

`max_depth` Native parameter (int); defaults come from the n4m manifest.

`max_features` Native parameter (int); defaults come from the n4m manifest.

`max_half_window` Native parameter (int); defaults come from the n4m manifest.

`max_irls_iter` Native parameter (int); defaults come from the n4m manifest.

`max_iter` Native parameter (int); defaults come from the n4m manifest.

`max_level` Native parameter (int); defaults come from the n4m manifest.

`max_nan_ratio` Native parameter (double); defaults come from the n4m manifest.

`max_rounds` Native parameter (int); defaults come from the n4m manifest.

`max_samples` Native parameter (int); defaults come from the n4m manifest.

`max_shift` Native parameter (double); defaults come from the n4m manifest.

`max_size` Native parameter (int); defaults come from the n4m manifest.

`max_value` Native parameter (double); defaults come from the n4m manifest.

`max_zero_ratio` Native parameter (double); defaults come from the n4m manifest.

`mean_size_um` Native parameter (double); defaults come from the n4m manifest.

`meta_intercept` Native parameter (double_array); defaults come from the n4m manifest.

`meta_weights` Native parameter (double_array); defaults come from the n4m manifest.

`method` Native parameter (enum); defaults come from the n4m manifest.

`min_features` Native parameter (int); defaults come from the n4m manifest.

`min_intervals` Native parameter (int); defaults come from the n4m manifest.

`min_path_length`
Native parameter (double); defaults come from the n4m manifest.

`min_selected` Native parameter (int); defaults come from the n4m manifest.

`min_size` Native parameter (int); defaults come from the n4m manifest.

`min_value` Native parameter (double); defaults come from the n4m manifest.

`min_variance` Native parameter (double); defaults come from the n4m manifest.

`missing_policy` Native parameter (enum); defaults come from the n4m manifest.

`mkl_top_k` Native parameter (int); defaults come from the n4m manifest.

`mode` Native parameter (enum); defaults come from the n4m manifest.

`mode_j` Native parameter (int); defaults come from the n4m manifest.

`mode_k` Native parameter (int); defaults come from the n4m manifest.

`moisture_content`
Native parameter (double); defaults come from the n4m manifest.

`moment_policy` Native parameter (enum); defaults come from the n4m manifest.

`mult_high` Native parameter (double); defaults come from the n4m manifest.

`mult_low` Native parameter (double); defaults come from the n4m manifest.

`mutation_rate` Native parameter (double); defaults come from the n4m manifest.

`n_bands` Native parameter (int); defaults come from the n4m manifest.

`n_bands_hi` Native parameter (int); defaults come from the n4m manifest.

`n_bands_lo` Native parameter (int); defaults come from the n4m manifest.

`n_bins` Native parameter (int); defaults come from the n4m manifest.

`n_components` Native parameter (int); defaults come from the n4m manifest.

`n_components_per_block`
Native parameter (int_array); defaults come from the n4m manifest.

`n_control_points`
Native parameter (int); defaults come from the n4m manifest.

`n_ensembles` Native parameter (int); defaults come from the n4m manifest.

`n_estimators` Native parameter (int); defaults come from the n4m manifest.

`n_generations` Native parameter (int); defaults come from the n4m manifest.

`n_iterations` Native parameter (int); defaults come from the n4m manifest.

`n_joint` Native parameter (int); defaults come from the n4m manifest.

`n_neighbors` Native parameter (int); defaults come from the n4m manifest.

`n_permutations` Native parameter (int); defaults come from the n4m manifest.

`n_predictive` Native parameter (int); defaults come from the n4m manifest.

`n_regions` Native parameter (int); defaults come from the n4m manifest.

`n_spikes_max` Native parameter (int); defaults come from the n4m manifest.

`n_spikes_min` Native parameter (int); defaults come from the n4m manifest.

`n_splits` Native parameter (int); defaults come from the n4m manifest.

`n_steps` Native parameter (int); defaults come from the n4m manifest.

`n_submodels` Native parameter (int); defaults come from the n4m manifest.

`n_swarm` Native parameter (int); defaults come from the n4m manifest.

`n_unique_per_block` Native parameter (int_array); defaults come from the n4m manifest.

`n_x_orthogonal` Native parameter (int); defaults come from the n4m manifest.

`n_y_orthogonal` Native parameter (int); defaults come from the n4m manifest.

`noise_amplification` Native parameter (double); defaults come from the n4m manifest.

`noise_base` Native parameter (double); defaults come from the n4m manifest.

`noise_estimator` Native parameter (enum); defaults come from the n4m manifest.

`noise_features` Native parameter (int); defaults come from the n4m manifest.

`noise_level` Native parameter (double); defaults come from the n4m manifest.

`noise_seed` Native parameter (int); defaults come from the n4m manifest.

`noise_signal_dep` Native parameter (double); defaults come from the n4m manifest.

`noise_std` Native parameter (double); defaults come from the n4m manifest.

`normalize` Native parameter (bool); defaults come from the n4m manifest.

`num_samples` Native parameter (int); defaults come from the n4m manifest.

`offset` Native parameter (double); defaults come from the n4m manifest.

`offset_hi` Native parameter (double); defaults come from the n4m manifest.

`offset_lo` Native parameter (double); defaults come from the n4m manifest.

`offset_max` Native parameter (double); defaults come from the n4m manifest.

`offset_min` Native parameter (double); defaults come from the n4m manifest.

`offset_std` Native parameter (double); defaults come from the n4m manifest.

`offsets` Native parameter (int_array); defaults come from the n4m manifest.

`op_kind` Native parameter (enum); defaults come from the n4m manifest.

`op_kinds` Native parameter (int_array); defaults come from the n4m manifest.

`op_params` Native parameter (double_array); defaults come from the n4m manifest.

`operator_range_max` Native parameter (double); defaults come from the n4m manifest.

`operator_range_min` Native parameter (double); defaults come from the n4m manifest.

`order` Native parameter (int); defaults come from the n4m manifest.

`outlier_threshold` Native parameter (double); defaults come from the n4m manifest.

`overall_strength`
: Native parameter (double); defaults come from the n4m manifest.

`p`
: Native parameter (double); defaults come from the n4m manifest.

`p_range`
: Native parameter (double); defaults come from the n4m manifest.

`pad_mode`
: Native parameter (enum); defaults come from the n4m manifest.

`param`
: Native parameter (double); defaults come from the n4m manifest.

`param_offsets`
: Native parameter (int_array); defaults come from the n4m manifest.

`path_length_sensitivity`
: Native parameter (double); defaults come from the n4m manifest.

`path_length_std`
: Native parameter (double); defaults come from the n4m manifest.

`peak_probability`
: Native parameter (double); defaults come from the n4m manifest.

`perturbation_density`
: Native parameter (double); defaults come from the n4m manifest.

`perturbation_intensity`
: Native parameter (double); defaults come from the n4m manifest.

`perturbation_range_max`
: Native parameter (double); defaults come from the n4m manifest.

`perturbation_range_min`
: Native parameter (double); defaults come from the n4m manifest.

`pls_components`
: Native parameter (int_array); defaults come from the n4m manifest.

`polynomial_order`
: Native parameter (int); defaults come from the n4m manifest.

`polynomial_strength`
: Native parameter (double); defaults come from the n4m manifest.

`polyorder`
: Native parameter (int); defaults come from the n4m manifest.

`population_size`
: Native parameter (int); defaults come from the n4m manifest.

`presence`
: Native parameter (int_array); defaults come from the n4m manifest.

`probability`
: Native parameter (double); defaults come from the n4m manifest.

`profile`
: Native parameter (enum); defaults come from the n4m manifest.

`randomization_seed`
: Native parameter (int); defaults come from the n4m manifest.

`rank`
: Native parameter (int); defaults come from the n4m manifest.

`rank_method`
: Native parameter (enum); defaults come from the n4m manifest.

`ratio_kept`
: Native parameter (double); defaults come from the n4m manifest.

`reference`
: Native parameter (double_array); defaults come from the n4m manifest.

`reference_mode`
: Native parameter (enum); defaults come from the n4m manifest.

`reference_size_um`
: Native parameter (double); defaults come from the n4m manifest.

`reference_valid_mask`
: Native parameter (int_array); defaults come from the n4m manifest.

`reference_water_activity`
: Native parameter (double); defaults come from the n4m manifest.

`region_specific`
: Native parameter (bool); defaults come from the n4m manifest.

`regularizer` Native parameter (double); defaults come from the n4m manifest.

`remove_count` Native parameter (int); defaults come from the n4m manifest.

`ridge` Native parameter (double); defaults come from the n4m manifest.

`ridge_lambda` Native parameter (double); defaults come from the n4m manifest.

`ridge_lambdas` Native parameter (double_array); defaults come from the n4m manifest.

`right_edge` Native parameter (bool); defaults come from the n4m manifest.

`sample_fraction`
: Native parameter (double); defaults come from the n4m manifest.

`scale` Native parameter (bool); defaults come from the n4m manifest.

`scale_x` Native parameter (bool); defaults come from the n4m manifest.

`scale_y` Native parameter (bool); defaults come from the n4m manifest.

`score_threshold`
: Native parameter (double); defaults come from the n4m manifest.

`score_weight` Native parameter (double); defaults come from the n4m manifest.

`seed` Native parameter (int); defaults come from the n4m manifest.

`segment` Native parameter (int); defaults come from the n4m manifest.

`shave_fraction` Native parameter (double); defaults come from the n4m manifest.

`shift_hi` Native parameter (double); defaults come from the n4m manifest.

`shift_lo` Native parameter (double); defaults come from the n4m manifest.

`shuffle` Native parameter (bool); defaults come from the n4m manifest.

`sigma` Native parameter (double); defaults come from the n4m manifest.

`sigma_gain` Native parameter (double); defaults come from the n4m manifest.

`sigma_hi` Native parameter (double); defaults come from the n4m manifest.

`sigma_lo` Native parameter (double); defaults come from the n4m manifest.

`sigmas` Native parameter (double_array); defaults come from the n4m manifest.

`size_effect_strength`
: Native parameter (double); defaults come from the n4m manifest.

`size_range_high_um`
: Native parameter (double); defaults come from the n4m manifest.

`size_range_low_um`
: Native parameter (double); defaults come from the n4m manifest.

`size_variation_um`
: Native parameter (double); defaults come from the n4m manifest.

`slope_max` Native parameter (double); defaults come from the n4m manifest.

slope_min Native parameter (double); defaults come from the n4m manifest.

slope_std Native parameter (double); defaults come from the n4m manifest.

smooth_half_window
Native parameter (int); defaults come from the n4m manifest.

solver Native parameter (enum); defaults come from the n4m manifest.

sparsity_lambda
Native parameter (double); defaults come from the n4m manifest.

spline_degree Native parameter (int); defaults come from the n4m manifest.

spline_points Native parameter (int); defaults come from the n4m manifest.

start Native parameter (int); defaults come from the n4m manifest.

std_penalty Native parameter (double); defaults come from the n4m manifest.

step Native parameter (int); defaults come from the n4m manifest.

strategy Native parameter (enum); defaults come from the n4m manifest.

stray_light Native parameter (bool); defaults come from the n4m manifest.

stray_light_fraction
Native parameter (double); defaults come from the n4m manifest.

stretch_hi Native parameter (double); defaults come from the n4m manifest.

stretch_lo Native parameter (double); defaults come from the n4m manifest.

target_wavelengths
Native parameter (double_array); defaults come from the n4m manifest.

tau Native parameter (double); defaults come from the n4m manifest.

temp_high Native parameter (double); defaults come from the n4m manifest.

temp_low Native parameter (double); defaults come from the n4m manifest.

temperature_delta
Native parameter (double); defaults come from the n4m manifest.

test_size Native parameter (double); defaults come from the n4m manifest.

tgt_min Native parameter (double); defaults come from the n4m manifest.

tgt_n Native parameter (int); defaults come from the n4m manifest.

tgt_step Native parameter (double); defaults come from the n4m manifest.

threshold Native parameter (double); defaults come from the n4m manifest.

threshold_factor
Native parameter (double); defaults come from the n4m manifest.

threshold_mode Native parameter (enum); defaults come from the n4m manifest.

threshold_multiplier
Native parameter (double); defaults come from the n4m manifest.

thresholds Native parameter (double_array); defaults come from the n4m manifest.

time_coordinates
Native parameter (double_array); defaults come from the n4m manifest.

tol Native parameter (double); defaults come from the n4m manifest.

top_k Native parameter (int); defaults come from the n4m manifest.

`trim_quantile` Native parameter (double); defaults come from the n4m manifest.

`truncate` Native parameter (double); defaults come from the n4m manifest.

`truncated_peaks` Native parameter (bool); defaults come from the n4m manifest.

`uniform` Native parameter (bool); defaults come from the n4m manifest.

`upper_percentile` Native parameter (double); defaults come from the n4m manifest.

`use_aw_range` Native parameter (bool); defaults come from the n4m manifest.

`use_crop` Native parameter (bool); defaults come from the n4m manifest.

`use_fwhm_range` Native parameter (bool); defaults come from the n4m manifest.

`use_size_range` Native parameter (bool); defaults come from the n4m manifest.

`use_temp_range` Native parameter (bool); defaults come from the n4m manifest.

`v_max` Native parameter (double); defaults come from the n4m manifest.

`valid_mask` Native parameter (int_array); defaults come from the n4m manifest.

`variation_scope` Native parameter (enum); defaults come from the n4m manifest.

`vip_threshold` Native parameter (double); defaults come from the n4m manifest.

`vote_threshold` Native parameter (double); defaults come from the n4m manifest.

`w` Native parameter (double); defaults come from the n4m manifest.

`water_activity_delta` Native parameter (double); defaults come from the n4m manifest.

`wavelength_exponent` Native parameter (double); defaults come from the n4m manifest.

`weight_floor` Native parameter (double); defaults come from the n4m manifest.

`weights` Native parameter (double_array); defaults come from the n4m manifest.

`width_hi` Native parameter (int); defaults come from the n4m manifest.

`width_high` Native parameter (int); defaults come from the n4m manifest.

`width_lo` Native parameter (int); defaults come from the n4m manifest.

`width_low` Native parameter (int); defaults come from the n4m manifest.

`width_max` Native parameter (double); defaults come from the n4m manifest.

`width_min` Native parameter (double); defaults come from the n4m manifest.

`window` Native parameter (int); defaults come from the n4m manifest.

`window_half_width` Native parameter (int); defaults come from the n4m manifest.

`window_length` Native parameter (int); defaults come from the n4m manifest.

`window_size` Native parameter (int); defaults come from the n4m manifest.

`with_center` Native parameter (bool); defaults come from the n4m manifest.

`with_mean` Native parameter (bool); defaults come from the n4m manifest.

`with_scale` Native parameter (bool); defaults come from the n4m manifest.

`with_std` Native parameter (bool); defaults come from the n4m manifest.

`y_factor` Native parameter (double); defaults come from the n4m manifest.

`y_metric` Native parameter (enum); defaults come from the n4m manifest.

Value

An unfitted estimator whose classes are its role interfaces; fit it with `n4m_estimator_fit`.

`n4m_estimator_roles` *Generic native estimator roles*

Description

Constructors such as `n4m_pls_regression()` or `n4m_cppls()` return an unfitted estimator whose classes are its role interfaces: `n4m_regressor` (`predict`), `n4m_classifier` (`predict`, `n4m_classes`), `n4m_transformer` (`n4m_estimator_transform`) and `n4m_selector` (`n4m_estimator_transform`, which returns the selected columns in input order, and `n4m_selected_indices`, 1-based in native selection order) and `n4m_sample_filter` (`n4m_sample_mask`, a logical keep mask; filters are train-only, filters on the target read `y`). The fitted state is kept as portable N4ME bytes, so `saveRDS()`/`readRDS()` and the Python and JS/WASM bindings reuse it without refitting.

Usage

```
n4m_estimator_fit(object, X, y = NULL, ...)

## S3 method for class 'n4m_estimator'
n4m_estimator_fit(object, X, y = NULL, sample_weight = NULL,
  groups = NULL, feature_groups = NULL, blocks = NULL, axis = NULL,
  X_target = NULL, fold_ids = NULL, ...)

## S3 method for class 'n4m_regressor'
predict(object, newdata, ...)

## S3 method for class 'n4m_classifier'
predict(object, newdata, type = c("class", "prob", "decision"), ...)

n4m_classes(object)

## S3 method for class 'n4m_classifier'
n4m_classes(object)

n4m_sample_mask(object, X, y = NULL)

## S3 method for class 'n4m_sample_filter'
n4m_sample_mask(object, X, y = NULL)

n4m_estimator_transform(object, X, ...)

## S3 method for class 'n4m_transformer'
n4m_estimator_transform(object, X, ...)

## S3 method for class 'n4m_selector'
```

```
n4m_estimator_transform(object, X, ...)

n4m_selected_indices(object)

## S3 method for class 'n4m_selector'
n4m_selected_indices(object)

n4m_estimator_export(object, allow_training_rows = FALSE)

n4m_contains_training_rows(object)

n4m_estimator_import(bytes)

## S3 method for class 'n4m_method'
print(x, ...)
```

Arguments

`object`	An estimator from one of the generated constructors (`n4m_estimator_role_constructors`).
`X`	Numeric matrix (rows are samples).
`y`	Numeric response vector (one value per row of `X`) or matrix (one row per row of `X`) for regressors; class labels (factor, character or numeric vector, one per row) for classifiers: integer labels are the class ids (within int64), strings and other finite numbers become a label table; missing (NA) and non-finite labels are refused. Nothing is recycled: another length is an error.
`sample_weight, groups, feature_groups, blocks, axis, X_target, fold_ids`	Optional fit inputs; each method declares which ones it requires and the native core refuses the others. Per-row inputs have one entry per row of `X`, `feature_groups` and `axis` one per column.
`newdata`	Numeric matrix of new samples.
`type`	For classifiers: `"class"` (labels, a factor when the estimator was fitted on a factor or character vector), `"prob"` (class probabilities, for methods that define them) or `"decision"` (method-defined class scores). Matrix columns are named by class.
`allow_training_rows`	`n4m_estimator_export()` refuses a state that embeds training rows (`n4m_contains_training_rows()`, e.g. kernel PLS, GPR-PLS, LW-PLS) unless this is `TRUE`: sharing the export shares those rows. `saveRDS()` keeps them, as a checkpoint of the in-process object.
`bytes`	Raw vector produced by `n4m_estimator_export()`.
`x`	An estimator.
`...`	Unused.

Value

`n4m_estimator_fit()` returns the fitted estimator (a failed fit leaves the object passed in unchanged); `predict()` and `n4m_estimator_transform()` return numeric results (class labels for

predict(type = "class")); n4m_classes() the fitted classes; n4m_sample_mask() a logical vector (TRUE keeps the row); n4m_estimator_export() a raw vector; n4m_contains_training_rows() a logical; n4m_estimator_import() a fitted estimator.

---

n4m_fit *Fit a PLS regression model via the libn4m C ABI.*

Description

Accepts a numeric matrix X (n x p) and a numeric vector or matrix Y (n x q). Both are coerced to double precision and row-major contiguous before being passed across the C boundary.

Usage

```
n4m_fit(
  X,
  Y,
  algo,
  n_components,
  store_scores = FALSE,
  center_x = TRUE,
  scale_x = TRUE,
  center_y = TRUE,
  scale_y = TRUE,
  embedded_snv_savgol = NULL
)
```

Arguments

X	Numeric matrix, n x p.
Y	Numeric matrix or vector.
algo	Character. Solver name (see Details).
n_components	Integer >= 1.
store_scores	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
center_x	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
scale_x	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
center_y	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
scale_y	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
embedded_snv_savgol	Optional two-number vector 'c(window_length, polyorder)'. The native N4MM model then embeds default SNV followed by Savitzky-Golay smoothing, and predicts directly from raw spectra.

Details

'algo' selects the solver. Recognized values: "pls_nipals", "pls_orthogonal_scores", "pls_simpls", "pls_kernel_algorithm", "pls_wide_kernel", "pls_svd", "pls_power", "pls_randomized_svd", "pcr_svd", "opls_nipals".

Value

An external pointer wrapping the fitted model handle. Pass it to [n4m_predict()] to obtain predictions. The model is freed automatically when the external pointer is garbage-collected.

`n4m_manifest` *Native manifest and method lookup*

Description

`n4m_manifest_json()` returns the native manifest (every catalog method's roles, DAG-ML node kinds, fit inputs and typed parameters with defaults) as a JSON string, the same document every n4m binding reads. `n4m_constructor()` returns the generated constructor of a catalog method id, so pipelines can be built from method ids.

Usage

```
n4m_manifest_json()

n4m_constructor(method_id)
```

Arguments

`method_id`	Catalog method id, for example `"models.pls.cppls"`.

Value

A JSON string, or a constructor function.

`n4m_method` *Low-level n4m method dispatcher.*

Description

Low-level n4m method dispatcher.

Usage

```
n4m_method(
  algo,
  X,
  Y,
  n_components,
  params = list(),
  center_x = TRUE,
  scale_x = FALSE,
  center_y = TRUE,
  scale_y = FALSE
)
```

Arguments

`algo`	Character; algorithm name (see Details).
`X`	Numeric matrix or NULL (for one_se_rule_compute).
`Y`	Numeric matrix or vector. Pass an n x 1 placeholder for classifier-style fits where labels go in 'params$y_labels'.
`n_components`	Positive integer.
`params`	Named list of algorithm-specific parameters (sparsity_lambda, sample_weights, block_sizes, X_target, y_labels, alpha_thresholds, ...).
`center_x`, `scale_x`, `center_y`, `scale_y`	Boolean preprocessing flags (default centering, no scaling — matches the Python tier-1 defaults).

Details

Supported algorithm names:

Fit/producer branches (37; not all support out-of-sample prediction): "sparse_simpls" "cppls" "ecr" "di_pls" "weighted_pls" "robust_pls" "ridge_pls" "continuum_regression" "recursive_pls" "n_pls" "kernel_pls" "o2pls" "sparse_pls_da" "group_sparse_pls" "fused_sparse_pls" "so_pls" "on_pls" "rosa" "bagging_pls" "boosting_pls" "random_subspace_pls" "gpr_pls" "pls_glm" "pls_qda" "pls_cox" "pds" "ds" "mir_pls" "missing_aware_nipals" "mb_pls" "lw_pls" "pls_lda" "pls_logistic" "ridge" "aom_preprocess" "aom_pls" "pop_pls"

Selectors (25): "spa_select" "cars_select" "interval_select" "stability_select" "uve_select" "random_frog_select" "scars_select" "ga_select" "pso_select" "vissa_select" "shaving_select" "bve_select" "t2_select" "wvc_select" "wvc_threshold_select" "emcuve_select" "randomization_select" "bipls_select" "sipls_select" "rep_select" "ipw_select" "st_select" "iriv_select" "irf_select" "vip_spa_select"

Diagnostics (2 via dispatcher): "approximate_press_compute" "one_se_rule_compute"

Value

A named list with the MethodResult arrays + scalars. Index fields ('selected_indices', 'top_k_intervals', ...) are 1-based.

n4m_model_descriptor *Inspect a fully validated N4MM model descriptor*

Description

Validates all N4MM sections before reporting the numerical recipe and capabilities. Enum values are the versioned C ABI values from `n4m.h`.

Usage

```
n4m_model_descriptor(bytes)
```

Arguments

`bytes`	Non-empty raw N4MM vector.

Value

A list of wire version, algorithm, solver, deflation, feature/target dimensions, component count, and capability mask.

n4m_model_export *Export a fitted n4m model as portable N4MM bytes*

Description

Exports the native fitted model to the portable libn4m format. Unlike an R external pointer, the bytes can be imported by another libn4m binding.

Usage

```
n4m_model_export(model)
```

Arguments

`model`	External pointer returned by `n4m_fit()` or `n4m_model_import()`.

Value

A raw vector containing an N4MM model.

See Also

`n4m_model_import`

n4m_model_import *Import a fitted model from portable N4MM bytes*

Description

Imports a fitted model through the native libn4m parser. Treat bytes from external sources as untrusted input.

Usage

```
n4m_model_import(bytes)
```

Arguments

bytes	Non-empty raw N4MM vector.

Value

An external pointer with the same lifecycle as `n4m_fit()`.

See Also

`n4m_model_export`

n4m_model_import_linear_predictor *Import an affine predictor as a portable N4MM model*

Description

Calls the n4m C ABI affine importer without fitting coefficients. Predictions are `X %*% coefficients + intercept`. The resulting model is exportable to portable N4MM but has no PLS latent decomposition.

Usage

```
n4m_model_import_linear_predictor(coefficients, intercept,
                                  source_training_samples = 0L)
```

Arguments

coefficients	Finite numeric features-by-targets matrix.
intercept	Finite numeric vector, one value per target.
source_training_samples	Non-negative training row count, or zero when the source format does not attest it.

Value

A native model pointer accepted by `n4m_predict()` and `n4m_model_export()`.

---

n4m_model_inspect *Inspect portable N4MM model metadata*

Description

Checks the N4MM header through the native libn4m parser before full import.

Usage

```
n4m_model_inspect(bytes)
```

Arguments

`bytes`	Non-empty raw N4MM vector.

Value

A list with `format_version` and `writer_abi`.

See Also

`n4m_model_import`

---

n4m_model_pipeline_info *Inspect the native preprocessing state embedded in N4MM bytes*

Description

Uses the authoritative libn4m decoder and rejects malformed payloads. Format-1 models return `present = FALSE`. For the supported format-2 profile, the fingerprint is a hexadecimal FNV-1a-64 identity.

Usage

```
n4m_model_pipeline_info(bytes)
```

Arguments

`bytes`	Non-empty raw N4MM vector.

Value

A list with `present`, `semantic_profile`, Savitzky-Golay parameters, raw/model feature widths, and `fingerprint`.

n4m_multimodal_classifier
Native Raw Multimodal PLS-logistic Classifier

Description

The portable C ABI learns raw StandardScaler, TensorPCA and mixed metadata encoders and a genuine PLS-logistic classifier. Bindings marshal raw arrays and typed label tables; they contain no learned numerical logic.

Usage

```
n4m_multimodal_classifier(recipe, source_schemas)
n4m_multimodal_classifier_from_state(state, recipe, source_schemas,
                                     class_names = NULL)
```

Arguments

recipe	Closed version-one early-fusion recipe using the existing native PLS-logistic head, with positive integer n_components and max_iter.
source_schemas	Named raw source declarations in recipe source order.
state	Bounded raw N4MC bytes containing encoders and classifier state.
class_names	Optional unique string or exact integer original labels in native class-column order. Without this table, native integer IDs are exposed.

Details

Use `object <- n4m_fit(object, X, y)`. Failed fitting leaves the supplied object usable. `predict(object, X, type = "prob")` returns columns in `n4m_classes(object)` order; `type = "class"` decodes original labels. N4MC bytes contain no training rows. Carry class_names beside the portable state to restore original labels. Recipe, schema and native class consistency are validated before hydration. ABI 2.17 is required.

Value

An n4m_multimodal_classifier object with a native owner.

`n4m_multimodal_pipeline` *Complete Native Raw Multimodal Pipeline*

Description

Native population scaling, learned PCA and training-only UTF-8 one-hot encoding precede weighted early fusion and one Ridge predictor. Unknown categories encode as zero. No host encoders or training rows are stored in the portable state. Import validates the expected recipe and full source identity without fitting.

Usage

```
n4m_multimodal_pipeline(recipe, source_schemas)
n4m_multimodal_pipeline_from_state(state, recipe, source_schemas)
n4m_export_state(object)
n4m_close(object)
## S3 method for class 'n4m_multimodal_pipeline'
n4m_estimator_fit(object, X, y = NULL, ...)
## S3 method for class 'n4m_multimodal_pipeline'
predict(object, newdata,
  source_schemas = object$source_schemas, ...)
## S3 method for class 'n4m_multimodal_pipeline'
n4m_estimator_transform(object, X,
  source_schemas = object$source_schemas, ...)
```

Arguments

`recipe`	Closed version-one early-fusion recipe: an ordered subset of one to four distinct nir, image, series and metadata sources, their encoder and weight declarations, and one centered, unscaled-X Ridge.
`source_schemas`	Named raw source descriptors containing representation_id, input_shape, dtype and the unchanged canonical IO identity string.
`state`	Raw vector containing a bounded native N4MF fitted state.
`object`	A native multimodal pipeline.
`X`, `newdata`	Named list of IO-aligned sample-first raw numeric arrays and an n-by-two character matrix (or declared numeric/string data frame) for metadata.
`y`	One numeric target per sample.
`...`	Unsupported additional options are refused.

Details

Requires ABI 2.16. Numeric arrays use R's double storage. Raw descriptors and array layouts are translated only; all learned arithmetic belongs to libn4m. Encoder, weight and schema mappings cover exactly the selected source names. Absent sources are not fitted; selected zero-weight sources

retain their historical encoder fitting semantics. N4MF format 1 and canonical four-source archives remain supported. The existing `n4m_fit(object, blocks, y)` entry point delegates this class to `n4m_estimator_fit`. Calls with extra constructor controls are refused.

Value

Constructors and fit return a pipeline; predict returns a numeric vector, transform returns the actual weighted encoded matrix, export returns raw N4MF bytes, and close releases native resources invisibly and idempotently.

`n4m_optimizer` *Native hyperparameter optimization from R*

Description

Build a typed search space and run the libn4m optimizer through its public C ABI. Sampling, pruning, ordered trial events, and N4MOPT checkpoints are owned by the native engine. The R binding marshals values and owns the native handle.

Usage

```
n4m_param_int(low, high, step = 1, log = FALSE)
n4m_param_float(low, high, step = 0, log = FALSE)
n4m_param_categorical(choices, type = NULL)
n4m_param_ordinal(choices)
n4m_param_sorted_tuple(length, low, high, integer = FALSE)
n4m_constraint(kind, refs, labels = rep("", length(refs)))
n4m_optimizer(space, constraints = list(), sampler = "random", pruner = "none",
  direction = "auto", eval_mode = "mean", metric = "rmse", liar = "none",
  n_startup_trials = 10L, seed = 0, timeout_seconds = 0,
  max_resource = 0L, reduction_factor = 0L)
n4m_optimizer_close(optimizer)
n4m_optimizer_ask(optimizer)
n4m_optimizer_ask_batch(optimizer, n)
n4m_optimizer_enqueue(optimizer, parameters)
n4m_optimizer_tell(optimizer, trial, score = NULL, status = "completed", error = NULL)
n4m_optimizer_intermediate(optimizer, trial, step, score)
n4m_optimizer_best(optimizer)
n4m_optimizer_trials(optimizer, since_id = 0)
n4m_optimizer_save(optimizer)
n4m_optimizer_load(checkpoint)
```

Arguments

`low, high`	Finite inclusive axis bounds. Integer axes use binary64-exact integers.
`step`	Grid step; zero means continuous for floating axes.
`log`	Whether native sampling uses a logarithmic axis.

choices	Nonempty homogeneous categorical vector, or ordered finite numeric values.
type	Optional native category type: string, integer, float, or boolean.
length	Number of elements in a sorted tuple.
integer	Whether tuple elements are integers.
kind	Constraint kind: mutex_group (also mutex), requires, exclude, condition_in, or condition_not_in.
refs, labels	Ordered parameter references and matching labels for a constraint.
space	Nonempty named list of axis declarations. Declaration order is significant.
constraints	List of constraint declarations.
sampler	One of random, sobol, lhs, ternary, ga, pso, cmaes, tpe, gp_ei.
pruner	One of none, median, asha, hyperband, racing.
direction	auto, minimize, or maximize.
eval_mode	Native evaluation mode. Only mean is currently implemented.
metric	Native metric name: rmse, mse, mae, r2, accuracy, balanced_accuracy, f1, or logloss.
liar	Native constant-liar policy. Only none is currently implemented.
n_startup_trials	Startup trial count for adaptive sampling and median pruning.
seed	Nonnegative binary64-exact native seed.
timeout_seconds	Finite nonnegative wall-clock timeout, zero for none.
max_resource	Positive resource limit for Hyperband, zero otherwise.
reduction_factor	ASHA/Hyperband factor, zero for native default.
optimizer	An n4m_optimizer handle.
n	Requested batch size, zero through 10000.
parameters	Named numeric warm-start values; categorical values are zero-based indices.
trial	A returned trial record or its exact numeric ID.
score	Finite score for a completed trial; NULL otherwise.
status	Terminal trial status: completed, pruned, failed, or cancelled.
error	Optional plain or versioned native error text for failed or cancelled trials.
since_id	Inclusive nonnegative trial-ID filter.
checkpoint	List returned by n4m_optimizer_save().

Details

The native checkpoint contains the authoritative ordered search space and optimizer state. Keep the accompanying R axis declarations unchanged: they are used only to decode trial values because the C trial accessor does not enumerate its original declarations. Host evaluation remains the caller's responsibility. A binding does not make a Python-only estimator runnable in R. For the trace key/shape contract, see docs/methods/optimization.md in the source repository. Unsupported native options fail at creation.

Value

Axis and constraint constructors return plain R declarations. `n4m_optimizer()` and `n4m_optimizer_load()` return native handles. `n4m_optimizer_close()` releases one immediately; garbage collection also releases it. `n4m_optimizer_ask()` returns a list containing an exact numeric `id`, typed `parameters`, native `status`, and `rung`. Inactive conditional axes have `NULL` values. Batch asks return a list of committed trials; if a non-benign native error occurs after commits, the list has a `native_status` attribute and the caller must terminalize or recover those trials. `n4m_optimizer_best()` returns the best completed trial and score. `n4m_optimizer_trials()` returns the complete owning native trace v1 as a named list; int64 vectors become character when a value exceeds binary64's exact integer range. `n4m_optimizer_save()` returns a portable raw N4MOPT blob with the R axis declarations needed to decode trials after `n4m_optimizer_load()`.

Examples

```
space <- list(components = n4m_param_int(1, 4))
opt <- n4m_optimizer(space, seed = 42)
trial <- n4m_optimizer_ask(opt)
n4m_optimizer_tell(opt, trial, (trial$parameters$components - 2)^2)
n4m_optimizer_best(opt)$score
restored <- n4m_optimizer_load(n4m_optimizer_save(opt))
n4m_optimizer_trials(restored)$n_trials
```

---

n4m_predict *Predict with a fitted n4m model.*

Description

Predict with a fitted n4m model.

Usage

```
n4m_predict(model, X)
```

Arguments

`model`	External pointer returned by [n4m_fit()].
`X`	Numeric matrix, n_new x p.

Value

Numeric matrix, n_new x n_targets.

`n4m_preprocess_fit` *Generic native preprocessing pipeline*

Description

A thin R interface to the libn4m C ABI pipeline. The C ABI names 19 operator kinds; 15 currently implement generic pipeline fit and transform. The other four are reported as unsupported by `n4m_preprocess_capabilities()` and rejected by `n4m_preprocess_step()`.

Usage

```
n4m_preprocess_capabilities()

n4m_preprocess_step(kind, params = numeric())

n4m_preprocess_fit(X, steps, Y = NULL)

n4m_preprocess_transform(object, X)

n4m_preprocess_export(object)

n4m_preprocess_import(bytes)

n4m_preprocess_plan(object)
```

Arguments

`kind`	Operator name from `n4m_preprocess_capabilities()`.
`params`	Finite numeric vector of positional C ABI parameters for the operator.
`X`	Finite numeric matrix. At transform time, its feature width must match training.
`steps`	Nonempty list of `n4m_preprocess_step()` specifications.
`Y`	Optional finite numeric response matrix, required for supervised OSC and EPO.
`object`	Fitted native pipeline returned by `n4m_preprocess_fit()`.
`bytes`	A raw vector containing a fitted native N4MP pipeline.

Details

All numerical fitting and transformation is performed by libn4m. The fitted state is an in-process native pointer; save the exported N4MP raw bytes, not this pointer, for use in another R session or supported language binding. The generic pipeline EMSC uses a reference, intercept, and polynomial basis on a normalized wavelength axis. It is not numerically interchangeable with the historical standalone `emsc_transform()` convention, which has no intercept and uses integer wavelength powers.

Parameter vectors are positional; use `numeric()` for the native default. The contract is defined by `n4m_operator_kind_t` and the native pipeline parameter parsers in `cpp/src/core/pipeline.cpp`:

No parameters identity, center, autoscale, pareto_scale, snv, msc.

Optional one integer emsc polynomial degree (default 2, range 0–5); detrend_poly degree (default 1, range 0–5).

Savitzky–Golay savgol_smooth takes window, polynomial degree (defaults 11, 3); savgol_derivative takes window, polynomial degree, derivative order, and optional delta (defaults 11, 3, 1, 1). Window must be odd; delta is the positive feature spacing.

Other unsupervised norris_williams takes odd segment, odd gap, derivative order (defaults 5, 3, 1); asls_baseline takes lambda, asymmetry, iterations (defaults 100000, 0.001, 10); wavelet_denoise takes levels, nonnegative threshold (defaults 1, 0).

Supervised osc and epo take max_iter, positive tolerance (defaults 100, 1e-10); each requires `Y` at fit time.

The native fit rejects invalid arity or parameter ranges; R does not duplicate the numerical contract.

Value

`n4m_preprocess_capabilities()` returns a data frame of names, numeric codes, and support flags. `n4m_preprocess_step()` returns an operator specification. `n4m_preprocess_fit()` returns fitted native state. `n4m_preprocess_transform()` returns a numeric matrix. `n4m_preprocess_export()` returns portable raw bytes; `n4m_preprocess_import()` restores a fitted native pipeline. `n4m_preprocess_plan()` returns its ordered original operator plan.

`n4m_procedures` *Generic native procedures*

Description

Constructors such as `n4m_kennard_stone()`, `n4m_gaussian_noise()` or `n4m_regression_metrics()` return a procedure: a catalog method run once, without fitted state. Its class is its role: `n4m_splitter` (`n4m_split`), `n4m_augmenter` (`n4m_augment`, train-only) or `n4m_procedure` (`n4m_run`). Parameters, seeds included, are native and shared with the Python and JS/WASM bindings.

Usage

```
n4m_split(object, X, y = NULL, groups = NULL)

## S3 method for class 'n4m_splitter'
n4m_split(object, X, y = NULL, groups = NULL)

n4m_augment(object, X, y = NULL, axis = NULL)

## S3 method for class 'n4m_augmenter'
n4m_augment(object, X, y = NULL, axis = NULL)

n4m_run(object, X, y = NULL, ...)

## S3 method for class 'n4m_procedure'
n4m_run(object, X, y = NULL, ...)
```

Arguments

object	A procedure from one of the generated constructors (n4m_estimator_role_constructors).
X	Numeric matrix (rows are samples).
y	Optional target vector or matrix, when the method uses it.
groups	Optional sample groups for group-aware splitters.
axis	Optional spectral axis for axis-dependent augmenters (wavelengths in nm for the augmenters with nanometre constants).
...	Further named inputs (X_target, fold_ids, ...).

Value

n4m_split() a list of folds, each list(train, test) of 1-based row indices; n4m_augment() the augmented matrix, or list(X, Y) for the augmenters that mix rows (mixup; y required, mixed with the same draw and returned with its shape); n4m_run() a named list of the native outputs.

---

n4m_role_pipeline *Native role pipelines*

Description

n4m_role_pipeline() builds a trained linear recipe of catalog estimators: sample filters (training rows only), transformers and selectors, then one regressor or classifier. The recipe is validated natively (role order, final step, parameters). n4m_estimator_fit() fits every step natively: y holds responses for a final regressor (a vector or a matrix, one row per sample) or class labels for a final classifier; steps that require y receive every response column (or the class ids), sample filters drop training rows of every row-aligned input, and inputs no step uses are refused. Column names of X are stored and checked at every later call (a reordered, renamed or missing column is refused); matrices without column names are positional. The fitted state is one N4ME payload per stateful step: n4m_role_pipeline_export() refuses states that embed training rows unless allow_training_rows = TRUE, and n4m_role_pipeline_import() refuses states that contradict the recipe (method, parameters, widths). The same states are read by the Python, JS/WASM and Rust bindings.

Usage

```
n4m_role_pipeline(steps)

## S3 method for class 'n4m_role_pipeline'
n4m_estimator_fit(object, X, y = NULL, sample_weight = NULL,
  groups = NULL, feature_groups = NULL, blocks = NULL, axis = NULL,
  X_target = NULL, fold_ids = NULL, ...)

## S3 method for class 'n4m_role_pipeline'
predict(object, newdata, type = c("class", "prob", "decision"), ...)
```

```
## S3 method for class 'n4m_role_pipeline'
n4m_estimator_transform(object, X, ...)

n4m_role_pipeline_export(object, allow_training_rows = FALSE)

n4m_role_pipeline_import(steps, states, feature_names = NULL, class_names = NULL)

n4m_role_pipeline_steps(object)

n4m_role_pipeline_info(object)

## S3 method for class 'n4m_role_pipeline'
n4m_classes(object)

## S3 method for class 'n4m_role_pipeline'
print(x, ...)
```

Arguments

steps	A list of steps: method ids ("models.pls.cppls" or "n4m:models.pls.cppls"), list(method_id =, params = list()) or recipe tokens list(class = "n4m:<id>", params = list()).
object	A role pipeline.
X, newdata	Numeric matrix or data frame (rows are samples).
y	Responses (regression) or class labels (classification).
sample_weight, groups, feature_groups, blocks, axis, X_target, fold_ids	Optional fit inputs, routed natively to the steps that declare them.
type	For a final classifier: "class", "prob" or "decision".
allow_training_rows	Export states that embed training rows.
states	List of N4ME raw vectors, one per stateful step (or the list returned by n4m_role_pipeline_export()).
feature_names	Optional input column names of the imported pipeline.
class_names	Optional label table of an imported classifier (index = class id): a non-empty vector, or list of scalars, of unique strings or finite numbers with an entry for every class id of the fitted state.
x	A role pipeline to print.
...	Unused.

Value

n4m_role_pipeline() an unfitted pipeline; n4m_estimator_fit() and n4m_role_pipeline_import() a fitted one; predict() and n4m_estimator_transform() numeric results (class labels for type = "class"); n4m_role_pipeline_export() a list of list(method_id, n4me, contains_training_rows); n4m_role_pipeline_steps() a data frame describing each step. n4m_role_pipeline_info() a list with the input width n_features, the stored feature_names (or NULL), the classes of a classifier pipeline (labels when fitted on labels, integer ids otherwise; NULL for a regressor), the

label_names table of a classifier fitted on labels (index = class id + 1; unlike classes it keeps labels whose rows a sample filter removed; NULL otherwise) and the steps data frame. n4m_classes() the classes of a classifier pipeline.

---

n4m_splitter_run *Run one of nine native sample splitters*

Description

Runs the selected existing libn4m splitter via one C ABI dispatch. R only marshals data and indices; partitioning remains native.

Usage

```
n4m_splitter_run(kind, X = NULL, Y = NULL, groups = NULL,
  fold = 1L, test_size = 0.25, n_splits = 3L, y_metric = 1L,
  aggregation = 0L, n_bins = 5L, strategy = 0L, shuffle = TRUE,
  max_iter = 100L, seed = 0, zero_based = FALSE)
```

Arguments

kind	One of kennard_stone, spxy, spxy_fold, spxy_group_fold, kmeans, kbins_stratified, binned_strat_group_fold, systematic_circular, or data_twinning.
X, Y	Finite numeric matrices required by the selected kind.
groups	Exact integer group ID per sample for group-fold kinds.
fold	One-based fold number for fold kinds.
test_size	Fraction assigned to the test set for non-fold kinds.
n_splits	Number of folds for fold kinds.
y_metric	SPXY-fold Y metric: 0 none, 1 Euclidean, 2 Hamming.
aggregation	Grouped SPXY aggregation: 0 mean, 1 median.
n_bins	Number of target bins for binned splitters.
strategy	Bin strategy: 0 uniform, 1 quantile.
shuffle	Whether binned group folds shuffle.
max_iter	Maximum KMeans iterations.
seed	Exact nonnegative random seed.
zero_based	Return zero-based indices when TRUE.

Value

A list with native-ordered integer vectors train and test.

`n4m_version` *Runtime version string of the loaded libn4m.*

Description

Runtime version string of the loaded libn4m.

Usage

```
n4m_version()
```

Value

A character scalar like "1.0.3+abi.2.0.0".

`o2pls` *O2-PLS — formula entry point (uses n_predictive for component count).*

Description

O2-PLS — formula entry point (uses n_predictive for component count).

Usage

```
o2pls(
  formula,
  data,
  n_predictive = 2L,
  n_x_orthogonal = 1L,
  n_y_orthogonal = 1L,
  na.action = stats::na.omit
)
```

Arguments

`formula`	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
`data`	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
`n_predictive`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
`n_x_orthogonal`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
`n_y_orthogonal`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
`na.action`	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A 'n4m_method_fit' object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use 'predict()' for inference and 'coef()' to extract regression coefficients.

`o2pls_fit` *O2-PLS (bi-directional OPLS).*

Description

O2-PLS (bi-directional OPLS).

Usage

```
o2pls_fit(X, Y, n_predictive = 2L, n_x_orthogonal = 1L, n_y_orthogonal = 1L)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_predictive`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
`n_x_orthogonal`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
`n_y_orthogonal`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

`on_pls_fit` *OnPLS — Orthogonal multi-block PLS (joint + unique loadings).*

Description

OnPLS — Orthogonal multi-block PLS (joint + unique loadings).

Usage

```
on_pls_fit(X, Y, n_joint, n_unique_per_block, block_sizes)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_joint	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
n_unique_per_block	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
block_sizes	Integer vector. Per-block feature counts for multi-block PLS.

---

one_se_rule *One-SE rule from a (max_components × n_folds) fold RMSE matrix.*

Description

One-SE rule from a (max_components × n_folds) fold RMSE matrix.

Usage

```
one_se_rule(fold_rmse_matrix)
```

Arguments

fold_rmse_matrix	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

opls *Formula-based OPLS regression wrapper around the n4m C ABI.*

Description

Uses the same formula/S3 return contract as [pls()], with the model API configured as 'algo = "opls_nipals"'.

Usage

```
opls(formula, data, ncomp = 2L, na.action = stats::na.omit)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

An object of class ‘c("opls_fit", "n4m_fit", "pls_fit")‘.

---

pds_fit *Piecewise Direct Standardization (calibration transfer).*

Description

Piecewise Direct Standardization (calibration transfer).

Usage

```
pds_fit(X_source, X_target, window_half_width = 2L)
```

Arguments

X_source	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
X_target	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
window_half_width	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.

---

pls *Formula-based PLS regression wrapper around the n4m C ABI*

Description

Tier-2 idiomatic R interface for tier-1 [n4m_fit()] / [n4m_predict()]. Provides a formula entry point and S3 generics so the returned object plays well with ‘predict()‘, ‘summary()‘, ‘coef()‘, ‘print()‘, base R ‘cv.glmnet‘-style workflows, and any caret model that supports a ‘predict(object, newdata)‘ adapter.

When the first argument is not a formula, ‘pls(x, y, ...)‘ dispatches to the matrix-oriented [pls_mdatools()] compatibility facade.

Usage

```
pls(
  formula,
  data,
  ncomp = 2L,
  algo = "pls_nipals",
  na.action = stats::na.omit,
  ...
)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
algo	Character. One of ‘"pls_nipals"‘, ‘"pls_simpls"‘, ‘"pls_svd"‘, ‘"pls_power"‘, ‘"pls_kernel_algorithm"‘, ‘"pls_wide_kernel"‘, ‘"pls_orthogonal_scores"‘, ‘"pls_randomized_svd"‘, ‘"pcr_svd"‘, ‘"opls_nipals"‘. Defaults to ‘"pls_nipals"‘ to match the R ‘pls‘ package’s default.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.
...	For matrix-style calls, forwarded to [pls_mdatools()].

Value

An object of class ‘c("n4m_fit", "pls_fit")‘ with components: * ‘model‘ — external pointer to the libn4m model handle * ‘formula‘ — the call’s formula * ‘terms‘ — the ‘terms()‘ object describing the model * ‘xlevels‘ — factor levels of the predictors, for newdata coercion * ‘call‘ — the original call (for ‘print‘ / ‘summary‘) * ‘ncomp‘ — components used * ‘algo‘ — solver used * ‘n_features_in‘ — predictor count * ‘response_name‘ — left-hand side of the formula (string)

Examples

```
## Not run:
  set.seed(0)
  df <- data.frame(
    x1 = rnorm(100), x2 = rnorm(100), x3 = rnorm(100)
  )
  df$y <- 2 * df$x1 - df$x2 + rnorm(100, sd = 0.1)
  fit <- pls(y ~ ., data = df, ncomp = 3)
  summary(fit)
  preds <- predict(fit, newdata = df)

## End(Not run)
```

pls_cox_fit *PLS-Cox proportional hazards.*

Description

PLS-Cox proportional hazards.

Usage

```
pls_cox_fit(X, n_components, survival_times, event_indicators)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
n_components	Integer. Number of latent components.
survival_times	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
event_indicators	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

pls_diagnostics *PLS diagnostics: T², Q, DModX from a fitted model.*

Description

PLS diagnostics: T², Q, DModX from a fitted model.

Usage

```
pls_diagnostics(model, X)
```

Arguments

model	External pointer from 'n4m_fit()'.
X	Numeric matrix to score (typically the training matrix).

Value

A list with 't2', 'q', 'dmodx' — each is a 1-row matrix of length nrow(X).

pls_glm *PLS-GLM — formula entry point. Default is Gaussian; set ‘family = "poisson"‘ for Poisson IRLS.*

Description

PLS-GLM — formula entry point. Default is Gaussian; set ‘family = "poisson"‘ for Poisson IRLS.

Usage

```
pls_glm(
  formula,
  data,
  ncomp = 2L,
  family = "gaussian",
  na.action = stats::na.omit
)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
family	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

pls_glm_fit *PLS-GLM — Gaussian (default) or Poisson IRLS.*

Description

PLS-GLM — Gaussian (default) or Poisson IRLS.

Usage

```
pls_glm_fit(X, Y, n_components, poisson = FALSE)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
poisson	Logical; TRUE selects the Poisson-link path.

Value

A list with ‘coefficients‘, ‘intercept‘, ‘predictions‘, ‘x_mean‘, ‘rmse‘.

---

pls_lda_fit *PLS-LDA — Linear Discriminant Analysis on PLS scores.*

Description

PLS-LDA — Linear Discriminant Analysis on PLS scores.

Usage

```
pls_lda_fit(X, y_labels, n_components, n_classes = NULL)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
y_labels	Integer vector. Class labels.
n_components	Integer. Number of latent components.
n_classes	Integer >= 2. Number of class labels.

---

pls_logistic_fit *Multinomial logistic regression on PLS scores.*

Description

Multinomial logistic regression on PLS scores.

Usage

```
pls_logistic_fit(X, y_labels, n_components, n_classes = NULL)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
y_labels	Integer vector. Class labels.
n_components	Integer. Number of latent components.
n_classes	Integer >= 2. Number of class labels.

`pls_mdatools` *mdatools-style matrix PLS interface*

Description

Matrix-oriented PLS facade modelled after `mdatools::pls(x, y, ...)` for NIRS/chemometrics workflows. The top-level `pls(x, y, ...)` function dispatches here when its first argument is not a formula.

Usage

```
pls_mdatools(
  x,
  y,
  ncomp = min(nrow(x) - 1L, ncol(x), 20L),
  center = TRUE,
  scale = FALSE,
  cv = NULL,
  exclcols = NULL,
  exclrows = NULL,
  x.test = NULL,
  y.test = NULL,
  method = c("simpls", "cppls", "kernelpls",
             "widekernelpls", "oscorespls", "nipals", "pcr"),
  info = "",
  ncomp.selcrit = "min",
  lim.type = "ddmoments",
  alpha = 0.05,
  gamma = 0.5,
  cv.scope = "local",
  prep = NULL,
  fit_components = TRUE,
  ...
)
```

Arguments

`x`	Numeric predictor matrix.
`y`	Numeric response vector or matrix.
`ncomp`	Maximum number of components.
`center`	Logical; center predictors and response.
`scale`	Logical; standardize predictors and response.
`cv`	`NULL`, an integer number of folds, or a list of assessment row indices.
`exclcols`	Optional predictor columns to exclude.
`exclrows`	Optional rows to exclude before fitting.

x.test	Optional external test predictor matrix.
y.test	Optional external test response vector or matrix.
method	PLS/PCR algorithm selector.
info	Free-form model label.
ncomp.selcrit	Stored for compatibility.
lim.type	Stored for compatibility.
alpha	Stored for compatibility.
gamma	CPPLS gamma when method = "cppls".
cv.scope	Stored for compatibility.
prep	Optional preprocessing function applied to x and x.test.
fit_components	Logical; fit component prefixes 1:ncomp.
...	Reserved for compatibility.

Value

A n4m_mdatools_pls object with calibration results in calres; optional cross-validation and test results are stored in cvres and testres.

Examples

```
## Not run:
set.seed(1)
X <- matrix(rnorm(80), 20, 4)
y <- X[, 1] - X[, 2] + rnorm(20, sd = 0.05)
fit <- pls_mdatools(X, y, ncomp = 3, cv = 5)
predict(fit, x = X)
fit$calres$rmse
fit$cvres$rmse

## End(Not run)
```

---

pls_monitoring *PLS process monitoring (Hotelling T^2 + Q with alarms).*

Description

Computes phase-1 thresholds on ‘X_reference‘, then evaluates ‘X_monitor‘ against those thresholds and reports per-row alarms.

Usage

```
pls_monitoring(model, X_reference, X_monitor, alpha = 0.95)
```

Arguments

model	External pointer from ‘n4m_fit()‘.
X_reference	Phase-1 numeric matrix (used to set thresholds).
X_monitor	Phase-2 numeric matrix (rows are evaluated).
alpha	Confidence level (e.g. 0.95). Thresholds correspond to the (1 - alpha) quantile.

Value

A list with ‘t2‘, ‘q‘, ‘t2_alarms‘, ‘q_alarms‘, ‘any_alarms‘, ‘t2_threshold‘, ‘q_threshold‘.

---

pls_qda_fit *PLS-QDA (Quadratic Discriminant Analysis on PLS scores).*

Description

PLS-QDA (Quadratic Discriminant Analysis on PLS scores).

Usage

```
pls_qda_fit(X, y_labels, n_components)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
y_labels	Integer vector. Class labels.
n_components	Integer. Number of latent components.

---

plsr *PLS package compatibility facade*

Description

These functions provide the familiar formula-oriented surface of the R **pls** package without importing it. Computation is backed by **n4m**; the facade is intended for NIRS/chemometrics scripts that already use `plsr()`, `pcr()`, `predict()`, `RMSEP()` and `selectNcomp()` patterns.

Usage

```
plsr(
  formula,
  ncomp = 2L,
  data,
  subset,
  na.action = stats::na.omit,
  method = c("kernelpls", "widekernelpls", "simpls",
             "oscorespls", "cppls", "nipals"),
  scale = FALSE,
  validation = c("none", "CV"),
  segments = 10L,
  center = TRUE,
  fit_components = TRUE,
  gamma = 0.5,
  sparsity_lambda = 0,
  ...
)

pcr(
  formula,
  ncomp = 2L,
  data,
  subset,
  na.action = stats::na.omit,
  method = c("svdpc", "pcr"),
  scale = FALSE,
  validation = c("none", "CV"),
  segments = 10L,
  center = TRUE,
  fit_components = TRUE,
  ...
)

mvr(
  formula,
  ncomp = 2L,
  data,
  subset,
  na.action = stats::na.omit,
  method = c("kernelpls", "widekernelpls", "simpls",
             "oscorespls", "cppls", "nipals", "pcr"),
  scale = FALSE,
  validation = c("none", "CV"),
  segments = 10L,
  center = TRUE,
  model = c("plsr", "pcr"),
  fit_components = TRUE,
```

```
  gamma = 0.5,
  sparsity_lambda = 0,
  ...
)

MSEP(object, ...)

RMSEP(object, ...)

R2(object, ...)

selectNcomp(object, method = c("min"), estimate = c("CV", "train"), ...)
```

Arguments

formula	A two-sided model formula.
ncomp	Number of latent components.
data	Data frame containing formula variables.
subset	Optional row subset.
na.action	Missing-value action.
method	PLS/PCR algorithm selector.
scale	Logical; standardize predictors and response.
validation	"none" or "CV".
segments	Number of contiguous CV segments, or a list of assessment row indices.
center	Logical; center predictors and response.
fit_components	Logical; fit component prefixes 1:ncomp for component-wise prediction and metrics.
gamma	CPPLS gamma.
sparsity_lambda	Sparse-SIMPLS soft threshold for the dense SIMPLS-compatible path.
model	"plsr" or "pcr".
object	A n4m_mvr object.
estimate	Metric source, "train" or "CV".
...	Reserved for compatibility.

Value

plsr(), pcr() and mvr() return a n4m_mvr object. MSEP(), RMSEP() and R2() return a small mvrVal-compatible list with a val array. selectNcomp() returns an integer component count.

Examples

```
## Not run:
set.seed(1)
df <- data.frame(x1 = rnorm(40), x2 = rnorm(40))
df$y <- 2 * df$x1 - df$x2 + rnorm(40, sd = 0.05)
fit <- plsr(y ~ ., data = df, ncomp = 2, method = "simpls")
predict(fit, df, ncomp = 1:2)
RMSEP(fit)
selectNcomp(fit)

## End(Not run)
```

`predict.n4m_fit` *Predict from a [pls()]-fitted model.*

Description

Predict from a [pls()]-fitted model.

Usage

```
## S3 method for class 'n4m_fit'
predict(object, newdata = NULL, ...)
```

Arguments

`object`	A ‘n4m_fit‘ returned by [pls()].
`newdata`	A ‘data.frame‘ (or matrix). If ‘NULL‘, predictions on the training matrix are returned.
`...`	Ignored (for S3 generic compatibility).

Value

Numeric vector (for single-target regression) or matrix (multi-target).

`predict.n4m_method_fit`
Predict from a MethodResult-based n4m fit.

Description

Predict from a MethodResult-based n4m fit.

Usage

```
## S3 method for class 'n4m_method_fit'
predict(object, newdata = NULL, ...)
```

Arguments

object	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
newdata	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
...	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

pso_select *PSO-PLS (Binary Particle Swarm Optimization).*

Description

PSO-PLS (Binary Particle Swarm Optimization).

Usage

```
pso_select(
  X,
  Y,
  n_components,
  n_swarm = 30L,
  n_iterations = 50L,
  w = 0.729,
  c1 = 1.494,
  c2 = 1.494,
  v_max = 4,
  seed = 0L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_swarm	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
n_iterations	Integer >= 1. Number of outer-loop iterations.
w	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

c1	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
c2	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
v_max	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
seed	Integer. Random seed for reproducibility.

`random_frog_select` *Random Frog (Phase 5g).*

Description

Random Frog (Phase 5g).

Usage

```
random_frog_select(
  X,
  Y,
  n_components,
  n_iterations = 100L,
  initial_size = 30L,
  min_size = NULL,
  max_size = NULL,
  top_k = 10L,
  seed = 0L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_iterations	Integer >= 1. Number of outer-loop iterations.
initial_size	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
min_size	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
max_size	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
top_k	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
seed	Integer. Random seed for reproducibility.

`random_subspace_pls` *Random-subspace PLS — formula entry point.*

Description

Random-subspace PLS — formula entry point.

Usage

```
random_subspace_pls(
  formula,
  data,
  ncomp = 2L,
  n_estimators = 50L,
  features_per_subspace = 10L,
  seed = 0L,
  na.action = stats::na.omit
)
```

Arguments

`formula`	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
`data`	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
`ncomp`	Integer. Number of latent components.
`n_estimators`	Integer >= 1. Number of bootstrap / boosting / random-subspace estimators.
`features_per_subspace`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
`seed`	Integer. Random seed for reproducibility.
`na.action`	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

random_subspace_pls_fit *Random-subspace PLS (Ho 1998).*

Description

Random-subspace PLS (Ho 1998).

Usage

```
random_subspace_pls_fit(
  X,
  Y,
  n_components,
  n_estimators = 50L,
  features_per_subspace = 10L,
  seed = 0L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_estimators	Integer >= 1. Number of bootstrap / boosting / random-subspace estimators.
features_per_subspace	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
seed	Integer. Random seed for reproducibility.

randomization_select *Randomization test selector.*

Description

Randomization test selector.

Usage

```
randomization_select(
  X,
  Y,
  n_components,
  n_permutations = 100L,
  randomization_seed = 0L,
  alpha = 0.05
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_permutations	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
randomization_seed	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
alpha	Numeric in [0, 1]. Elastic-net / penalty mixing parameter.

---

recursive_pls *Recursive PLS – formula entry point.*

Description

Recursive PLS stores in-sample predictions rather than a reusable coefficient model. 'predict()' therefore accepts only the fitted training design.

Usage

```
recursive_pls(
  formula,
  data,
  ncomp = 2L,
  window_size = 50L,
  na.action = stats::na.omit
)
```

Arguments

formula	A two-sided formula (e.g. 'y ~ .' or 'y ~ x1 + x2 + x3'). Response on the left, predictors on the right.
data	A 'data.frame' (or anything 'as.data.frame'-coercible) containing the response and predictor columns referenced by 'formula'.
ncomp	Integer. Number of latent components.
window_size	Integer moving-window size.
na.action	What to do with 'NA's. Default: 'na.omit'.

recursive_pls_fit *Moving-window recursive PLS.*

Description

Moving-window recursive PLS.

Usage

```
recursive_pls_fit(X, Y, n_components, window_size = 50L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
window_size	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

rep_select *REP-PLS.*

Description

REP-PLS.

Usage

```
rep_select(
  X,
  Y,
  n_components,
  n_steps = 10L,
  min_features = 5L,
  remove_count = 1L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_steps	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

min_features	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
remove_count	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

`ridge_fit` *Direct (closed-form) multi-output Ridge regression.*

Description

Solves `beta = (Xc'Xc + lambda I)^-1 Xc'Yc` on column-centered X/Y (distinct from the ridge-augmented SIMPLS of `ridge_pls_fit`). `ridge_lambda` is the sklearn `alpha` (the L2 penalty).

Usage

```
ridge_fit(X, Y, n_components = 1L, ridge_lambda = 1)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Unused by the closed-form solver; accepted for a uniform method signature.
ridge_lambda	Numeric >= 0. L2 penalty (sklearn `alpha`).

`ridge_pls` *Ridge PLS — formula entry point.*

Description

Ridge PLS — formula entry point.

Usage

```
ridge_pls(
  formula,
  data,
  ncomp = 2L,
  ridge_lambda = 1,
  na.action = stats::na.omit
)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
ridge_lambda	Numeric >= 0. Ridge regularisation strength.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

---

ridge_pls_fit *L2-augmented PLS regression.*

Description

L2-augmented PLS regression.

Usage

```
ridge_pls_fit(X, Y, n_components, ridge_lambda = 1)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
ridge_lambda	Numeric >= 0. Ridge regularisation strength.

`robust_pls` *Robust PLS — formula entry point.*

Description

Robust PLS — formula entry point.

Usage

```
robust_pls(
  formula,
  data,
  ncomp = 2L,
  huber_k = 1.345,
  max_irls_iter = 20L,
  na.action = stats::na.omit
)
```

Arguments

`formula`	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
`data`	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
`ncomp`	Integer. Number of latent components.
`huber_k`	Numeric >= 0. Huber loss tuning constant.
`max_irls_iter`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
`na.action`	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

`robust_pls_fit` *Robust PLS via Huber IRLS.*

Description

Robust PLS via Huber IRLS.

Usage

```
robust_pls_fit(X, Y, n_components, huber_k = 1.345, max_irls_iter = 20L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
huber_k	Numeric >= 0. Huber loss tuning constant.
max_irls_iter	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

robust_snv_transform *Robust Standard Normal Variate transform*

Description

Apply libn4m's median and robust-dispersion row normalization.

Usage

```
robust_snv_transform(X, with_center = TRUE, with_scale = TRUE, k = 1.4826)
```

Arguments

X	Numeric matrix.
with_center	Center each row by its median.
with_scale	Scale each row by robust dispersion.
k	Robust scale factor.

Value

Numeric matrix with the same shape as X.

rosa_fit *ROSA — Response-Oriented Sequential Alternation.*

Description

ROSA — Response-Oriented Sequential Alternation.

Usage

```
rosa_fit(X, Y, n_components, block_sizes)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
block_sizes	Integer vector. Per-block feature counts for multi-block PLS.

scars_select *SCARS — Stability + CARS.*

Description

SCARS — Stability + CARS.

Usage

```
scars_select(
  X,
  Y,
  n_components,
  n_iterations = 50L,
  min_features = 5L,
  sample_fraction = 0.8,
  seed = 0L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_iterations	Integer >= 1. Number of outer-loop iterations.

min_features	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
sample_fraction	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
seed	Integer. Random seed for reproducibility.

`selectivity_ratio_select` *Selectivity-ratio ranker.*

Description

Selectivity-ratio ranker.

Usage

```
selectivity_ratio_select(model, X, top_k)
```

Arguments

model	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
X	Numeric matrix used for the fit (re-passed for diagnostics).
top_k	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

Value

A list with 'scores' and 'selected_indices'.

`shaving_select` *Shaving selector.*

Description

Shaving selector.

Usage

```
shaving_select(
  X,
  Y,
  n_components,
  n_steps = 10L,
  min_features = 5L,
  shave_fraction = 0.1
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_steps	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
min_features	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
shave_fraction	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

sipls_select *siPLS — synergistic interval PLS.*

Description

siPLS — synergistic interval PLS.

Usage

```
sipls_select(X, Y, n_components, interval_width = 10L, combination_size = 2L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
interval_width	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
combination_size	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

snv_transform *Portable preprocessing transforms*

Description

Thin R wrappers over libn4m's preprocessing C ABI. The R layer performs matrix layout conversion only; all numerical work is delegated to libn4m.

Usage

```
snv_transform(X, with_mean = TRUE, with_std = TRUE, ddof = 0L)

savgol_transform(
  X,
  window_length,
  polyorder = 3L,
  deriv = 0L,
  delta = 1,
  mode = "mirror",
  cval = 0
)
```

Arguments

X	Numeric matrix.
with_mean	Logical; center each row before scaling.
with_std	Logical; scale each row by its standard deviation.
ddof	Integer degrees-of-freedom correction.
window_length	Odd integer Savitzky-Golay window length.
polyorder	Savitzky-Golay polynomial order.
deriv	Savitzky-Golay derivative order.
delta	Sample spacing.
mode	Boundary mode: "mirror", "constant", "nearest", "wrap", or "interp".
cval	Constant fill value used when mode = "constant".

Value

Numeric matrix with the same dimensions as X.

`so_pls_fit` *Sequential & Orthogonalised multi-block PLS (Næs et al. 2011). 'block_sizes' integer vector summing to ncol(X); 'n_components_per_block' integer vector of same length.*

Description

Sequential & Orthogonalised multi-block PLS (Næs et al. 2011). 'block_sizes' integer vector summing to ncol(X); 'n_components_per_block' integer vector of same length.

Usage

```
so_pls_fit(X, Y, block_sizes, n_components_per_block)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`block_sizes`	Integer vector. Per-block feature counts for multi-block PLS.
`n_components_per_block`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

`spa_select` *SPA — Successive Projections Algorithm.*

Description

Fit-data selector: pass (X, Y) and the desired number of features.

Usage

```
spa_select(X, Y, n_components, top_k)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_components`	Integer. Number of latent components.
`top_k`	Number of features to select.

Value

A list with 'selected_indices' (1-based) and 'best_rmse'.

sparse_pls *Sparse SIMPLS — formula entry point.*

Description

Sparse SIMPLS — formula entry point.

Usage

```
sparse_pls(
  formula,
  data,
  ncomp = 2L,
  sparsity_lambda = 0.05,
  na.action = stats::na.omit
)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
sparsity_lambda	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

Value

A ‘n4m_method_fit‘ object carrying the fitted model, in-sample predictions, training RMSE, and method-specific metadata. Use ‘predict()‘ for inference and ‘coef()‘ to extract regression coefficients.

sparse_pls_da_fit *Sparse PLS-DA classifier (‘y_labels‘ is an integer vector of class IDs).*

Description

Sparse PLS-DA classifier (‘y_labels‘ is an integer vector of class IDs).

Usage

```
sparse_pls_da_fit(X, y_labels, n_components, sparsity_lambda = 0.05)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
y_labels	Integer vector. Class labels.
n_components	Integer. Number of latent components.
sparsity_lambda	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

---

sparse_simpls_fit *Sparse SIMPLS fit.*

Description

Sparse SIMPLS fit.

Usage

```
sparse_simpls_fit(X, Y, n_components, sparsity_lambda = 0.05)
```

Arguments

X	Numeric matrix.
Y	Numeric matrix or vector.
n_components	Integer >= 1.
sparsity_lambda	Soft-threshold magnitude per component (>= 0).

Value

A list with 'coefficients', 'predictions', 'x_mean', 'y_mean', 'weights_w', 'rmse'.

---

spectral_encoding *Native fitted spectral representations*

Description

Fit LVSE local subspaces or GCU global nonnegative factors using only training spectra. All numerical operations use the native C ABI.

Usage

```
lvse_fit(X, width = 64L, rank = 4L, overlap = 0,
         standardize = TRUE, snv = FALSE)
gcu_fit(X, rank = 16L, max_iter = 60L, tol = 0.001)
spectral_transform(encoder, X)
spectral_export_affine(encoder)
```

Arguments

X	Numeric matrix, samples by channels.
width	Local window width.
rank	Maximum local LVSE rank or global GCU rank.
overlap	Window overlap fraction, in [0,1).
standardize	Use training channel standard deviations.
snv	Apply per-spectrum SNV before LVSE.
max_iter	Coordinate descent iteration limit.
tol	Relative projected-gradient tolerance.
encoder	Fitted native spectral encoder.

Details

LVSE caps each block rank at min(training rows, block width)-1 and skips singleton blocks. GCU uses training minima, a global standard deviation, exact NNDSVDa initialization and cyclic coordinate descent. All supplied rows are used. Fit inside each training fold to prevent validation leakage. Affine export is supported only for LVSE without SNV. External pointers must be refitted after restarting R.

Value

Fit functions return an external pointer. Transform returns a matrix. Affine export returns a list with operator (outputs by channels) and offset (one row).

---

st_select *ST-PLS — score-threshold selector.*

Description

ST-PLS — score-threshold selector.

Usage

```
st_select(X, Y, n_components, thresholds, min_selected = NULL)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
thresholds	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
min_selected	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

stability_select *Stability selector (coefficient stability, MCUVE-style).*

Description

Stability selector (coefficient stability, MCUVE-style).

Usage

```
stability_select(X, Y, n_components, top_k = 10L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
top_k	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

t2_select *T2-PLS - sweep over alpha thresholds.*

Description

T2-PLS - sweep over alpha thresholds.

Usage

```
t2_select(X, Y, n_components, alpha_thresholds, min_selected = NULL)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
alpha_thresholds	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
min_selected	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

uve_select — *UVE — Uninformative Variable Elimination.*

Description

UVE — Uninformative Variable Elimination.

Usage

```
uve_select(X, Y, n_components, noise_features = NULL, noise_seed = 0L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
noise_features	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
noise_seed	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

vip_select — *VIP (Variable Importance in Projection) ranker.*

Description

Operates on an already-fitted n4m model handle (from 'n4m_fit'). Returns the top-'top_k' features by VIP score.

Usage

```
vip_select(model, X, top_k)
```

Arguments

model	External pointer from 'n4m_fit()'.
X	Numeric matrix used for the fit (re-passed for diagnostics).
top_k	Integer; number of features to return.

Value

A list with 'scores' (length p VIP scores) and 'selected_indices' (1-based, length top_k).

vip_spa_select *VIP-SPA hybrid selector.*

Description

VIP-SPA hybrid selector.

Usage

```
vip_spa_select(X, Y, n_components, vip_threshold = 0.3, top_k = 10L)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
vip_threshold	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
top_k	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

vissa_select *VISSA — Variable Iterative Space Shrinkage Approach.*

Description

VISSA — Variable Iterative Space Shrinkage Approach.

Usage

```
vissa_select(
  X,
  Y,
  n_components,
  n_iterations = 20L,
  n_submodels = 100L,
  ratio_kept = 0.1,
  threshold = 0.5,
  floor_probability = 0.01,
  seed = 0L
)
```

Arguments

X	Numeric matrix of predictors (rows = samples, cols = features).
Y	Numeric matrix or vector of responses, with one row per sample.
n_components	Integer. Number of latent components.
n_iterations	Integer >= 1. Number of outer-loop iterations.
n_submodels	Integer >= 1. Number of inner sub-models per VISSA-style iteration.
ratio_kept	Numeric in (0, 1]. Fraction of features kept per iteration.
threshold	Numeric. Convergence / pruning threshold.
floor_probability	Numeric in [0, 0.5). Lower bound on per-feature retention probability.
seed	Integer. Random seed for reproducibility.

---

weighted_pls *Sample-weighted PLS — formula entry point.*

Description

Sample-weighted PLS — formula entry point.

Usage

```
weighted_pls(formula, data, ncomp = 2L, weights, na.action = stats::na.omit)
```

Arguments

formula	A two-sided formula (e.g. ‘y ~ .‘ or ‘y ~ x1 + x2 + x3‘). Response on the left, predictors on the right.
data	A ‘data.frame‘ (or anything ‘as.data.frame‘-coercible) containing the response and predictor columns referenced by ‘formula‘.
ncomp	Integer. Number of latent components.
weights	Numeric vector of length nrow(data) with sample weights.
na.action	What to do with ‘NA‘s. Default: ‘na.omit‘.

`weighted_pls_fit` *Sample-weighted PLS (sqrt(w)-prescaled SIMPLS).*

Description

Sample-weighted PLS (sqrt(w)-prescaled SIMPLS).

Usage

```
weighted_pls_fit(X, Y, n_components, sample_weights)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_components`	Integer. Number of latent components.
`sample_weights`	Numeric vector of length nrow(X), strictly positive finite weights.

Value

A list with ‘coefficients‘, ‘predictions‘, ‘x_mean‘, ‘y_mean‘, ‘rmse‘.

`wvc_select` *WVC-PLS — weighted vector correlation top-k selector.*

Description

WVC-PLS — weighted vector correlation top-k selector.

Usage

```
wvc_select(X, Y, n_components, top_k = 10L, normalize = TRUE)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_components`	Integer. Number of latent components.
`top_k`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.
`normalize`	Method-specific parameter. See the underlying ‘*_fit()‘ function for the exact semantics.

`wvc_threshold_select` *WVC-threshold selector.*

Description

WVC-threshold selector.

Usage

```
wvc_threshold_select(
  X,
  Y,
  n_components,
  normalize = TRUE,
  threshold = 0,
  threshold_factor = 1,
  min_selected = 1L
)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, cols = features).
`Y`	Numeric matrix or vector of responses, with one row per sample.
`n_components`	Integer. Number of latent components.
`normalize`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
`threshold`	Numeric. Convergence / pruning threshold.
`threshold_factor`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.
`min_selected`	Method-specific parameter. See the underlying '*_fit()' function for the exact semantics.

Index

∗ **models**
    aom_pls, 5

aom_pls, 5
aom_preprocess, 6
aompls *(*aom_pls*)*, 5
approximate_press, 6
area_normalization_transform, 7

bagging_pls, 8
bagging_pls_fit, 8
bipls_select, 9
boosting_pls, 9
boosting_pls_fit, 10
bve_select, 11

cars_select, 11
coef.n4m_fit, 12
coef.n4m_mvr *(*plsr*)*, 85
coefficient_select, 12
continuum_regression, 13
continuum_regression_fit, 13
cppls, 14
cppls_fit, 15

detrend_transform, 15
di_pls, 16
di_pls_fit, 16
ds_fit, 17

ecr, 17
ecr_fit, 18
emcuve_select, 19
emsc_fit, 19
emsc_transform, 20

fused_sparse_pls_fit, 20

ga_select, 21
gcu_fit *(*spectral_encoding*)*, 105
gpr_pls_fit, 22
group_sparse_pls_fit, 22

interval_select, 23
ipw_select, 23
irf_select, 24
iriv_select, 25

kennard_stone_split, 26
kernel_pls_fit, 26

local_snv_transform, 27
lvse_fit *(*spectral_encoding*)*, 105
lw_pls_fit, 28

mb_pls, 28
mb_pls_fit, 29
mir_pls, 29
mir_pls_fit, 30
missing_aware_nipals, 30
missing_aware_nipals_fit, 31
msc_fit, 31
msc_transform, 32
MSEP *(*plsr*)*, 85
mvr *(*plsr*)*, 85

n4m_abi_version, 33
n4m_affine_fit, 33
n4m_affine_model_export, 34
n4m_affine_supported_methods, 34
n4m_airpls
        *(*n4m_estimator_role_constructors*)*, 35
n4m_aom_chain_fixed_fit
        *(*n4m_estimator_role_constructors*)*, 35
n4m_aom_chain_ridge_pls
        *(*n4m_estimator_role_constructors*)*, 35
n4m_aom_chain_sweep
        *(*n4m_estimator_role_constructors*)*, 35

n4m_aom_pls
(n4m_estimator_role_constructors), 35
n4m_aom_pls_superblock
(n4m_estimator_role_constructors), 35
n4m_aom_preprocessing
(n4m_estimator_role_constructors), 35
n4m_aom_ridge_pls_superblock
(n4m_estimator_role_constructors), 35
n4m_aom_sweep
(n4m_estimator_role_constructors), 35
n4m_approximate_press
(n4m_estimator_role_constructors), 35
n4m_area_normalization
(n4m_estimator_role_constructors), 35
n4m_arpls
(n4m_estimator_role_constructors), 35
n4m_asls
(n4m_estimator_role_constructors), 35
n4m_augment (n4m_procedures), 71
n4m_augmentation_apply, 35
n4m_bagging_pls
(n4m_estimator_role_constructors), 35
n4m_band_mask
(n4m_estimator_role_constructors), 35
n4m_band_perturb
(n4m_estimator_role_constructors), 35
n4m_baseline
(n4m_estimator_role_constructors), 35
n4m_batch_effect
(n4m_estimator_role_constructors), 35
n4m_beads
(n4m_estimator_role_constructors), 35
n4m_binned_strat_group_kfold
(n4m_estimator_role_constructors), 35
n4m_bipls
(n4m_estimator_role_constructors), 35
n4m_boosting_pls
(n4m_estimator_role_constructors), 35
n4m_bve
(n4m_estimator_role_constructors), 35
n4m_calibration
(n4m_estimator_role_constructors), 35
n4m_cars
(n4m_estimator_role_constructors), 35
n4m_channel_dropout
(n4m_estimator_role_constructors), 35
n4m_classes (n4m_estimator_roles), 57
n4m_classes.n4m_role_pipeline
(n4m_role_pipeline), 72
n4m_close (n4m_multimodal_pipeline), 66
n4m_constraint (n4m_optimizer), 67
n4m_constructor (n4m_manifest), 60
n4m_contains_training_rows
(n4m_estimator_roles), 57
n4m_continuum_regression
(n4m_estimator_role_constructors), 35
n4m_correlation
(n4m_estimator_role_constructors), 35
n4m_cow_align
(n4m_estimator_role_constructors), 35
n4m_cppls
(n4m_estimator_role_constructors), 35
n4m_crop
(n4m_estimator_role_constructors), 35
n4m_dead_band
(n4m_estimator_role_constructors), 35
n4m_derivate
(n4m_estimator_role_constructors),

35
n4m_detector_rolloff
(n4m_estimator_role_constructors), 35
n4m_detrend
(n4m_estimator_role_constructors), 35
n4m_di_pls
(n4m_estimator_role_constructors), 35
n4m_direct_standardization
(n4m_estimator_role_constructors), 35
n4m_ds
(n4m_estimator_role_constructors), 35
n4m_dtw_align
(n4m_estimator_role_constructors), 35
n4m_ecr
(n4m_estimator_role_constructors), 35
n4m_edge_artifacts
(n4m_estimator_role_constructors), 35
n4m_edge_curvature
(n4m_estimator_role_constructors), 35
n4m_emcuve
(n4m_estimator_role_constructors), 35
n4m_emsc
(n4m_estimator_role_constructors), 35
n4m_emsc_distort
(n4m_estimator_role_constructors), 35
n4m_epo
(n4m_estimator_role_constructors), 35
n4m_estimator_export
(n4m_estimator_roles), 57
n4m_estimator_fit, *57*
n4m_estimator_fit
(n4m_estimator_roles), 57
n4m_estimator_fit.n4m_multimodal_pipeline
(n4m_multimodal_pipeline), 66
n4m_estimator_fit.n4m_role_pipeline
(n4m_role_pipeline), 72
n4m_estimator_import
(n4m_estimator_roles), 57
n4m_estimator_role_constructors, 35, *58*, *72*
n4m_estimator_roles, 57
n4m_estimator_transform
(n4m_estimator_roles), 57
n4m_estimator_transform.n4m_multimodal_pipeline
(n4m_multimodal_pipeline), 66
n4m_estimator_transform.n4m_role_pipeline
(n4m_role_pipeline), 72
n4m_export_state
(n4m_multimodal_pipeline), 66
n4m_fck_static
(n4m_estimator_role_constructors), 35
n4m_feature_select_bipls
(bipls_select), 9
n4m_feature_select_bve (bve_select), 11
n4m_feature_select_cars (cars_select), 11
n4m_feature_select_coefficient
(coefficient_select), 12
n4m_feature_select_emcuve
(emcuve_select), 19
n4m_feature_select_ga (ga_select), 21
n4m_feature_select_interval
(interval_select), 23
n4m_feature_select_ipw (ipw_select), 23
n4m_feature_select_irf (irf_select), 24
n4m_feature_select_iriv (iriv_select), 25
n4m_feature_select_pso (pso_select), 89
n4m_feature_select_random_frog
(random_frog_select), 90
n4m_feature_select_randomization
(randomization_select), 92
n4m_feature_select_rep (rep_select), 94
n4m_feature_select_scars
(scars_select), 99
n4m_feature_select_selectivity_ratio
(selectivity_ratio_select), 100
n4m_feature_select_shaving
(shaving_select), 100
n4m_feature_select_sipls
(sipls_select), 101
n4m_feature_select_spa (spa_select), 103

n4m_feature_select_st (st_select), 106
n4m_feature_select_stability (stability_select), 107
n4m_feature_select_t2 (t2_select), 107
n4m_feature_select_uve (uve_select), 108
n4m_feature_select_vip (vip_select), 108
n4m_feature_select_vip_spa (vip_spa_select), 109
n4m_feature_select_vissa (vissa_select), 109
n4m_feature_select_wvc (wvc_select), 111
n4m_feature_select_wvc_threshold (wvc_threshold_select), 112
n4m_first_derivative (n4m_estimator_role_constructors), 35
n4m_fit, 59
n4m_flexible_pca (n4m_estimator_role_constructors), 35
n4m_flexible_svd (n4m_estimator_role_constructors), 35
n4m_fraction_to_percent (n4m_estimator_role_constructors), 35
n4m_from_absorbance (n4m_estimator_role_constructors), 35
n4m_fused_sparse_pls (n4m_estimator_role_constructors), 35
n4m_ga (n4m_estimator_role_constructors), 35
n4m_gauss_jitter (n4m_estimator_role_constructors), 35
n4m_gaussian (n4m_estimator_role_constructors), 35
n4m_gaussian_noise (n4m_estimator_role_constructors), 35
n4m_gpr_pls (n4m_estimator_role_constructors), 35
n4m_group_sparse_pls (n4m_estimator_role_constructors), 35
n4m_haar (n4m_estimator_role_constructors), 35
n4m_hetero_noise (n4m_estimator_role_constructors), 35
n4m_high_leverage (n4m_estimator_role_constructors), 35
n4m_hotelling_t2 (n4m_estimator_role_constructors), 35
n4m_iasls (n4m_estimator_role_constructors), 35
n4m_icoshift_align (n4m_estimator_role_constructors), 35
n4m_imodpoly (n4m_estimator_role_constructors), 35
n4m_instrument_broaden (n4m_estimator_role_constructors), 35
n4m_interval (n4m_estimator_role_constructors), 35
n4m_ipw (n4m_estimator_role_constructors), 35
n4m_irf (n4m_estimator_role_constructors), 35
n4m_iriv (n4m_estimator_role_constructors), 35
n4m_kbins_discretizer (n4m_estimator_role_constructors), 35
n4m_kbins_stratified (n4m_estimator_role_constructors), 35
n4m_kennard_stone (n4m_estimator_role_constructors), 35
n4m_kernel

(n4m_estimator_role_constructors), 35
n4m_kmeans (n4m_estimator_role_constructors), 35
n4m_kubelka_munk (n4m_estimator_role_constructors), 35
n4m_linear_drift (n4m_estimator_role_constructors), 35
n4m_linear_stack_compress (n4m_estimator_role_constructors), 35
n4m_local_centering (n4m_estimator_role_constructors), 35
n4m_local_clip (n4m_estimator_role_constructors), 35
n4m_local_density (n4m_estimator_role_constructors), 35
n4m_local_mixup (n4m_estimator_role_constructors), 35
n4m_local_snv (n4m_estimator_role_constructors), 35
n4m_local_warp (n4m_estimator_role_constructors), 35
n4m_localized_msc (n4m_estimator_role_constructors), 35
n4m_log_transform (n4m_estimator_role_constructors), 35
n4m_lw_pls (n4m_estimator_role_constructors), 35
n4m_magnitude_warp (n4m_estimator_role_constructors), 35
n4m_manifest, 60
n4m_manifest_json (n4m_manifest), 60
n4m_mb_pls (n4m_estimator_role_constructors), 35
n4m_method, 60
n4m_mir_pls (n4m_estimator_role_constructors), 35
n4m_missing_aware_nipals (n4m_estimator_role_constructors), 35
n4m_mixup (n4m_estimator_role_constructors), 35
n4m_model_descriptor, 62
n4m_model_export, 62, *63*
n4m_model_import, *62*, 63, *64*
n4m_model_import_linear_predictor, 63
n4m_model_inspect, 64
n4m_model_pipeline_info, 64
n4m_model_selection (n4m_estimator_role_constructors), 35
n4m_model_selection_kennard_stone (kennard_stone_split), 26
n4m_modpoly (n4m_estimator_role_constructors), 35
n4m_moisture (n4m_estimator_role_constructors), 35
n4m_moments (n4m_estimator_role_constructors), 35
n4m_msc (n4m_estimator_role_constructors), 35
n4m_multimodal_classifier, 65
n4m_multimodal_classifier_from_state (n4m_multimodal_classifier), 65
n4m_multimodal_pipeline, 66
n4m_multimodal_pipeline_from_state (n4m_multimodal_pipeline), 66
n4m_multiplicative_noise (n4m_estimator_role_constructors), 35
n4m_normalize (n4m_estimator_role_constructors), 35
n4m_norris_williams (n4m_estimator_role_constructors),

35
n4m_o2pls (n4m_estimator_role_constructors), 35
n4m_on_pls (n4m_estimator_role_constructors), 35
n4m_operator_pls_stack (n4m_estimator_role_constructors), 35
n4m_optimizer, 67
n4m_optimizer_ask (n4m_optimizer), 67
n4m_optimizer_ask_batch (n4m_optimizer), 67
n4m_optimizer_best (n4m_optimizer), 67
n4m_optimizer_close (n4m_optimizer), 67
n4m_optimizer_enqueue (n4m_optimizer), 67
n4m_optimizer_intermediate (n4m_optimizer), 67
n4m_optimizer_load (n4m_optimizer), 67
n4m_optimizer_save (n4m_optimizer), 67
n4m_optimizer_tell (n4m_optimizer), 67
n4m_optimizer_trials (n4m_optimizer), 67
n4m_osc (n4m_estimator_role_constructors), 35
n4m_param_categorical (n4m_optimizer), 67
n4m_param_float (n4m_optimizer), 67
n4m_param_int (n4m_optimizer), 67
n4m_param_ordinal (n4m_optimizer), 67
n4m_param_sorted_tuple (n4m_optimizer), 67
n4m_particle_size (n4m_estimator_role_constructors), 35
n4m_path_length (n4m_estimator_role_constructors), 35
n4m_pca_reference_qc (n4m_estimator_role_constructors), 35
n4m_pcr (n4m_estimator_role_constructors), 35
n4m_pds (n4m_estimator_role_constructors), 35
n4m_percent_to_fraction (n4m_estimator_role_constructors), 35
n4m_piecewise_direct_standardization (n4m_estimator_role_constructors), 35
n4m_piecewise_msc (n4m_estimator_role_constructors), 35
n4m_piecewise_snv (n4m_estimator_role_constructors), 35
n4m_pls_cox (n4m_estimator_role_constructors), 35
n4m_pls_diagnostics (n4m_estimator_role_constructors), 35
n4m_pls_fit_simple (n4m_estimator_role_constructors), 35
n4m_pls_glm (n4m_estimator_role_constructors), 35
n4m_pls_lda (n4m_estimator_role_constructors), 35
n4m_pls_logistic (n4m_estimator_role_constructors), 35
n4m_pls_monitoring (n4m_estimator_role_constructors), 35
n4m_pls_qda (n4m_estimator_role_constructors), 35
n4m_pls_regression (n4m_estimator_role_constructors), 35
n4m_poly_drift (n4m_estimator_role_constructors), 35
n4m_pop_pls (n4m_estimator_role_constructors), 35
n4m_predict, 69
n4m_preprocess_capabilities

(n4m_preprocess_fit), 70
n4m_preprocess_export
(n4m_preprocess_fit), 70
n4m_preprocess_fit, 70
n4m_preprocess_import
(n4m_preprocess_fit), 70
n4m_preprocess_plan
(n4m_preprocess_fit), 70
n4m_preprocess_step
(n4m_preprocess_fit), 70
n4m_preprocess_transform
(n4m_preprocess_fit), 70
n4m_procedures, 71
n4m_pso
(n4m_estimator_role_constructors), 35
n4m_q_residuals
(n4m_estimator_role_constructors), 35
n4m_ragged_summary
(n4m_estimator_role_constructors), 35
n4m_random_frog
(n4m_estimator_role_constructors), 35
n4m_random_subspace_pls
(n4m_estimator_role_constructors), 35
n4m_random_x_op
(n4m_estimator_role_constructors), 35
n4m_randomization
(n4m_estimator_role_constructors), 35
n4m_range_discretizer
(n4m_estimator_role_constructors), 35
n4m_recursive
(n4m_estimator_role_constructors), 35
n4m_regression_metrics
(n4m_estimator_role_constructors), 35
n4m_regression_ridge (ridge_fit), 95
n4m_rep
(n4m_estimator_role_constructors), 35
n4m_resample_transformer
(n4m_estimator_role_constructors), 35
n4m_resampler
(n4m_estimator_role_constructors), 35
n4m_ridge
(n4m_estimator_role_constructors), 35
n4m_ridge_active_superblock
(n4m_estimator_role_constructors), 35
n4m_ridge_blender
(n4m_estimator_role_constructors), 35
n4m_ridge_global
(n4m_estimator_role_constructors), 35
n4m_ridge_mkl_superblock
(n4m_estimator_role_constructors), 35
n4m_ridge_pls
(n4m_estimator_role_constructors), 35
n4m_ridge_superblock
(n4m_estimator_role_constructors), 35
n4m_robust_direct_standardization
(n4m_estimator_role_constructors), 35
n4m_robust_hpo
(n4m_estimator_role_constructors), 35
n4m_robust_pls
(n4m_estimator_role_constructors), 35
n4m_robust_snv
(n4m_estimator_role_constructors), 35
n4m_role_pipeline, 72
n4m_role_pipeline_export
(n4m_role_pipeline), 72
n4m_role_pipeline_import
(n4m_role_pipeline), 72
n4m_role_pipeline_info
(n4m_role_pipeline), 72
n4m_role_pipeline_steps
(n4m_role_pipeline), 72
n4m_rolling_ball

(n4m_estimator_role_constructors), 35
n4m_rosa (n4m_estimator_role_constructors), 35
n4m_rotate_translate (n4m_estimator_role_constructors), 35
n4m_run (n4m_procedures), 71
n4m_sample_mask (n4m_estimator_roles), 57
n4m_saps (n4m_estimator_role_constructors), 35
n4m_savitzky_golay (n4m_estimator_role_constructors), 35
n4m_scars (n4m_estimator_role_constructors), 35
n4m_scatter_sim_msc (n4m_estimator_role_constructors), 35
n4m_second_derivative (n4m_estimator_role_constructors), 35
n4m_selected_indices (n4m_estimator_roles), 57
n4m_shaving (n4m_estimator_role_constructors), 35
n4m_signal_type_detector (n4m_estimator_role_constructors), 35
n4m_simple_scale (n4m_estimator_role_constructors), 35
n4m_sipls (n4m_estimator_role_constructors), 35
n4m_slope_bias (n4m_estimator_role_constructors), 35
n4m_snip (n4m_estimator_role_constructors), 35
n4m_snv (n4m_estimator_role_constructors), 35
n4m_so_pls (n4m_estimator_role_constructors), 35
n4m_spa (n4m_estimator_role_constructors), 35
n4m_sparse_pls_da (n4m_estimator_role_constructors), 35
n4m_sparse_simpls (n4m_estimator_role_constructors), 35
n4m_spectral_events_qc (n4m_estimator_role_constructors), 35
n4m_spectral_noise_proxy (n4m_estimator_role_constructors), 35
n4m_spectral_quality (n4m_estimator_role_constructors), 35
n4m_spectral_reference_qc (n4m_estimator_role_constructors), 35
n4m_spike_noise (n4m_estimator_role_constructors), 35
n4m_spline_curve_simplification (n4m_estimator_role_constructors), 35
n4m_spline_smoothing (n4m_estimator_role_constructors), 35
n4m_spline_x_perturbations (n4m_estimator_role_constructors), 35
n4m_spline_x_simplification (n4m_estimator_role_constructors), 35
n4m_spline_y_perturbations (n4m_estimator_role_constructors), 35
n4m_split (n4m_procedures), 71
n4m_split_splitter (n4m_estimator_role_constructors), 35
n4m_splitter_run, 74

n4m_spxy
(n4m_estimator_role_constructors), 35
n4m_spxy_fold
(n4m_estimator_role_constructors), 35
n4m_spxy_g_fold
(n4m_estimator_role_constructors), 35
n4m_st
(n4m_estimator_role_constructors), 35
n4m_stability
(n4m_estimator_role_constructors), 35
n4m_standard_scale
(n4m_estimator_role_constructors), 35
n4m_stray_light
(n4m_estimator_role_constructors), 35
n4m_sweep
(n4m_estimator_role_constructors), 35
n4m_systematic_circular
(n4m_estimator_role_constructors), 35
n4m_t2
(n4m_estimator_role_constructors), 35
n4m_temperature
(n4m_estimator_role_constructors), 35
n4m_tensor_pls
(n4m_estimator_role_constructors), 35
n4m_to_absorbance
(n4m_estimator_role_constructors), 35
n4m_transfer_metrics
(n4m_estimator_role_constructors), 35
n4m_transform_savgol (snv_transform), 102
n4m_transform_snv (snv_transform), 102
n4m_truncated_peak
(n4m_estimator_role_constructors), 35
n4m_unsharp_mask
(n4m_estimator_role_constructors), 35
n4m_uve
(n4m_estimator_role_constructors), 35
n4m_variable_select
(n4m_estimator_role_constructors), 35
n4m_variance
(n4m_estimator_role_constructors), 35
n4m_version, 75
n4m_vip_spa
(n4m_estimator_role_constructors), 35
n4m_vissa
(n4m_estimator_role_constructors), 35
n4m_vsn
(n4m_estimator_role_constructors), 35
n4m_wavelength_shift
(n4m_estimator_role_constructors), 35
n4m_wavelength_stretch
(n4m_estimator_role_constructors), 35
n4m_wavelet
(n4m_estimator_role_constructors), 35
n4m_wavelet_denoise
(n4m_estimator_role_constructors), 35
n4m_wavelet_features
(n4m_estimator_role_constructors), 35
n4m_wavelet_pca
(n4m_estimator_role_constructors), 35
n4m_wavelet_svd
(n4m_estimator_role_constructors), 35
n4m_weighted_pls
(n4m_estimator_role_constructors), 35
n4m_weighted_snv
(n4m_estimator_role_constructors),

35
n4m_wvc
(n4m_estimator_role_constructors), 35
n4m_wvc_threshold
(n4m_estimator_role_constructors), 35
n4m_x_outlier
(n4m_estimator_role_constructors), 35
n4m_xcorr_align
(n4m_estimator_role_constructors), 35
n4m_y_outlier
(n4m_estimator_role_constructors), 35
n_pls_fit, 32

o2pls, 75
o2pls_fit, 76
on_pls_fit, 76
one_se_rule, 77
opls, 77

pcr (plsr), 85
pds_fit, 78
pls, 78
pls_cox_fit, 80
pls_diagnostics, 80
pls_glm, 81
pls_glm_fit, 81
pls_lda_fit, 82
pls_logistic_fit, 82
pls_mdatools, 83
pls_monitoring, 84
pls_qda_fit, 85
plsr, 85
pop_pls (aom_pls), 5
poppls (aom_pls), 5
predict.n4m_affine_fit
(n4m_affine_fit), 33
predict.n4m_classifier
(n4m_estimator_roles), 57
predict.n4m_fit, 88
predict.n4m_mdatools_pls
(pls_mdatools), 83
predict.n4m_method_fit, 88
predict.n4m_multimodal_pipeline
(n4m_multimodal_pipeline), 66
predict.n4m_mvr (plsr), 85
predict.n4m_regressor
(n4m_estimator_roles), 57
predict.n4m_role_pipeline
(n4m_role_pipeline), 72
print.n4m_method (n4m_estimator_roles), 57
print.n4m_role_pipeline
(n4m_role_pipeline), 72
pso_select, 89

R2 (plsr), 85
random_frog_select, 90
random_subspace_pls, 91
random_subspace_pls_fit, 92
randomization_select, 92
recursive_pls, 93
recursive_pls_fit, 94
rep_select, 94
ridge_fit, 95
ridge_pls, 95
ridge_pls_fit, 95, 96
RMSEP (plsr), 85
robust_pls, 97
robust_pls_fit, 97
robust_snv_transform, 98
rosa_fit, 99

savgol_transform (snv_transform), 102
scars_select, 99
selectivity_ratio_select, 100
selectNcomp (plsr), 85
shaving_select, 100
sipls_select, 101
snv_transform, 102
so_pls_fit, 103
spa_select, 103
sparse_pls, 104
sparse_pls_da_fit, 104
sparse_simpls_fit, 105
spectral_encoding, 105
spectral_export_affine
(spectral_encoding), 105
spectral_transform (spectral_encoding), 105
st_select, 106
stability_select, 107

t2_select, 107

uve_select, 108

vip_select, 108
vip_spa_select, 109
vissa_select, 109

weighted_pls, 110
weighted_pls_fit, 111
wvc_select, 111
wvc_threshold_select, 112