Package: bigPLSR (via r-universe)

May 31, 2026

Version 0.7.2

Date 2025-11-26

Depends R (>= 4.0.0)

Imports Rcpp, bigmemory

LinkingTo Rcpp, RcppArmadillo, BH, bigmemory

Suggests bench, dplyr, forcats, future, future.apply, ggplot2, knitr, pls, plsRglm, rmarkdown, RhpcBLASctl, svglite, testthat (>= 3.0.0), tidyr, withr

VignetteBuilder knitr

Title Partial Least Squares Regression Models with Big Matrices

Author Frederic Bertrand [cre, aut] (<`https://orcid.org/0000-0002-0837-8281`>), Myriam Maumy [aut] (<`https://orcid.org/0000-0002-4615-1512`>)

Maintainer Frederic Bertrand <`frederic.bertrand@lecnam.net`>

Description Fast partial least squares (PLS) for dense and out-of-core data. Provides SIMPLS (straightforward implementation of a statistically inspired modification of the PLS method) and NIPALS (non-linear iterative partial least-squares) solvers, plus kernel-style PLS variants ('kernelpls' and 'widekernelpls') with parity to 'pls'. Optimized for 'bigmemory'-backed matrices with streamed cross-products and chunked BLAS (Basic Linear Algebra Subprograms) (XtX/XtY and XXt/YX), optional file-backed score sinks, and deterministic testing helpers. Includes an auto-selection strategy that chooses between XtX SIMPLS, XXt (wide) SIMPLS, and NIPALS based on (n, p) and a configurable memory budget. About the package, Bertrand and Maumy (2023) <`https://hal.science/hal-05352069`>, and <`https://hal.science/hal-05352061`> highlighted fitting and cross-validating PLS regression models to big data. For more details about some of the techniques featured in the package, Dayal and MacGregor (1997) <doi:10.1002/(SICI)1099-128X(199701)11:1%3C73::AID-CEM435%3E3.0.CO;2-%23>, Rosipal & Trejo (2001)

<https://www.jmlr.org/papers/v2/rosipal01a.html>, Tenenhaus, Viennet, and Saporta (2007) <doi:10.1016/j.csda.2007.01.004>, Rosipal (2004) <doi:10.1007/978-3-540-45167-9_17>, Rosipal (2019) <https://ieeexplore.ieee.org/document/8616346>, Song, Wang, and Bai (2024) <doi:10.1016/j.chemolab.2024.105238>. Includes kernel logistic PLS with 'C++'-accelerated alternating iteratively reweighted least squares (IRLS) updates, streamed reproducing kernel Hilbert space (RKHS) solvers with reusable centering statistics, and bootstrap diagnostics with graphical summaries for coefficients, scores, and cross-validation workflows, alongside dedicated plotting utilities for individuals, variables, ellipses, and biplots. The streaming backend uses far less memory and keeps memory bounded across data sizes. For PLS1, streaming is often fast enough while preserving a small memory footprint; for PLS2 it remains competitive with a bounded footprint. On small problems that fit comfortably in RAM (random-access memory), dense in-memory solvers are slightly faster; the crossover occurs as n or p grow and the Gram/cross-product cost dominates.

License GPL-3

Encoding UTF-8

URL https://fbertran.github.io/bigPLSR/,
https://github.com/fbertran/bigPLSR

BugReports https://github.com/fbertran/bigPLSR/issues

Classification/MSC 62N01, 62N02, 62N03, 62N99

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LazyData true

Config/testthat/edition 3

SystemRequirements C++17, Optional CBLAS (detected at compile time)

NeedsCompilation yes

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

Date/Publication 2025-11-26 01:49:56 UTC

RemoteUrl https://github.com/fbertran/bigplsr

RemoteRef HEAD

RemoteSha 2033eb023bea3771b67270a4f81fe3d922b587de

Contents

`bigPLSR-package` *bigPLSR-package*

Description

Provides Partial least squares Regression for big data. It allows for missing data in the explanatory variables. Repeated k-fold cross-validation of such models using various criteria. Bootstrap confidence intervals constructions are also available.

Author(s)

Maintainer: Frederic Bertrand `<frederic.bertrand@lecnam.net>` (ORCID)

Authors:

- Myriam Maumy `<myriam.maumy@ehesp.fr>` (ORCID)

References

Maumy, M., Bertrand, F. (2023). PLS models and their extension for big data. Joint Statistical Meetings (JSM 2023), Toronto, ON, Canada.

Maumy, M., Bertrand, F. (2023). bigPLS: Fitting and cross-validating PLS-based Cox models to censored big data. BioC2023 — The Bioconductor Annual Conference, Dana-Farber Cancer Institute, Boston, MA, USA. Poster. https://doi.org/10.7490/f1000research.1119546.1

See Also

Useful links:

- `https://fbertran.github.io/bigPLSR/`
- `https://github.com/fbertran/bigPLSR`
- Report bugs at `https://github.com/fbertran/bigPLSR/issues`

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r", algorithm = "simpls")
head(pls_predict_response(fit, X, ncomp = 2))
```

`.finalize_pls_fit` *Finalize pls objects*

Description

Finalize pls objects

Usage

```
.finalize_pls_fit(fit, algorithm)
```

Arguments

`fit`	Fitted object
`algorithm`	Name of the algorithm used to fit the object

Value

The `fit` object with normalized naming and class attributes.

`bigPLSR_stream_kstats` *Streamed centering statistics for RKHS kernels*

Description

Compute the column means and grand mean of the kernel matrix $K(X, X)$ without materialising it in memory. The input design matrix must be stored as a `bigmemory::big.matrix` (or descriptor), and the kernel is evaluated by iterating over row/column chunks.

Usage

```
bigPLSR_stream_kstats(
  Xbm,
  kernel,
  gamma,
  degree,
  coef0,
  chunk_rows = getOption("bigPLSR.predict.chunk_rows", 8192L),
  chunk_cols = getOption("bigPLSR.predict.chunk_cols", 8192L)
)
```

Arguments

`Xbm`	A `bigmemory::big.matrix` (or descriptor) containing the training design matrix.
`kernel`	Kernel name passed to `stats::kernel()` compatible helpers (`"linear"`, `"rbf"`, `"poly"`, `"sigmoid"`).
`gamma, degree, coef0`	Kernel hyper-parameters.
`chunk_rows, chunk_cols`	Numbers of rows/columns to process per chunk.

Value

A list with entries `r` (column means) and `g` (grand mean) of the kernel matrix.

`cpp_irls_binomial` *Fast IRLS for binomial logit with class weights*

Description

Fast IRLS for binomial logit with class weights

Usage

```
cpp_irls_binomial(TT, ybin, w_class = NULL, maxit = 50L, tol = 1e-08)
```

Arguments

`TT`	n x A numeric matrix of latent scores (no intercept column)
`ybin`	integer vector of {0,1} labels (length n)
`w_class`	optional length-2 numeric vector: weights for classes c(w0, w1)
`maxit`	max IRLS iterations
`tol`	relative tolerance on parameter change

Value

list(beta = A-vector, b = scalar intercept, fitted = n-vector, iter = integer, converged = logical)

`cpp_kernel_pls` *Internal kernel and wide-kernel PLS solver*

Description

Internal kernel and wide-kernel PLS solver

Usage

```
cpp_kernel_pls(X, Y, ncomp, tol, wide)
```

Arguments

`X`	Centered design matrix.
`Y`	Centered response matrix.
`ncomp`	Maximum number of components.
`tol`	Numerical tolerance.
`wide`	Whether to use the wide-kernel update.

Value

A list containing the kernel PLS factors.

`external_pls_benchmarks`

Benchmark results against external PLS implementations

Description

Pre-computed runtime comparisons between **bigPLSR** (dense and big.memory backends) and reference implementations from the **pls** and **mixOmics** packages.

Usage

```
data(external_pls_benchmarks)
```

Format

A data frame with 384 rows and 11 columns:

task Character vector identifying the task (`"pls1"` or `"pls2"`).

algorithm PLS algorithm used for the benchmark (e.g., `"simpls"`).

package Package providing the implementation.

median_time_s Median execution time in seconds.

itr_per_sec Iterations per second recorded by `bench::mark()`.

mem_alloc_bytes Memory usage in bytes recorded by `bench::mark()`.

n Number of observations in the simulated dataset.

p Number of predictors (X) in the simulated dataset.

q Number of responses (Y) in the simulated dataset.

ncomp Number of extracted components.

notes Helpful context on dependencies or configuration.

Details

Fix `task = "pls1"` and select algorithms in `"kernelpls"`, `"nipals"` or `"simpls"` to get a full factorial design. Fix `task = "pls1"` and fix `algorithm = "widekernelpls"` to get a full factorial design. Fix `task = "pls2"` and select algorithms in `"kernelpls"`, `"nipals"` or `"simpls"` to get a full factorial design. Fix `task = "pls2"` and fix `algorithm = "widekernelpls"` to get a full factorial design.

Source

Generated via `inst/scripts/external_pls_benchmarks.R`.

Examples

```
data("external_pls_benchmarks", package = "bigPLSR")

sub_pls1 <- subset(external_pls_benchmarks, task == "pls1" &
                                             algorithm != "widekernelpls")
sub_pls1$n     <- factor(sub_pls1$n)
sub_pls1$p     <- factor(sub_pls1$p)
sub_pls1$q     <- factor(sub_pls1$q)
sub_pls1$ncomp <- factor(sub_pls1$ncomp)
if (exists("replications")) replications(~ package + algorithm + task + n +
                                           p + ncomp, data = sub_pls1)

sub_pls1_wide <- subset(external_pls_benchmarks, task == "pls1" &
                                                  algorithm == "widekernelpls")
sub_pls1_wide$n     <- factor(sub_pls1_wide$n)
sub_pls1_wide$p     <- factor(sub_pls1_wide$p)
sub_pls1_wide$q     <- factor(sub_pls1_wide$q)
sub_pls1_wide$ncomp <- factor(sub_pls1_wide$ncomp)
if (exists("replications")) replications(~ package + algorithm + task + n +
                                           p + ncomp, data = sub_pls1_wide)

sub_pls2 <- subset(external_pls_benchmarks, task == "pls2" &
                                             algorithm != "widekernelpls")
sub_pls2$n     <- factor(sub_pls2$n)
sub_pls2$p     <- factor(sub_pls2$p)
sub_pls2$q     <- factor(sub_pls2$q)
sub_pls2$ncomp <- factor(sub_pls2$ncomp)
if (exists("replications")) replications(~ package + algorithm + task + n +
                                           p + ncomp, data = sub_pls2)

sub_pls2_wide <- subset(external_pls_benchmarks, task == "pls2" &
                                                  algorithm == "widekernelpls")
sub_pls2_wide$n     <- factor(sub_pls2_wide$n)
sub_pls2_wide$p     <- factor(sub_pls2_wide$p)
sub_pls2_wide$q     <- factor(sub_pls2_wide$q)
sub_pls2_wide$ncomp <- factor(sub_pls2_wide$ncomp)
if (exists("replications")) replications(~ package + algorithm + task + n +
                                           p + ncomp, data = sub_pls2_wide)
```

`kf_pls_state_fit` *Finalize a KF-PLS state into a fitted model*

Description

Converts the accumulated KF-PLS state into a SIMPLS-equivalent fitted model (using the current sufficient statistics). The result is compatible with `predict.big_plsr()`.

Usage

```
kf_pls_state_fit(state, tol = 1e-08)
```

Arguments

`state`	External pointer created by `kf_pls_state_new()`.
`tol`	Numeric tolerance for the inner SIMPLS step.

Value

A list with PLS factors and coefficients, classed as `big_plsr`.

Examples

```
n <- 200; p <- 30; m <- 2; A <- 3
X <- matrix(rnorm(n*p), n, p)
Y <- X[,1:2] %*% matrix(c(0.7, -0.3, 0.2, 0.9), 2, m) + matrix(rnorm(n*m, sd=0.2), n, m)

state <- kf_pls_state_new(p, m, A, lambda = 0.99, q_proc = 1e-6)

# stream in mini-batches
bs <- 64
for (i in seq(1, n, by = bs)) {
  idx <- i:min(i+bs-1, n)
  kf_pls_state_update(state, X[idx, , drop=FALSE], Y[idx, , drop=FALSE])
}

fit <- kf_pls_state_fit(state)  # returns a big_plsr-compatible list
# predict via your existing predict.big_plsr (linear case)
Yhat <- cbind(1, scale(X, center = fit$x_means, scale = FALSE)) %*%
  rbind(fit$intercept, fit$coefficients)
```

`kf_pls_state_new` *KF-PLS streaming state (constructor)*

Description

Create a persistent Kalman–filter PLS (KF-PLS) state that accumulates cross-products from streaming mini-batches and later produces a `big_plsr`-compatible fit via `kf_pls_state_fit()`.

Usage

```
kf_pls_state_new(p, m, ncomp, lambda = 0.99, q_proc = 0, r_meas = 0)
```

Arguments

`p`	Integer, number of predictors (columns of `X`).
`m`	Integer, number of responses (columns of `Y`).
`ncomp`	Integer, number of latent components to extract at fit time.
`lambda`	Numeric in (0,1], forgetting factor (closer to 1 = slower decay).

`q_proc`	Non-negative numeric, process-noise magnitude (adds a ridge to C_{xx} each update; useful for stabilizing ill-conditioned problems).
`r_meas`	Reserved measurement-noise parameter (not used by the minimal API yet; kept for forward compatibility).

Details

The state maintains exponentially weighted cross-moments C_{xx} and C_{xy} with forgetting factor `lambda`. When `lambda >= 0.999999` and `q_proc == 0`, the backend switches to an *exact* accumulation mode that matches concatenating all chunks (no decay).

Value

An external pointer to an internal KF-PLS state (opaque object) that you pass to `kf_pls_state_update()` and then to `kf_pls_state_fit()` to produce model coefficients.

See Also

`kf_pls_state_update()`, `kf_pls_state_fit()`, `pls_fit()` (use `algorithm = "kf_pls"` for the one-shot dense path).

Examples

```
set.seed(1)
n <- 1000; p <- 50; m <- 2
X1 <- matrix(rnorm(n/2 * p), n/2, p)
X2 <- matrix(rnorm(n/2 * p), n/2, p)
B  <- matrix(rnorm(p*m), p, m)
Y1 <- scale(X1, TRUE, FALSE) %*% B + 0.05*matrix(rnorm(n/2*m), n/2, m)
Y2 <- scale(X2, TRUE, FALSE) %*% B + 0.05*matrix(rnorm(n/2*m), n/2, m)

st <- kf_pls_state_new(p, m, ncomp = 4, lambda = 0.99, q_proc = 1e-6)
kf_pls_state_update(st, X1, Y1)
kf_pls_state_update(st, X2, Y2)
fit <- kf_pls_state_fit(st)          # returns a big_plsr-compatible list
preds <- predict(bigPLSR::.finalize_pls_fit(fit, "kf_pls"), rbind(X1, X2))
head(preds)
```

`kf_pls_state_update` *Update a KF-PLS streaming state with a mini-batch*

Description

Feed one chunk (`X_chunk`, `Y_chunk`) to an existing KF-PLS state created by `kf_pls_state_new()`. The function updates exponentially weighted means and cross-products (or exact sufficient statistics when in exact mode).

Usage

```
kf_pls_state_update(state, X_chunk, Y_chunk)
```

Arguments

`state`	External pointer produced by `kf_pls_state_new()`.
`X_chunk`	Numeric matrix with the same number of columns `p` used to create the state.
`Y_chunk`	Numeric matrix with `m` columns (or a numeric vector if `m == 1`). Must have the same number of rows as `X_chunk`.

Details

Call this repeatedly for each incoming batch. When you want model coefficients (weights/loadings/intercepts), call `kf_pls_state_fit()`, which solves SIMPLS on the accumulated cross-moments without re-materializing all past data.

Value

Invisibly returns `state`, updated in place.

See Also

`kf_pls_state_new()`, `kf_pls_state_fit()`

`plot_pls_biplot` *PLS biplot*

Description

PLS biplot

Usage

```
plot_pls_biplot(
  object,
  comps = c(1L, 2L),
  scale_variables = 1,
  circle = TRUE,
  circle_col = "grey85",
  arrow_col = "firebrick",
  groups = NULL,
  ellipse = TRUE,
  ellipse_level = 0.95,
  ellipse_n = 200L,
  group_col = NULL,
  ...
)
```

Arguments

`object`	A fitted PLS model with scores and loadings.
`comps`	Components to display.
`scale_variables`	Scaling factor applied to variable loadings.
`circle`	Logical; draw a unit circle behind loadings.
`circle_col`	Colour of the unit circle guide.
`arrow_col`	Colour for loading arrows.
`groups`	Optional factor or character vector defining groups for individuals. When supplied, group-specific colours are used and, if `ellipse = TRUE`, confidence ellipses are drawn for each group.
`ellipse`	Logical; draw group confidence ellipses when `groups` are provided.
`ellipse_level`	Confidence level for group ellipses (between 0 and 1).
`ellipse_n`	Number of points used to draw each ellipse.
`group_col`	Optional vector of colours for the groups. Recycled as needed.
`...`	Additional arguments passed to `graphics::plot()`.

Value

Invisibly returns `NULL` after drawing the biplot.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
plot_pls_biplot(fit)
```

`plot_pls_bootstrap_coefficients`

Boxplots of bootstrap coefficient distributions

Description

Boxplots of bootstrap coefficient distributions

Usage

```
plot_pls_bootstrap_coefficients(
  boot_result,
  responses = NULL,
  variables = NULL,
  ...
)
```

Arguments

`boot_result`	Result returned by `pls_bootstrap()`.
`responses`	Optional character vector selecting response columns.
`variables`	Optional character vector selecting predictor variables.
`...`	Additional arguments passed to `graphics::boxplot()`.

Value

Invisibly returns `NULL` after drawing the boxplots.

`plot_pls_bootstrap_scores`

Boxplots of bootstrap score distributions

Description

Visualise the variability of latent scores obtained through `pls_bootstrap()` when `return_scores = TRUE`.

Usage

```
plot_pls_bootstrap_scores(
  boot_result,
  components = NULL,
  observations = NULL,
  ...
)
```

Arguments

`boot_result`	Result returned by `pls_bootstrap()`.
`components`	Optional vector of component indices or names to include.
`observations`	Optional vector of observation indices or names to include.
`...`	Additional arguments passed to `graphics::boxplot()`.

Value

Invisibly returns `NULL` after drawing the boxplots.

`plot_pls_individuals` *Plot individual scores*

Description

Plot individual scores

Usage

```
plot_pls_individuals(
  object,
  comps = c(1L, 2L),
  labels = NULL,
  groups = NULL,
  ellipse = TRUE,
  ellipse_level = 0.95,
  ellipse_n = 200L,
  group_col = NULL,
  ...
)
```

Arguments

`object`	A fitted PLS model with scores.
`comps`	Components to plot (length two).
`labels`	Optional character vector of point labels.
`groups`	Optional factor or character vector defining groups for individuals. When supplied, group-specific colours are used and, if `ellipse = TRUE`, confidence ellipses are drawn for each group.
`ellipse`	Logical; draw group confidence ellipses when `groups` are provided.
`ellipse_level`	Confidence level for the ellipses (between 0 and 1).
`ellipse_n`	Number of points used to draw each ellipse.
`group_col`	Optional vector of colours for the groups. Recycled as needed.
`...`	Additional plotting parameters passed to `graphics::plot()`.

Value

Invisibly returns `NULL` after drawing the plot.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
plot_pls_individuals(fit)
```

`plot_pls_variables` *Plot variable loadings*

Description

Plot variable loadings

Usage

```
plot_pls_variables(
  object,
  comps = c(1L, 2L),
  circle = TRUE,
  circle_col = "grey80",
  arrow_col = "steelblue",
  arrow_scale = 1,
  ...
)
```

Arguments

`object`	A fitted PLS model.
`comps`	Components to display (length two).
`circle`	Logical; draw the unit circle.
`circle_col`	Colour of the unit circle.
`arrow_col`	Colour of the variable arrows.
`arrow_scale`	Scaling applied to variable vectors.
`...`	Additional plotting parameters passed to `graphics::plot()`.

Value

Invisibly returns `NULL` after drawing the plot.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
plot_pls_variables(fit)
```

`plot_pls_vip` *Plot Variable Importance in Projection (VIP)*

Description

Plot Variable Importance in Projection (VIP)

Usage

```
plot_pls_vip(
  object,
  comps = NULL,
  threshold = 1,
  palette = c("#4575b4", "#d73027"),
  ...
)
```

Arguments

`object`	A fitted PLS model.
`comps`	Components to aggregate. Defaults to all available.
`threshold`	Optional threshold to highlight influential variables.
`palette`	Colour palette used for bars.
`...`	Additional parameters passed to `graphics::barplot()`.

Value

Invisibly returns the VIP scores used to create the bar plot.

Examples

```
set.seed(123)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(10, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
plot_pls_vip(fit)
```

`pls_bootstrap` *Bootstrap a PLS model*

Description

Draw bootstrap replicates of a fitted PLS model, refitting on each resample.

Usage

```
pls_bootstrap(
  X,
  Y,
  ncomp,
  R = 100L,
  algorithm = c("simpls", "nipals", "kernelpls", "widekernelpls"),
  backend = "arma",
  conf = 0.95,
  seed = NULL,
  type = c("xy", "xt"),
  parallel = c("none", "future"),
  future_seed = TRUE,
  return_scores = FALSE,
  ...
)
```

Arguments

`X`	Predictor matrix.
`Y`	Response matrix or vector.
`ncomp`	Number of components.
`R`	Number of bootstrap replications.
`algorithm`	Backend algorithm ("simpls", "nipals", "kernelpls" or "widekernelpls").
`backend`	Backend argument passed to the fitting routine.
`conf`	Confidence level.
`seed`	Optional seed.
`type`	Character; bootstrap scheme, e.g. `"pairs"`, `"residual"`, or `"parametric"`.
`parallel`	Logical or character; if `TRUE` or one of `c("sequential", "multisession", "multicore")`, uses the future framework.
`future_seed`	Logical or integer; forwarded to `future.seed` for reproducible parallel streams.
`return_scores`	Logical; if `TRUE`, return component scores for each replicate (may be large).
`...`	Additional arguments forwarded to `pls_fit()`.

Value

A list with bootstrap estimates and summaries.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
pls_bootstrap(X, y, ncomp = 2, R = 20)
```

`pls_cross_validate` *Cross-validate PLS models*

Description

Cross-validate PLS models

Usage

```
pls_cross_validate(
  X,
  Y,
  ncomp,
  folds = 5L,
  type = c("kfold", "loo"),
  algorithm = c("simpls", "nipals", "kernelpls", "widekernelpls"),
  backend = "arma",
  metrics = c("rmse", "mae", "r2"),
  seed = NULL,
  parallel = c("none", "future"),
  future_seed = TRUE,
  ...
)
```

Arguments

`X`	Predictor matrix as accepted by `pls_fit()`
`Y`	Response matrix or vector as accepted by `pls_fit()`
`ncomp`	Integer; components grid to evaluate.
`folds`	Number of folds (ignored when `type = "loo"`).
`type`	Either "kfold" (default) or "loo".
`algorithm`	Backend algorithm: "simpls", "nipals", "kernelpls" or "widekernelpls".
`backend`	Backend passed to `pls_fit()`.
`metrics`	Metrics to compute (subset of "rmse", "mae", "r2").
`seed`	Optional seed for reproducibility.

`parallel`	Logical or character; same semantics as in `pls_bootstrap()`.
`future_seed`	Logical or integer; reproducible seeds for parallel evaluation.
`...`	Passed to `pls_fit()`.

Value

A list containing per-fold metrics and their summary across folds.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
pls_cross_validate(X, y, ncomp = 2, folds = 3)
```

`pls_cv_select` *Select components from cross-validation results*

Description

Select components from cross-validation results

Usage

```
pls_cv_select(cv_result, metric = c("rmse", "mae", "r2"), minimise = NULL)
```

Arguments

`cv_result`	Result returned by `pls_cross_validate()`.
`metric`	Metric to optimise.
`minimise`	Logical; whether the metric should be minimised.

Value

Selected number of components.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
cv <- pls_cross_validate(X, y, ncomp = 2, folds = 3)
pls_cv_select(cv, metric = "rmse")
```

`pls_fit` *Unified PLS fit with auto backend and selectable algorithm*

Description

Dispatches to a dense (Arm/BLAS) backend for in-memory matrices or to a streaming big.matrix backend when X (or Y) is a big.matrix. Algorithm can be chosen between: "simpls" (default), "nipals", "kernelpls", "widekernelpls", "rkhs" (Rosipal & Trejo), "klogitpls", "sparse_kpls", "rkhs_xy" (double RKHS), and "kf_pls" (Kalman-filter PLS, streaming).

The "kernelpls" paths now include a streaming XX' variant for big.matrix inputs, with an optional row-chunking loop controlled by `chunk_cols`.

Usage

```
pls_fit(
  X,
  y,
  ncomp,
  tol = 1e-08,
  backend = c("auto", "arma", "bigmem"),
  mode = c("auto", "pls1", "pls2"),
 algorithm = c("auto", "simpls", "nipals", "kernelpls", "widekernelpls", "rkhs",
    "klogitpls", "sparse_kpls", "rkhs_xy", "kf_pls"),
  scores = c("none", "r", "big"),
  chunk_size = 10000L,
  chunk_cols = NULL,
  scores_name = "scores",
  scores_target = c("auto", "new", "existing"),
  scores_bm = NULL,
  scores_backingfile = NULL,
  scores_backingpath = NULL,
  scores_descriptorfile = NULL,
  scores_colnames = NULL,
  return_scores_descriptor = FALSE,
  coef_threshold = NULL,
  kernel = c("linear", "rbf", "poly", "sigmoid"),
  gamma = 1,
  degree = 3L,
  coef0 = 0,
  approx = c("none", "nystrom", "rff"),
  approx_rank = NULL,
  class_weights = NULL
)
```

Arguments

`X`	numeric matrix or `bigmemory::big.matrix`

`y`	numeric vector/matrix or `big.matrix`
`ncomp`	number of latent components
`tol`	numeric tolerance used in the core solver
`backend`	one of `"auto"`, `"arma"`, `"bigmem"`
`mode`	one of `"auto"`, `"pls1"`, `"pls2"`
`algorithm`	one of `"auto"`, `"simpls"`, `"nipals"`, `"kernelpls"`, `"widekernelpls"`, `"rkhs"`, `"klogitpls"`, `"sparse_kpls"`, `"rkhs_xy"`, `"kf_pls"`
`scores`	one of `"none"`, `"r"`, `"big"`
`chunk_size`	chunk size for the bigmem backend
`chunk_cols`	columns chunk size for the bigmem backend
`scores_name`	name for dense scores (or output big.matrix)
`scores_target`	one of `"auto"`, `"new"`, `"existing"`
`scores_bm`	optional existing big.matrix or descriptor for scores
`scores_backingfile`	Character; file name for file-backed scores (when `scores="big"`).
`scores_backingpath`	Character; directory for the file-backed scores. Defaults to `getwd()` or `tempdir()` in streamed predict, unless overridden.
`scores_descriptorfile`	Character; descriptor file name for the file-backed scores.
`scores_colnames`	optional character vector for score column names
`return_scores_descriptor`	logical; if TRUE and scores is big.matrix, add `$scores_descriptor`
`coef_threshold`	Optional non-negative value used to hard-threshold the fitted coefficients after model estimation. When supplied, absolute coefficients strictly below the threshold are set to zero via `pls_threshold()`.
`kernel`	kernel name for RKHS/KPLS (`"linear"`, `"rbf"`, `"poly"`, `"sigmoid"`)
`gamma`	RBF/sigmoid/poly scale parameter
`degree`	polynomial degree
`coef0`	polynomial/sigmoid bias
`approx`	kernel approximation: `"none"`, `"nystrom"`, `"rff"`
`approx_rank`	rank (columns / features) for the approximation
`class_weights`	optional numeric weights for classes in `klogitpls`

Value

a list with coefficients, intercept, weights, loadings, means, and optionally `$scores`.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r", algorithm = "simpls")
head(pls_predict_response(fit, X, ncomp = 2))
```

`pls_information_criteria`

Compute information criteria for component selection

Description

Compute information criteria for component selection

Usage

```
pls_information_criteria(object, X, Y, max_comp = NULL)
```

Arguments

`object`	A fitted PLS model.
`X`	Training design matrix.
`Y`	Training response matrix or vector.
`max_comp`	Maximum number of components to consider.

Value

A data frame with RSS, RMSE, AIC and BIC per component.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
pls_information_criteria(fit, X, y)
```

`pls_predict_response` *Predict responses from a PLS fit*

Description

Predict responses from a PLS fit

Usage

```
pls_predict_response(object, newdata, ncomp = NULL)
```

Arguments

`object`	A fitted PLS model.
`newdata`	Predictor matrix for scoring.
`ncomp`	Number of components to use.

Value

A numeric matrix or vector of predictions.

Examples

```
set.seed(123)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(10, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
pls_predict_response(fit, X, ncomp = 2)
```

`pls_predict_scores` *Predict latent scores from a PLS fit*

Description

Predict latent scores from a PLS fit

Usage

```
pls_predict_scores(object, newdata, ncomp = NULL)
```

Arguments

`object`	A fitted PLS model.
`newdata`	Predictor matrix for scoring.
`ncomp`	Number of components to use.

Value

Matrix of component scores.

Examples

```
set.seed(123)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(10, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
pls_predict_scores(fit, X, ncomp = 2)
```

`pls_select_components` *Component selection via information criteria*

Description

Component selection via information criteria

Usage

```
pls_select_components(
  object,
  X,
  Y,
  criteria = c("aic", "bic"),
  max_comp = NULL
)
```

Arguments

`object`	A fitted PLS model.
`X`	Training design matrix.
`Y`	Training response matrix or vector.
`criteria`	Character vector specifying which criteria to compute.
`max_comp`	Maximum number of components to consider.

Value

A list with the per-component table and the selected components.

Examples

```
set.seed(123)
X <- matrix(rnorm(60), nrow = 20)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(20, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
pls_select_components(fit, X, y)
```

`pls_threshold` *Naive sparsity control by coefficient thresholding*

Description

Naive sparsity control by coefficient thresholding

Usage

```
pls_threshold(object, threshold)
```

Arguments

`object`	A fitted PLS model.
`threshold`	Values below this absolute magnitude are set to zero.

Value

A modified copy of `object` with thresholded coefficients.

Examples

```
set.seed(123)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(10, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2)
pls_threshold(fit, threshold = 0.05)
```

`pls_vip` *Variable importance in projection (VIP) scores*

Description

Variable importance in projection (VIP) scores

Usage

```
pls_vip(object, comps = NULL)
```

Arguments

`object`	A fitted PLS model.
`comps`	Components used to compute the VIP scores. Defaults to all available components.

Value

A named numeric vector of VIP scores.

Examples

```
set.seed(123)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(10, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
pls_vip(fit)
```

`predict.big_plsr` *Predict method for big_plsr objects*

Description

Predict method for big_plsr objects

Usage

```
## S3 method for class 'big_plsr'
predict(
  object,
  newdata,
  ncomp = NULL,
  type = c("response", "scores", "prob", "class"),
  ...
)
```

Arguments

`object`	A fitted PLS model produced by `pls_fit()`.
`newdata`	Matrix or `bigmemory::big.matrix` with predictor values.
`ncomp`	Number of components to use for prediction.
`type`	Either "response" (default) or "scores".
`...`	Unused, for compatibility with the generic.

Value

Predicted responses or component scores.

Examples

```
set.seed(123)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(10, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
predict(fit, X, ncomp = 2)
```

`print.summary.big_plsr`

Print a `summary.big_plsr` *object*

Description

Print a `summary.big_plsr` object

Usage

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

Arguments

`x`	A `summary.big_plsr` object.
`...`	Passed to lower-level print methods.

Value

`x`, invisibly.

Examples

```
set.seed(123)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(10, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
print(summary(fit))
```

`summarise_pls_bootstrap`

Summarise bootstrap estimates

Description

Summarise bootstrap estimates

Usage

```
summarise_pls_bootstrap(boot_result)
```

Arguments

`boot_result`	Result returned by `pls_bootstrap()`.

Value

A data frame containing mean, standard deviation, percentile and BCa confidence intervals for each coefficient.

`summary.big_plsr` *Summarize a* `big_plsr` *model*

Description

Summarize a `big_plsr` model

Usage

```
## S3 method for class 'big_plsr'
summary(object, ..., X = NULL, Y = NULL)
```

Arguments

`object`	A fitted PLS model.
`...`	Unused.
`X`	Optional design matrix to recompute reconstruction metrics.
`Y`	Optional response matrix/vector.

Value

An object of class `summary.big_plsr`.

Examples

```
set.seed(123)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] - 0.5 * X[, 2] + rnorm(10, sd = 0.1)
fit <- pls_fit(X, y, ncomp = 2, scores = "r")
summary(fit)
```

Index