Package: fastPLS (via r-universe)

July 8, 2026

Type Package

Title Fast Partial Least Squares for High-Dimensional Data

Version 0.99.3

Date 2026-07-05

Author Stefano Cacciatore [aut, cre] (<https://orcid.org/0000-0001-7052-7156>), Dupe Ojo [aut] (<https://orcid.org/0000-0002-5301-8592>), Leonardo Tenori [aut] (<https://orcid.org/0000-0001-6438-059X>), Alessia Vignoli [aut] (<https://orcid.org/0000-0003-4038-6596>)

Maintainer Stefano Cacciatore <stefano.cacciatore@icgeb.org>

Description Fast implementations of partial least squares models for high-dimensional regression and classification. The package provides compiled implementations of PLS-SVD, SIMPLS, OPLS and kernel PLS, together with truncated singular value decomposition backends, discriminant classifiers, cross-validation utilities and optional CUDA or Apple Metal acceleration when the required system libraries are available.

License GPL-3

URL https://github.com/tkcaccia/fastPLS

BugReports https://github.com/tkcaccia/fastPLS/issues

biocViews Software, DimensionReduction, Classification, Regression, GeneExpression, Metabolomics, SingleCell, GPU

SystemRequirements Optional CUDA Toolkit and NVIDIA GPU driver for CUDA acceleration; optional Apple Metal framework on macOS for Metal acceleration. CPU-only builds do not require GPU software.

Depends R (>= 4.5.0), Matrix

Imports Rcpp (>= 0.12.17), methods, float

LinkingTo Rcpp, RcppArmadillo, RcppEigen, Matrix

Suggests BiocStyle, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr
NeedsCompilation yes
Config/testthat/edition 3
Encoding UTF-8
Roxygen list(markdown = TRUE)
Config/roxygen2/version 8.0.0
Config/pak/sysreqs nvidia-cuda-dev
Repository https://biocstaging.r-universe.dev
Date/Publication 2026-07-08 06:52:10 UTC
RemoteUrl https://github.com/BiocStaging/fastPLS
RemoteRef HEAD
RemoteSha ef4aa0e4ea663a097fb3c10f6a4ce9f2884a278f

Contents

`breast` *Breast-cancer mRNA subtype example*

Description

Breast-cancer mRNA expression example for supervised multivariate modelling. The data contain tumour samples assigned to three clinically relevant molecular subtypes: basal-like breast cancer, HER2-enriched breast cancer, and luminal A breast cancer. The bundled version keeps one transcriptomic block only (`200` gene-expression variables) and provides a fixed train/test split: `150` training samples and `70` test samples, for `220` samples in total. The class balance is preserved across the split, with `66` Basal, `44` Her2, and `110` LumA samples overall.

Usage

```
data(breast)
```

Format

A list with four elements:

`X_train` Numeric mRNA expression matrix with 150 training tumour samples and 200 gene-expression variables.

`y_train` Training molecular subtype factor with three breast-cancer classes: `"Basal"` (45 samples), `"Her2"` (30 samples), and `"LumA"` (75 samples).

`X_test` Numeric mRNA expression matrix with 70 independent test tumour samples and the same 200 gene-expression variables.

`y_test` Test molecular subtype factor with the same three classes: `"Basal"` (21 samples), `"Her2"` (14 samples), and `"LumA"` (35 samples).

Source

Derived from the `breast.TCGA` dataset distributed in the GPL (>= 2) **mixOmics** package. The data are a filtered TCGA breast-cancer multi-omics example used in the DIABLO framework.

References

Singh A., Shannon C., Gautier B., Rohart F., Vacher M., Tebbutt S. and Le Cao K.A. (2019). DIABLO: an integrative approach for identifying key molecular drivers from multi-omics assays. *Bioinformatics*, 35(17), 3055-3062.

Examples

```
data(breast)
dim(breast$X_train)
table(breast$y_train)

fit <- pls(breast$X_train[, 1:50], breast$y_train,
           breast$X_test[, 1:50], ncomp = 2, method = "simpls",
           backend = "cpu", svd.method = "rsvd",
           return_variance = FALSE)
head(fit$Ypred)
```

`colon` *Colon cancer gene-expression example*

Description

Colon cancer transcriptomic example for binary supervised modelling. The data represent oligonucleotide microarray expression profiles measured from human colon tissue specimens, comparing primary colon tumour samples with normal colon mucosa samples. The bundled matrix contains `62` samples in total: `40` tumour samples and `22` normal samples. Each row is one tissue sample and each column is one gene-expression variable; the bundled version contains `2000` gene-expression variables selected from the original microarray study.

Usage

```
data(colon)
```

Format

A list with three elements:

`X` Numeric oligonucleotide microarray expression matrix with 62 colon tissue samples and 2000 gene-expression variables.

`y` Two-level diagnostic factor with classes `"normal"` (22 samples) and `"tumor"` (40 samples).

`gene_names` Character vector with the 2000 probe/gene identifiers.

Source

Derived from the `Colon` dataset distributed in the GPL (>= 2) **plsgenomics** package. The original data are described in Alon et al. (1999) and were originally collected from the Princeton oncology microarray resource.

References

Alon U., Barkai N., Notterman D.A., Gish K., Ybarra S., Mack D. and Levine A.J. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences USA*, 96(12), 6745-6750.

Examples

```
data(colon)
dim(colon$X)
table(colon$y)

fit <- pls(colon$X[, 1:100], colon$y, ncomp = 1, method = "plssvd",
           backend = "cpu", svd.method = "rsvd",
           return_variance = FALSE)
head(fit$Ypred)
```

`evaluate` *Evaluate prediction performance*

Description

Computes common classification or regression performance metrics from observed and predicted values. The function accepts two vectors, two matrices, or classification score matrices.

Usage

```
evaluate(
  observed,
  predicted,
  task = c("auto", "classification", "regression"),
  ytrain = NULL,
  top_k = c(1L, 5L),
  relative_epsilon = .Machine$double.eps,
  na.rm = TRUE
)
```

Arguments

`observed`	Observed response values. Use a factor or character vector for classification, a numeric vector or matrix for regression, or a one-hot matrix for classification.
`predicted`	Predicted values. Use a factor or character vector for predicted classes, a numeric vector or matrix for regression, or a class-score matrix for classification.
`task`	One of `"auto"`, `"classification"`, or `"regression"`. `"auto"` treats factor or character observed values as classification and numeric values as regression, unless a one-hot observed matrix is detected.
`ytrain`	Optional training response for regression Q2. When supplied, Q2 is computed relative to the training-set response mean. When omitted, Q2 uses the observed response mean and is therefore identical to R2.
`top_k`	Integer vector of top-k classification accuracies to compute when `predicted` is a class-score matrix.
`relative_epsilon`	Values with absolute observed response below this threshold are ignored for relative-error metrics.
`na.rm`	Remove incomplete observations before computing metrics.

Details

For classification, the returned metrics include accuracy, balanced accuracy, macro precision, macro recall, macro F1, Cohen's kappa, and a confusion matrix. When predicted class scores are supplied, top-k accuracy is also reported.

For regression, the returned metrics include R2, Q2, RMSD/RMSE, MAE, bias, median relative error percentage, mean absolute percentage error, ratio of performance to deviation, Pearson correlation, and Spearman correlation. These include the main spectral prediction metrics used by Vignoli et al. (2025): median relative error percentage, RMSE, R2, and RPD.

Value

A list with `task`, `metrics`, and optionally `per_response`, `per_class`, `confusion`, and `topk`. A `notes` element is included only when the evaluation has an explanatory note to report.

Examples

```
evaluate(iris$Species, iris$Species)

set.seed(1)
y <- mtcars$mpg
pred <- y + rnorm(length(y), sd = 2)
evaluate(y, pred)$metrics
```

---

fastcor *Fast Pearson Correlation*

Description

Centers and normalizes rows (or columns when `byrow = FALSE`) and then computes Pearson correlations using fast matrix cross-products. This function does not rank-transform the inputs and therefore does not compute Spearman correlation.

Usage

```
fastcor(a, b = NULL, byrow = TRUE, diag = TRUE)
```

Arguments

`a`	Numeric matrix.
`b`	Optional numeric matrix with the same row or column orientation as `a`.
`byrow`	Logical; when `TRUE`, rows are correlated. When `FALSE`, columns are correlated.
`diag`	Logical; when `b` is supplied and `diag = TRUE`, return only the pairwise diagonal correlations between matching rows (or columns).

Value

A correlation matrix, or a numeric vector of diagonal correlations when `b` is supplied with `diag = TRUE`.

Author(s)

Stefano Cacciatore, Leonardo Tenori, Dupe Ojo, Alessia Vignoli

See Also

pls.single.cv, pls.double.cv

Examples

```
data(iris)
x <- as.matrix(iris[, -5])
fastcor(x)
fastcor(x[1:10, ], x[11:20, ])
```

---

fastsvd *Singular value decomposition through fastPLS backends*

Description

Computes a truncated singular value decomposition of a dense numeric matrix with a selected CPU, CUDA, or Metal backend. The result contains singular values, requested singular vectors, decomposition rank, elapsed time, and backend metadata.

Usage

```
fastsvd(
  x,
  nu = NULL,
  nv = NULL,
  ncomp = NULL,
  backend = c("cpu", "cuda", "metal"),
  method = c("rsvd", "irlba"),
  oversample = 10L,
  power = 1L,
  svds_tol = 0,
  work = 0L,
  maxit = 1000L,
  tol = 1e-05,
  eps = 1e-09,
  svtol = 1e-05,
  seed = 1L
)
```

Arguments

x	Numeric matrix to decompose, with observations or rows in rows and variables or columns in columns. Sparse matrices should be converted by the caller; fastsvd() currently works on a dense numeric matrix.
nu	Number of left singular vectors to return. If NULL, the function uses the largest feasible rank implied by the matrix dimensions. When ncomp is supplied, ncomp controls the decomposition rank and nu controls only how many left vectors are kept in the returned object.

nv	Number of right singular vectors to return. If NULL, the function uses the largest feasible rank implied by the matrix dimensions. When ncomp is supplied, ncomp controls the decomposition rank and nv controls only how many right vectors are kept in the returned object.
ncomp	Optional truncated rank. When supplied, it overrides the rank implied by nu and nv; the final rank is always capped at min(nrow(x), ncol(x)).
backend	Compute backend. cpu runs on the host CPU. cuda dispatches randomized SVD to a CUDA-capable backend. metal dispatches randomized SVD to the Apple Metal backend.
method	SVD algorithm family. irlba uses the bundled iterative IRLBA-style CPU backend and is valid only with backend = cpu. rsvd uses the native fastPLS randomized SVD on the selected backend.
oversample	Non-negative oversampling dimension used by randomized SVD. The sketch dimension is approximately ncomp + oversample, capped by the matrix rank. Larger values can improve approximation accuracy at the cost of extra time and memory.
power	Number of randomized-SVD power iterations. Larger values improve accuracy when singular values decay slowly, but each iteration adds additional matrix multiplications.
svds_tol	Tolerance forwarded to iterative SVD backends. A value of 0 keeps the backend default.
work	IRLBA working subspace size. A value of 0 lets the bundled IRLBA backend choose its default workspace.
maxit	Maximum number of IRLBA iterations before the CPU IRLBA backend stops.
tol	IRLBA residual convergence tolerance. Smaller values can improve numerical convergence but may increase runtime.
eps	IRLBA orthogonality threshold used internally by the bundled implementation.
svtol	IRLBA singular-value convergence tolerance.
seed	Random seed used by randomized backends to generate the Gaussian sketch. It affects rsvd results and is ignored by deterministic backends.

Value

A list compatible with base::svd() containing d, u, and v, plus backend metadata backend, method, svd.method, elapsed, and ncomp.

Examples

```
set.seed(1)
x <- matrix(rnorm(12 * 5), 12, 5)
s <- fastsvd(x, ncomp = 2, backend = "cpu", method = "rsvd", seed = 1)
s$d

s_irlba <- fastsvd(x, ncomp = 2, backend = "cpu", method = "irlba")
s_irlba$svd.method
```

has_cuda *Check CUDA Runtime Availability for fastPLS*

Description

Report whether CUDA-native fastPLS execution can be used in the current session.

Usage

```
has_cuda()
```

Details

`has_cuda()` returns `TRUE` only when both conditions hold:

- the package was built with CUDA support (compile-time macro `FASTPLS_HAS_CUDA`);
- CUDA runtime reports at least one available device.

If either condition is not met, the function returns `FALSE`. The result describes backend availability for the current R session; it does not select a device, allocate GPU memory, or run a test computation.

Value

A length-1 logical.

See Also

`pls`, `fastsvd`, `pca`

Examples

```
has_cuda()
```

has_metal *Check Apple Metal Backend Availability for fastPLS*

Description

Report whether Apple Metal-native fastPLS execution can be used in the current session.

Usage

```
has_metal()
```

Details

has_metal() returns TRUE only when both conditions hold:

- the package was built with Apple Metal support;
- the Metal backend is available in the current session.

If either condition is not met, the function returns FALSE. This includes non-macOS builds and portable builds that use the Metal stub backend. The result describes backend availability for the current R session; it does not select a device, allocate GPU memory, or run a test computation.

Value

A length-1 logical.

See Also

has_cuda, fastsvd, pca, pls

Examples

```
has_metal()
```

---

pca *Principal component analysis through fastPLS SVD backends*

Description

Computes PCA from the selected fastPLS SVD backend and returns a compact object with a score-plot method.

Usage

```
pca(
  x,
  ncomp = 2L,
  xtest = NULL,
  center = TRUE,
  scale = FALSE,
  backend = c("cpu", "cuda", "metal"),
  method = c("rsvd", "irlba"),
  ...
)
```

Arguments

x	Numeric matrix with samples in rows and variables in columns.
ncomp	Number of principal components.
xtest	Optional independent matrix to project using the PCA loadings learned from x. The same centering and scaling estimated from x are applied to xtest.
center	Logical; center columns before SVD.
scale	Logical; scale columns before SVD.
backend	Compute backend. cpu runs on the host CPU. cuda and metal use the corresponding native randomized-SVD backend when available.
method	SVD algorithm family. irlba is available only with backend = cpu. rsvd uses native fastPLS randomized SVD on the selected backend.
...	Additional arguments passed to fastsvd.

Value

A fastPLSPCA object containing training scores, optional scores_test, loadings, standard deviations, per-component variance_explained, cumulative variance explained, preprocessing values, and SVD metadata.

Examples

```
pc <- pca(as.matrix(iris[, 1:4]), ncomp = 2, backend = "cpu",
          method = "rsvd", seed = 1)
head(pc$scores)
pc$variance_explained
head(predict(pc, as.matrix(iris[1:5, 1:4])))
```

---

plot.fastPLSPCA *Plot PCA or PLS scores*

Description

Draws a two-component PCA or PLS score plot. Ellipses can be drawn globally or per group, using either a standard data confidence ellipse or a Hotelling's T2 score ellipse. Grouped points use filled plotting symbols by default, with the group color assigned to bg and a black point contour assigned to col. PCA plots use pch = 22 unless another plotting character is supplied through .... Axis labels include the predictor-space variance explained by each plotted PCA component or PLS latent variable when available.

Usage

```
## S3 method for class 'fastPLSPCA'
plot(
  x,
  comps = c(1L, 2L),
  groups = NULL,
  ellipse = FALSE,
  ellipse.type = c("confidence", "hotelling"),
  conf = 0.95,
  ...
)

## S3 method for class 'fastPLS'
plot(
  x,
  comps = c(1L, 2L),
  groups = NULL,
  score.set = c("auto", "train", "test"),
  ellipse = FALSE,
  ellipse.type = c("confidence", "hotelling"),
  conf = 0.95,
  ...
)
```

Arguments

x	A fastPLSPCA or fastPLS object.
comps	Two component indices to plot.
groups	Optional grouping vector for color and grouped ellipses.
score.set	For PLS objects, plot train scores, test scores, or auto to use training scores when available.
ellipse	Logical; draw confidence ellipses when TRUE.
ellipse.type	Either confidence for a covariance confidence ellipse or hotelling for a Hotelling's T2 ellipse.
conf	Confidence level.
...	Additional arguments passed to plot().

Value

Invisibly returns the plotted score matrix.

Examples

```
pc <- pca(as.matrix(iris[, 1:4]), ncomp = 2, backend = "cpu",
          method = "rsvd", seed = 1)
plot(pc, groups = iris$Species, ellipse = TRUE)
```

`plot.permutation` *Plot PLS permutation-test R2 and Q2 values*

Description

Draws the permutation-test diagnostic plot produced by `pls(..., perm.test = TRUE)`. The x-axis is the correlation between the original and permuted response structure; the y-axis is the observed or permuted R2/Q2 value. R2 is shown in blue and Q2 in red.

Usage

```
## S3 method for class 'permutation'
plot(
  x,
  ncomp = NULL,
  main = NULL,
  xlab = "Cor",
  ylab = "Value",
  col = c(R2 = "#3155B7", Q2 = "#E5332A"),
  pch = c(R2 = 16, Q2 = 15),
  legend_position = "bottomright",
  ...
)
```

Arguments

`x`	A `fastPLS` model fitted with `perm.test = TRUE`, or a permutation data frame stored in `model$permutation`.
`ncomp`	Component count to plot. Defaults to the largest component stored in the permutation table.
`main`, `xlab`, `ylab`	Plot title and axis labels.
`col`, `pch`	Colors and point symbols for R2 and Q2.
`legend_position`	Legend position passed to `legend()`.
`...`	Additional graphical parameters passed to `plot()`.

Value

Invisibly returns the plotted permutation data.

Examples

```
set.seed(1)
X <- as.matrix(iris[, 1:4])
y <- iris$Sepal.Length
idx <- sample(seq_len(nrow(X)), 30)
```

```
fit <- pls(X[idx, ], y[idx], X[idx, ], y[idx],
           ncomp = 2, perm.test = TRUE, times = 5)
plot.permutation(fit)
```

---

pls *Partial Least Squares with selectable model family and backend*

Description

Fits PLSSVD, SIMPLS, OPLS, or kernel PLS models for regression or classification using a selected CPU, CUDA, or Metal backend. The fitted model can include predictions for held-out samples, latent scores, fitted values, variance summaries, and optional classification heads.

Usage

```
pls(
  Xtrain,
  Ytrain,
  Xtest = NULL,
  Ytest = NULL,
  ncomp = 2,
  scaling = c("centering", "autoscaling", "none"),
  method = c("simpls", "plssvd", "opls", "kernelpls"),
  svd.method = c("rsvd", "irlba"),
  classifier = c("argmax", "lda", "cknn"),
  lda_ridge = 1e-08,
  k = 10L,
  tau = 0.2,
  alpha = 0.75,
  top_m = 20L,
  cknn_memory = c("auto", "standard", "blocked", "streaming"),
  fit = FALSE,
  return_variance = TRUE,
  return_loadings = FALSE,
  proj = FALSE,
  perm.test = FALSE,
  times = 100,
  backend = c("cpu", "cuda", "metal"),
  north = 1L,
  kernel = c("linear", "rbf", "poly"),
  gamma = NULL,
  degree = 3L,
  coef0 = 1,
  ...
)
```

Arguments

Xtrain	Numeric training predictor matrix, or a float::float32 predictor matrix for the supported float32 route.
Ytrain	Training response (numeric or factor).
Xtest	Optional test predictor matrix.
Ytest	Optional test response for Q2Y.
ncomp	Number of components (scalar or vector).
scaling	One of centering, autoscaling, or none.
method	One of simpls, plssvd, opls, or kernelpls. simpls uses the fastPLS accelerated SIMPLS core.
svd.method	SVD algorithm family. rsvd uses the native fastPLS randomized SVD for the selected backend, and irlba uses the bundled CPU iterative backend.
classifier	Classification decision rule. argmax keeps the standard PLS-DA response-score argmax. lda fits an LDA classifier on the PLS latent scores. cknn is the compact name for the PLS-score candidate-kNN classifier: class centroids in PLS score space choose candidate classes, then every sample is reranked by within-candidate kNN in the same PLS score space. The compiled implementation is selected automatically from backend: C++ for cpu, CUDA for cuda, and Metal for metal where available.
lda_ridge	Relative diagonal ridge added to the pooled LDA covariance.
k	Number of same-class PLS-score neighbours used by the candidate-kNN classifier. Larger values use more neighbours per candidate class and therefore give a smoother, less local decision.
tau	Positive temperature for pooling the top neighbour similarities in candidate-kNN scoring. Smaller values make the score close to the best neighbour; larger values average the top neighbours more smoothly.
alpha	Weight of the centroid/prototype candidate score added to the local kNN score. The final candidate score is the smoothed same-class kNN evidence plus alpha times the centroid score. Higher values keep the decision closer to the global PLS class-centroid ranking; lower values make the reranking depend more strongly on local neighbours.
top_m	Number of centroid-ranked candidate classes passed to the kNN reranker.
cknn_memory	Memory strategy for classifier = "cknn". standard uses the historical one-pass candidate-kNN path. blocked predicts test samples in blocks, reducing test-side latent-score memory. streaming additionally builds the training candidate-score cache in blocks for scalar component counts. auto chooses a memory-aware strategy from the data size.
fit	Return fitted values and R2Y when TRUE.
return_variance	Compute predictor-space latent-variable variance explained. Set to FALSE for timing/memory benchmarks that do not need plotting variance metadata.
return_loadings	Compute and store predictor loadings P. The default is FALSE because most prediction workflows only need the projection weights and response-side coefficients.

proj	Return projected Ttest when TRUE.
perm.test	Run a single-split permutation test when Xtest and Ytest are supplied. The rows of Xtrain are randomly permuted, the model is refitted, and the permuted test-set Q2Y path is compared with the observed Q2Y path.
times	Number of permutations. For pls(), the empirical p-value for each component is mean(Q2Y_permuted > Q2Y_observed) and is returned in pval. No +1 correction is applied.
backend	Implementation backend: cpu for compiled CPU, cuda for CUDA-native fitting, or experimental metal for Apple Metal randomized-SVD/GEMM acceleration.
north	Number of orthogonal components removed by OPLS.
kernel	Kernel type for kernel PLS: linear, rbf, or poly.
gamma	Kernel scale. Defaults internally to 1 / ncol(Xtrain).
degree	Polynomial kernel degree.
coef0	Polynomial kernel offset.
...	Optional SVD tuning controls forwarded to the selected backend. Use the same compact names documented in fastsvd(), such as oversample, power, svds_tol, work, maxit, tol, eps, svtol, and seed.

Details

If Xtrain, Xtest, Ytrain, or Ytest are supplied as float::float32 objects, pls() uses a float32 route instead of converting them to double. In the current implementation this route is available for method = "plssvd" or method = "simpls". CPU and Metal support svd.method = "rsvd" or svd.method = "irlba"; CUDA supports float32 randomized SVD. Classification with float32 input supports classifier = "argmax", classifier = "lda", and classifier = "cknn". Unsupported float32 combinations stop with a clear error rather than silently upcasting to double.

Value

A fastPLS object. The object is a list whose fields depend on the selected method, backend, classifier, and whether test data or optional summaries were requested. Common fields are:

- P: predictor loadings, with one column per latent component.
- Q: response loadings or response-side latent coefficients.
- R: predictor weights/rotations used to project new samples into the PLS latent space.
- Ttrain: training latent scores. This is returned when the backend stores scores or when they are needed for fitted values, classifiers, variance summaries, or compact prediction.
- C_latent, W_latent: low-rank latent prediction factors used by PLSSVD-style compact prediction when a full coefficient array is avoided.
- B: regression coefficient matrix or coefficient array, when stored. For vector-valued ncomp, a three-dimensional array may contain the coefficient path for all requested component counts.
- mX, vX: training predictor centering and scaling values. vX is one when no scaling is applied.
- mY: response centering values for regression or dummy-coded PLS-DA.
- lev: factor levels used for classification.

- `Yfit`: fitted training responses or fitted class labels, returned when `fit = TRUE`.
- `R2Y`: training-set coefficient of determination path when `fit = TRUE`; otherwise `NA` placeholders may be returned for compatibility. Elements are named by component count, for example `"ncomp=2"`.
- `Ypred`: predictions for `Xtest`, returned only when `Xtest` is supplied to `pls()`. For classification this contains predicted factor labels; for regression it contains numeric predictions.
- `Ypred_index`: integer class indices for classification predictions, when available.
- `Ttest`: test-set latent scores, returned when `proj = TRUE`.
- `Q2Y`: test-set Q2 for numeric `Ytest`, or dummy-response PLS-DA Q2 for factor `Ytest`, returned when response scores are available. Elements are named by component count.
- `accuracy`: decoded-label accuracy for factor `Ytest`, returned when classification predictions are available. Elements are named by component count.
- `pval`: single-split permutation-test p-values by component, returned when `perm.test = TRUE`. Each p-value is the fraction of permuted `Q2Y` values larger than the observed `Q2Y`.
- `permutation`: long-format permutation table, returned when `perm.test = TRUE`, with observed and permuted `R2`/`Q2` values and the permutation correlation used by `plot.permutation()`.
- `variance`, `variance_explained`, `cumulative_variance_explained`, `variance_total`, `variance_basis`: predictor-space variance summaries returned when `return_variance = TRUE`.
- `x_variance`, `x_variance_explained`, `x_cumulative_variance_explained`, `x_variance_total`: aliases of the predictor-space variance summaries.
- `inner_model`: fitted inner PLS model used by OPLS.
- `W_orth`, `P_orth`, `north`, `opls_engine`, `xprod_mode`, `gpu_resident`: OPLS-specific orthogonal-component and backend metadata.
- `kernel`, `kernel_engine`, `kernel_linear_direct`: kernelPLS-specific kernel settings and execution metadata.

Function settings and backend bookkeeping, such as the component grid and resolved classifier backend, are retained internally for prediction and plotting but are not shown as public output fields.

Examples

```
X <- as.matrix(mtcars[, c("disp", "hp", "wt", "qsec")])
y <- mtcars$mpg
fit <- pls(X, y, ncomp = 2, method = "simpls", backend = "cpu",
           svd.method = "rsvd", return_variance = FALSE)
head(predict(fit, X)$Ypred)

cv <- pls.single.cv(X, y, ncomp = 1:2, kfold = 3, method = "simpls",
                    backend = "cpu", svd.method = "rsvd", seed = 1)
fit_cv <- pls(cv, Xtest = X, return_variance = FALSE)
cv$best_ncomp
head(fit_cv$Ypred)
```

`pls.double.cv` *Nested cross-validation for PLS*

Description

Performs nested grouped cross-validation with an outer loop for unbiased performance estimation and an inner loop for component and hyperparameter selection. Constraint groups are respected in both loops so related samples remain in the same fold.

Usage

```
pls.double.cv(
  Xdata,
  Ydata,
  ncomp = 2,
  constrain = 1:nrow(Xdata),
  scaling = c("centering", "autoscaling", "none"),
  method = c("simpls", "plssvd", "opls", "kernelpls"),
  backend = c("cpu", "cuda", "metal"),
  svd.method = c("irlba", "rsvd"),
  seed = 1L,
  perm.test = FALSE,
  times = 100,
  runn = 1,
  kfold_inner = 10,
  kfold_outer = 10,
  north = 1L,
  kernel = c("linear", "rbf", "poly"),
  gamma = NULL,
  degree = 3L,
  coef0 = 1,
  classifier = c("argmax", "lda", "cknn"),
  lda_ridge = 1e-08,
  k = 10L,
  tau = 0.2,
  alpha = 0.75,
  top_m = 20L,
  cknn_memory = c("auto", "standard", "blocked", "streaming"),
  xprod = NULL,
  ...
)
```

Arguments

`Xdata`	Predictor matrix.
`Ydata`	Response (numeric or factor).

ncomp	Number of components (scalar or vector).
constrain	Grouping vector for grouped cross-validation. It must have one value per sample. Samples with the same value are assigned to the same fold, so all rows from the same patient, subject, batch, or technical replicate stay together in training or test data. The default `1:nrow(Xdata)` treats every sample as an independent group.
scaling	One of `centering`, `autoscaling`, or `none`.
method	One or more of `simpls`, `plssvd`, `opls`, or `kernelpls`. Multiple values are tuned in the inner loop.
backend	Implementation backend: `cpu`, `cuda`, or `metal`. Multiple values are tuned in the inner loop.
svd.method	SVD algorithm family. `rsvd` uses the native fastPLS randomized SVD for the selected backend, and `irlba` uses the bundled CPU iterative backend.
seed	Random seed used for outer/inner fold assignment and randomized SVD steps.
perm.test	Run a nested-CV permutation test. For each permutation, the rows of `Xdata` are shuffled, the complete double cross-validation is repeated, and the median permuted `Q2Y` is compared with the observed median `Q2Y`.
times	Number of permutations. For predictive metrics where larger is better, such as `Q2Y`, the empirical p-value is `mean(Q2Y_permuted >= Q2Y_observed)`. For loss metrics where smaller is better, such as RMSD, it is `mean(loss_permuted <= loss_observed)`. No +1 correction is applied.
runn	Number of repeated runs.
kfold_inner	Inner-fold count, or `"loocv"` to leave out one constraint group at a time inside each outer training set.
kfold_outer	Outer-fold count, or `"loocv"` to leave out one constraint group at a time in the outer loop. In both loops, samples sharing the same constraint value are never split across training and test.
north	Number of orthogonal components removed by OPLS.
kernel	Kernel type for kernel PLS: `linear`, `rbf`, or `poly`.
gamma	Kernel scale. Defaults internally to `1 / ncol(Xdata)`. For `method = "kernelpls"`, multiple values are tuned in the inner loop.
degree	Polynomial kernel degree.
coef0	Polynomial kernel offset.
classifier	Classification decision rule. `argmax` keeps the standard PLS-DA response-score argmax. `lda` fits an LDA classifier on the PLS latent scores. `cknn` is the compact name for the PLS-score candidate-kNN classifier: class centroids in PLS score space choose candidate classes, then every sample is reranked by within-candidate kNN in the same PLS score space. The compiled implementation is selected automatically from `backend`: C++ for `cpu`, CUDA for `cuda`, and Metal for `metal` where available.
lda_ridge	Relative diagonal ridge added to the pooled LDA covariance.
k	Number of same-class PLS-score neighbours used by the candidate-kNN classifier. Larger values use more neighbours per candidate class and therefore give a smoother, less local decision.

tau	Positive temperature for pooling the top neighbour similarities in candidate-kNN scoring. Smaller values make the score close to the best neighbour; larger values average the top neighbours more smoothly.
alpha	Weight of the centroid/prototype candidate score added to the local kNN score. The final candidate score is the smoothed same-class kNN evidence plus alpha times the centroid score. Higher values keep the decision closer to the global PLS class-centroid ranking; lower values make the reranking depend more strongly on local neighbours.
top_m	Number of centroid-ranked candidate classes passed to the kNN reranker.
cknn_memory	Memory strategy for classifier = "cknn". standard uses the historical one-pass candidate-kNN path. blocked predicts test samples in blocks, reducing test-side latent-score memory. streaming additionally builds the training candidate-score cache in blocks for scalar component counts. auto chooses a memory-aware strategy from the data size.
xprod	Use the matrix-free cross-product route where available for inner component optimization. NULL applies fastPLS defaults.
...	Optional SVD tuning controls forwarded to the selected backend. Use the same compact names documented in fastsvd(), such as oversample, power, svds_tol, work, maxit, tol, eps, and svtol. Vector values are tuned in the inner loop. Inner selection can also be controlled with selection_metric = "auto", "accuracy", "r2", "q2", or "rmsd"; the selection metric itself is scalar.

Value

A list with the following elements:

- results: list with one element per repeated run. Each run stores Ypred/pred, the outer fold assignment, best_ncomp selected in each outer fold, fold-level best_parameters, the complete inner-CV objects in inner, run-level metric_name and metric_value, and the default backend and method.
- Ypred: final cross-validated predictions. For classification, repeated runs are combined by voting; for regression, numeric predictions are averaged across runs.
- Q2Y: one held-out Q2 value per repeated run. For factor responses this is calculated on dummy-coded PLS-DA response scores.
- R2Y: one training-fit R2 value per repeated run, averaged across the selected outer-fold models.
- RMSD: one held-out RMSD value per repeated run for numeric responses; NA for classification.
- metric_name: held-out metric used for each repeated run.
- bcomp: most frequently selected component count across outer folds and repeated runs.
- backend, method: default backend and PLS method supplied to the call. If vector-valued methods or backends are tuned, selected fold-level values are stored in results[[run]]$best_parameters.
- selection_metric: criterion used by the inner CV loop.
- acc_tot: classification-only text summary of correctly classified samples and percentage accuracy.
- conf: classification-only confusion matrix printed as counts and column percentages.

- vote_counts: classification-only vote-count matrix with one row per sample and one column per class.
- accuracy: classification-only decoded-label accuracy, one value per repeated run.
- medianR2Y, CI95R2Y, medianQ2Y, CI95Q2Y, medianRMSD, CI95RMSD: repeated-run summaries returned only when runn > 1.
- Q2Ysampled: permutation median-Q2 values returned when perm.test = TRUE.
- p.value: permutation-test p-value returned when perm.test = TRUE.

Examples

```
idx <- c(1:10, 51:60, 101:110)
X <- as.matrix(iris[idx, 1:4])
y <- factor(iris[idx, 5])
dcv <- pls.double.cv(X, y, ncomp = 1:2, runn = 1, kfold_inner = 2,
                     kfold_outer = 2, method = "simpls", backend = "cpu",
                     svd.method = "rsvd", seed = 1)
names(dcv)
```

---

pls.single.cv *Single cross-validation for PLS component optimization*

Description

Performs grouped k-fold or leave-one-out cross-validation over candidate component counts and, when vector-valued predictive arguments are supplied, over a compact hyperparameter grid. The selection can be based on cross-validated accuracy, R2, Q2, or RMSD.

Usage

```
pls.single.cv(
  Xdata,
  Ydata,
  ncomp = 2,
  constrain = NULL,
  scaling = c("centering", "autoscaling", "none"),
  method = c("simpls", "plssvd", "opls", "kernelpls"),
  backend = c("cpu", "cuda", "metal"),
  svd.method = c("irlba", "rsvd"),
  seed = 1L,
  kfold = 10,
  north = 1L,
  kernel = c("linear", "rbf", "poly"),
  gamma = NULL,
  degree = 3L,
  coef0 = 1,
  classifier = c("argmax", "lda", "cknn"),
```

```
  lda_ridge = 1e-08,
  k = 10L,
  tau = 0.2,
  alpha = 0.75,
  top_m = 20L,
  cknn_memory = c("auto", "standard", "blocked", "streaming"),
  fit = TRUE,
  xprod = NULL,
  ...
)
```

Arguments

Xdata	Predictor matrix.
Ydata	Response (numeric or factor).
ncomp	Number of components (scalar or vector).
constrain	Optional grouping vector for grouped cross-validation. It must have one value per sample. Samples with the same value are assigned to the same fold, so all rows from the same patient, subject, batch, or technical replicate stay together in training or test data. When NULL, each sample is treated as its own group.
scaling	One of centering, autoscaling, or none.
method	One or more of simpls, plssvd, opls, or kernelpls. Multiple values are treated as a tuning grid.
backend	Implementation backend: cpu, cuda, or metal. Multiple values are treated as a tuning grid.
svd.method	SVD algorithm family. rsvd uses the native fastPLS randomized SVD for the selected backend, and irlba uses the bundled CPU iterative backend.
seed	Random seed used for fold assignment and randomized SVD steps.
kfold	Number of folds, or "loocv" for leave-one-out cross-validation. When constrain is supplied, LOOCV means leave-one-constraint-group-out: samples sharing the same constraint value are always held out together and are never split across training and test.
north	Number of orthogonal components removed by OPLS.
kernel	Kernel type for kernel PLS: linear, rbf, or poly.
gamma	Kernel scale. Defaults internally to 1 / ncol(Xdata). For method = "kernelpls", multiple values are treated as a tuning grid.
degree	Polynomial kernel degree.
coef0	Polynomial kernel offset.
classifier	Classification rule for factor responses: "argmax", "lda", or "cknn". Multiple values are treated as a tuning grid.
lda_ridge	Ridge added to the pooled LDA covariance diagonal. Multiple values are used only when classifier = "lda".
k	Number of same-class PLS-score neighbours used by candidate-kNN when classifier = "cknn".

tau	Positive temperature for smoothing the top neighbour similarities. Smaller values emphasize the nearest neighbour; larger values produce a smoother local class score.
alpha	Weight of the centroid/prototype candidate score in candidate-kNN. The final candidate score is local kNN evidence plus alpha times the centroid score, so larger values keep more of the global centroid ranking.
top_m	Number of centroid-ranked candidate classes passed to the kNN reranker.
cknn_memory	Memory strategy for classifier = "cknn". standard uses the historical one-pass candidate-kNN path. blocked predicts test samples in blocks, reducing test-side latent-score memory. streaming additionally builds the training candidate-score cache in blocks for scalar component counts. auto chooses a memory-aware strategy from the data size.
fit	Fit one additional model on the full dataset and return its fitted values (Yfit) and training R2Y path. The default is TRUE for backward compatibility. Set to FALSE to skip this extra full-data fit; held-out cross-validated Q2Y and RMSD are still calculated.
xprod	Use the matrix-free cross-product route where available. NULL applies fastPLS defaults.
...	Optional SVD tuning controls forwarded to the selected backend. Use the same compact names documented in fastsvd(), such as oversample, power, svds_tol, work, maxit, tol, eps, and svtol. Vector values are included in the tuning grid. Component selection can also be controlled with selection_metric = "auto", "accuracy", "r2", "q2", or "rmsd"; the selection metric itself is scalar.

Value

A list describing the cross-validation run and selected model:

- best_ncomp: number of components selected by the chosen metric.
- best_index: position of best_ncomp in the tested component grid.
- selection_metric: metric used for optimization. With "auto", classification uses accuracy and regression uses the default prediction error rule.
- best_metric_name and best_metric_value: name and value of the metric at the selected component count.
- Q2Y: held-out cross-validated Q2. For factor responses, Q2 is calculated on the dummy-coded PLS-DA response scores.
- accuracy: held-out decoded-label accuracy for factor responses.
- RMSD: held-out root mean squared deviation for regression. It is NA for classification.
- Yfit: fitted values from the full-data model when fit = TRUE.
- R2Y: training-set explained-variance path from a model fitted on the full dataset when fit = TRUE; otherwise NA. For factor responses, this is calculated on the dummy-coded PLS-DA response scores, not on the decoded class labels.
- fold: fold assignment used for each sample.
- pred: decoded cross-validated predictions when predictions are stored.

- `Ypred`: raw prediction array when score predictions are stored.
- `metrics`: per-component metric table returned by the CV backend.
- `best_parameters`: compact list containing only `ncomp` plus the arguments that were actually optimized, for example `classifier` when `classifier = c("argmax", "lda")`.
- `tuning_config`: relevant selected configuration used for the run. Irrelevant classifier- or method-specific defaults are omitted; for example, cKNN controls are not shown when `classifier = "argmax"`.
- `tuning_summary` and `tuning_metrics`: tables for all tested configurations when more than one predictive configuration is supplied.
- The returned object can be passed as the first argument to `pls()` to refit the selected model on the full training data and predict new samples.

Examples

```
idx <- c(1:12, 51:62, 101:112)
X <- as.matrix(iris[idx, 1:4])
y <- factor(iris[idx, 5])
opt <- pls.single.cv(X, y, ncomp = 1:2, kfold = 3, method = "simpls",
                     backend = "cpu", svd.method = "rsvd", seed = 1)
opt$best_ncomp
opt_kernel <- pls.single.cv(X, y, ncomp = 1:2, kfold = 3,
                            method = "kernelpls", backend = "cpu",
                            svd.method = "rsvd",
                            kernel = c("linear", "rbf"),
                            gamma = c(0.1, 1), seed = 1)
opt_kernel$best_parameters
```

---

predict.fastPLS *Predict from fitted fastPLS models*

Description

Generates predictions for new samples from fitted PLSSVD, SIMPLS, OPLS, or kernel PLS models. Stored centering, scaling, latent projections, and model-specific filtering are applied before producing numeric response predictions or classification labels.

Usage

```
## S3 method for class 'fastPLS'
predict(
  object,
  newdata,
  Ytest = NULL,
  proj = FALSE,
  backend = c("auto", "cpu", "cpu_flash", "cuda_flash", "metal"),
  flash.block_size = NULL,
  top = 1L,
```

```
  top5 = FALSE,
  raw_scores = FALSE,
  ...
)

## S3 method for class 'fastPLSKernel'
predict(object, newdata, Ytest = NULL, proj = FALSE, ...)

## S3 method for class 'fastPLSOpls'
predict(object, newdata, Ytest = NULL, proj = FALSE, ...)
```

Arguments

object	A fitted fastPLS, fastPLSKernel, or fastPLSOpls object.
newdata	Numeric predictor matrix.
Ytest	Optional observed response used to compute Q2Y.
proj	Logical; return projected Ttest when TRUE.
backend	Prediction backend. auto uses FlashSVD-style low-rank prediction when compact factors are available and the low-rank application is expected to be beneficial.
flash.block_size	Row block size for cpu_flash prediction.
top	Number of ranked classes to return for classification. Rank 1 is the ordinary predicted class stored in Ypred; ranks 2, 3, and so on are lower-scoring alternatives returned in Ypred_top when top > 1.
top5	Convenience flag equivalent to top = max(top, 5), useful for reporting ImageNet-style top-5 candidate labels.
raw_scores	If TRUE, keep raw classification score cubes as Yscore when available.
...	Unused.

Value

A list containing Ypred, optional Q2Y, optional Ttest, and optional LDA scores for LDA classification models.

Examples

```
X <- as.matrix(mtcars[, c("disp", "hp", "wt", "qsec")])
y <- mtcars$mpg
fit <- pls(X, y, ncomp = 2, method = "simpls", backend = "cpu",
           svd.method = "rsvd", return_variance = FALSE)
pred <- predict(fit, X[1:3, , drop = FALSE])
pred$Ypred
```

predict.fastPLSPCA *Project new data with a fitted fastPLS PCA model*

Description

Applies the centering, scaling, and loading matrix stored in a `fastPLSPCA` object to an independent dataset.

Usage

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

Arguments

`object`	A `fastPLSPCA` object returned by `pca`.
`newdata`	Numeric matrix with the same columns, in the same order, as the matrix used to fit `object`.
`ncomp`	Optional number of principal components to return. By default all components stored in `object` are returned.
`...`	Ignored.

Value

Matrix of projected PCA scores for `newdata`.

Examples

```
pc <- pca(as.matrix(iris[, 1:4]), ncomp = 2, backend = "cpu",
          method = "rsvd", seed = 1)
predict(pc, as.matrix(iris[1:3, 1:4]))
```

ViP *Variable importance in projection (VIP)*

Description

Computes VIP trajectories from fitted model components.

Usage

```
ViP(model)
```

Arguments

`model`	Fitted `fastPLS` model.

Value

Numeric matrix (single response) or list of matrices (multi-response).

Examples

```
X <- as.matrix(mtcars[, c("disp", "hp", "wt", "qsec")])
y <- mtcars$mpg
fit <- pls(X, y, ncomp = 1, method = "plssvd", backend = "cpu",
           svd.method = "rsvd", return_variance = FALSE)
ViP(fit)
```

Index