Package: lwpls (via r-universe)

October 18, 2026

Type Package

Title Locally-Weighted Partial Least Squares for 'tidymodels'

Version 0.3.0

Description Locally-weighted partial least squares (LW-PLS) is a just-in-time learning method that builds a dedicated weighted partial least squares model around every new sample, where training samples are weighted by their similarity to that sample (Kim et al. (2011) <doi:10.1016/j.ijpharm.2011.10.007>). Regression (single or multiple outcomes) and classification (locally weighted partial least squares-discriminant analysis; Bevilacqua and Marini (2014) <doi:10.1016/j.aca.2014.05.057>) are supported. The local models can be robust to outliers, by partial robust M-regression (Serneels et al. (2005) <doi:10.1016/j.chemolab.2005.04.007>), and sparse (Hoffmann et al. (2015) <doi:10.1016/j.chemolab.2015.09.019>). The prediction kernel is written in C++ with 'RcppArmadillo', and the model is integrated with the 'parsnip', 'dials', 'tune' and 'yardstick' packages so it can be used within the 'tidymodels' framework. Diagnostics of the local models (prediction reliability, local regression vectors, robust weights and selected predictors) come with 'ggplot2' plots designed for spectroscopic data.

License MIT + file LICENSE

URL https://github.com/ChristianGoueguel/lwpls, https://christiangoueguel.com/lwpls/

BugReports https://github.com/ChristianGoueguel/lwpls/issues

Depends R (>= 4.1.0)

Imports cli, dials (>= 1.2.0), generics, ggplot2 (>= 3.5.0), hardhat (>= 1.3.0), parsnip (>= 1.3.0), Rcpp, rlang (>= 1.1.0), scales (>= 1.3.0), stats, tibble, vctrs, yardstick (>= 1.2.0)

Suggests dplyr, knitr, modeldata, plotly, pls, recipes, rmarkdown, rsample, spelling, testthat (>= 3.0.0), tune (>= 1.1.0), withr, workflows

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Christian L. Goueguel [aut, cre], Hiromasa Kaneko [cph]
(reference LW-PLS implementation transcribed in the tests)

Maintainer Christian L. Goueguel <christian.goueguel@gmail.com>

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

Date/Publication 2026-10-18 18:13:19 UTC

RemoteUrl https://github.com/cran/lwpls

RemoteRef HEAD

RemoteSha cded278d38e5411b3d2d33ca7d393e46aae4dd34

Contents

localization 2
lwpls 3
lwpls_fit 7
lwpls_local 10
lwpls_plots 13
multi_predict._lwpls_fit 15
predict.lwpls_fit 16
similarity 17
sparsity 18
trimmed_rmse 19
update.lwpls 20

Index **22**

`localization` *Localization parameter for LW-PLS*

Description

The localization parameter controls how fast the similarity weights of `lwpls()` decrease with the distance to the query sample. Small values give very local models; large values give all training samples similar weights, so that the model tends to a global PLS model.

Usage

```
localization(range = c(-9, 5), trans = scales::transform_log2())
```

Arguments

range	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

The default range, 2^{-9} to 2^{5} on a log-2 scale, follows the candidate values used by Kaneko for the LW-PLS hyperparameter search.

Value

A quant_param object (see dials::new_quant_param()).

Examples

```
localization()
dials::value_seq(localization(), 5)
dials::grid_regular(dials::num_comp(c(1, 5)), localization(), levels = 3)
```

---

lwpls *Locally-weighted partial least squares*

Description

lwpls() defines a locally-weighted partial least squares (LW-PLS) model for use with the parsnip package. LW-PLS is a *just-in-time* (lazy) learner: instead of fitting one global model, a dedicated weighted PLS model is built around every new sample, where each training sample is weighted by its similarity to that sample. This lets a linear method capture nonlinear and time-varying relationships, which makes it popular for soft sensors and spectroscopic calibration.

The model can be used for regression (one or more numeric outcomes) and classification (discriminant analysis on class indicators). The local models can also be sparse, so that each one selects its own predictors, and robust to outliers in the training data.

There is a single engine, "lwpls", implemented in this package (C++ via RcppArmadillo).

Usage

```
lwpls(
  mode = "unknown",
  num_comp = NULL,
  localization = NULL,
  neighbors = NULL,
  similarity = NULL,
```

```
  sparsity = NULL,
  engine = "lwpls"
)
```

Arguments

`mode`	A single character string for the prediction outcome mode. Possible values are `"unknown"`, `"regression"`, or `"classification"`.
`num_comp`	The number of PLS components (latent variables) in each local model (engine default: 2).
`localization`	A positive number controlling how local the models are (engine default: 1). Small values give large weights to the closest training samples only; large values give all samples similar weights, so that LW-PLS tends to a global PLS model. See Details.
`neighbors`	The number of nearest training samples used for each local model. The default (`NULL`) uses all training samples, as in the original LW-PLS algorithm.
`similarity`	The similarity index: `"euclidean"` (engine default) or `"covariance"` for covariance-based LW-PLS (CbLW-PLS). See Details and `similarity()`.
`sparsity`	A number in `[0, 1)`: the sparsity threshold η of the local models (engine default: 0, no sparsity). See Details and `sparsity()`.
`engine`	A single character string specifying the computational engine. Only `"lwpls"` is available.

Details

For a query sample x_q, the similarity weight of the training sample x_i is

$$\omega_i = \exp\left(-\frac{d_i}{\sigma_d\,\varphi}\right),$$

where $d_i = \|x_i - x_q\|$ is the Euclidean distance (on standardized predictors by default), σ_d is the standard deviation of the distances and φ is the `localization` parameter (λ in Kaneko's implementation). A weighted PLS model is then fitted with weighted centering, weighted covariances and the query is projected onto its local latent space to obtain the prediction.

With `similarity = "covariance"`, the distances are computed after projecting the samples on the covariance direction $\Gamma = X^\top Y / \|X^\top Y\|$ of the training data, $d_i = \|\Gamma^\top (x_i - x_q)\|$. This covariance-based LW-PLS (CbLW-PLS; Hazama and Kano, 2015) accounts for the relationships among the predictors and between the predictors and the outcome(s). For a single outcome, Γ is the first PLS weight vector; for several outcomes (or classes), the distance combines the covariance directions of all of them. Hazama and Kano write the weights as $\exp(-\phi d_i/\sigma_d)$, so their ϕ is `1 / localization`.

Value

A model specification object with classes `lwpls` and `model_spec`.

Sparse local models

With `sparsity` $\eta > 0$, the local models are fitted by sparse NIPALS (SNIPLS; Hoffmann et al., 2015): the weight vector of each component is soft-thresholded at $\eta \max_j |w_j|$, so that the predictors with small weights are dropped, and the X loadings are zero outside the predictors selected so far. Each local model therefore selects its own predictors, such as the wavelengths relevant around the query. For several outcomes or classes, the weight vector is the dominant direction of the covariance with all of them before thresholding.

Robust local models

With the engine argument `robust = TRUE`, each local model is fitted by partial robust M-regression (PRM; Serneels et al., 2005), which makes it robust to outlying reference values (vertical outliers) and to outlying samples in the predictor space (leverage points) among the neighbors of the query. Each training sample gets the weight

$$\omega_i \, w_i^r \, w_i^t,$$

the product of its similarity weight ω_i, a weight w_i^r that decreases with its residual in the local model, and a weight w_i^t that decreases with its distance to the center of the scores, and the local model is refitted with the new weights until the norm of its regression coefficients changes by less than 1%. The weights are given by the Fair function $f(z) = 1/(1 + |z/c|)^2$ (default, $c = 4$) of the residuals and distances divided by their median, or by the Hampel function, which gives zero weight to the most outlying samples, with cutoffs at quantiles of the χ distribution (Hoffmann et al., 2015). As in PRM, the local center of the predictors is their L1-median, the intercept is the median of the residuals, and all these medians are weighted by the similarity weights, so the model stays local. For several outcomes, the residual weights use the norm of the scaled residual vectors; for classification, the indicators are centered by their (weighted) means so that the predicted indicators still sum to one.

The robust models also standardize the data by the median and the median absolute deviation (MAD), and use the MAD of the distances instead of their standard deviation in the similarity weights. With `similarity = "covariance"`, Γ is computed with the initial (Fair) weights of PRM.

The weights depend on the number of components, so a robust model is refitted for each value of `num_comp`: tuning `num_comp` costs about as many robust fits as values in the grid.

Neighbors

When `neighbors` is set, only the `neighbors` closest training samples receive a non-zero weight and σ_d is computed over their distances. This is the "KNN-LW" strategy, which Lesnoff et al. (2020) compare with the original LW-PLS that weights all training samples.

For classification, the outcome is converted to class indicator columns and modeled with a multi-response LW-PLS (LW-PLS-DA; Bevilacqua and Marini, 2014). The predicted class is the one with the largest predicted indicator. Class probabilities are obtained by truncating the predicted indicators to `[0, 1]` and renormalizing them to sum to one.

Because LW-PLS is a lazy learner, fitting only stores the (standardized) training data; the computational work happens at prediction time. A prediction for `num_comp` components also yields the predictions for all smaller numbers of components, so `tune::tune_grid()` evaluates every value of `num_comp` from a single model (the "submodel trick").

Engine details

The "lwpls" engine calls lwpls_fit(). It accepts the engine arguments:

- scale (logical, default TRUE): should the predictors (and numeric outcomes) be standardized to unit variance? Set it to FALSE for spectra and other data measured on a common scale.
- robust (logical, default FALSE): robust local models (see above).
- weight_function ("fair", the default, or "hampel"), robust_constant and max_iter (default 30): the settings of the robust models, see lwpls_fit().

Predictors with factor columns are converted to dummy variables when the model is fitted with a formula.

Tuning parameters:

- num_comp: dials::num_comp()
- localization: localization()
- neighbors: dials::neighbors() (default range 10–200)
- similarity: similarity()
- sparsity: sparsity()

References

Kim, S., Kano, M., Nakagawa, H. and Hasebe, S. (2011). Estimation of active pharmaceutical ingredients content using locally weighted partial least squares and statistical wavelength selection. *International Journal of Pharmaceutics*, 421(2), 269–274. doi:10.1016/j.ijpharm.2011.10.007

Bevilacqua, M. and Marini, F. (2014). Local classification: Locally weighted-partial least squares-discriminant analysis (LW-PLS-DA). *Analytica Chimica Acta*, 838, 20–30. doi:10.1016/j.aca.2014.05.057

Hazama, K. and Kano, M. (2015). Covariance-based locally weighted partial least squares for high-performance adaptive modeling. *Chemometrics and Intelligent Laboratory Systems*, 146, 55–62. doi:10.1016/j.chemolab.2015.05.007

Serneels, S., Croux, C., Filzmoser, P. and Van Espen, P. J. (2005). Partial robust M-regression. *Chemometrics and Intelligent Laboratory Systems*, 79(1–2), 55–64. doi:10.1016/j.chemolab.2005.04.007

Hoffmann, I., Serneels, S., Filzmoser, P. and Croux, C. (2015). Sparse partial robust M regression. *Chemometrics and Intelligent Laboratory Systems*, 149, 50–59. doi:10.1016/j.chemolab.2015.09.019

Lesnoff, M., Metz, M. and Roger, J.-M. (2020). Comparison of locally weighted PLS strategies for regression and discrimination on agronomic NIR data. *Journal of Chemometrics*, 34(5), e3209. doi:10.1002/cem.3209

See Also

lwpls_fit() for the underlying fitting function, multi_predict._lwpls_fit(), localization(), similarity(), sparsity().

Examples

```
lwpls(num_comp = 3, localization = 0.5) |>
  parsnip::set_mode("regression") |>
  parsnip::translate()

lwpls_mod <-
  lwpls(num_comp = 3, localization = 0.5) |>
  parsnip::set_mode("regression") |>
  parsnip::fit(mpg ~ ., data = mtcars[-(1:5), ])

lwpls_mod
predict(lwpls_mod, new_data = mtcars[1:5, ])

# Robust and sparse local models
lwpls(num_comp = 3, sparsity = 0.3) |>
  parsnip::set_mode("regression") |>
  parsnip::set_engine("lwpls", robust = TRUE) |>
  parsnip::fit(mpg ~ ., data = mtcars[-(1:5), ]) |>
  predict(new_data = mtcars[1:5, ])

# Classification
lwpls(num_comp = 2) |>
  parsnip::set_mode("classification") |>
  parsnip::fit(Species ~ ., data = iris) |>
  predict(new_data = iris[c(1, 51, 101), ], type = "prob")
```

`lwpls_fit` *Fit a locally-weighted partial least squares model*

Description

`lwpls_fit()` is the computational engine behind `lwpls()`. It can also be used on its own through the matrix, data frame, formula or recipe interfaces.

LW-PLS is a lazy learner: fitting validates and standardizes the training data and stores it together with the hyperparameters. The local models are built at prediction time (see `predict.lwpls_fit()`).

Sparse (`sparsity > 0`) and robust (`robust = TRUE`) local models are described in `lwpls()`.

Usage

```
lwpls_fit(x, ...)

## Default S3 method:
lwpls_fit(x, ...)

## S3 method for class 'data.frame'
lwpls_fit(
  x,
  y,
```

```
  num_comp = 2L,
  localization = 1,
  neighbors = NULL,
  similarity = "euclidean",
  sparsity = 0,
  scale = TRUE,
  robust = FALSE,
  weight_function = "fair",
  robust_constant = NULL,
  max_iter = 30L,
  ...
)

## S3 method for class 'matrix'
lwpls_fit(
  x,
  y,
  num_comp = 2L,
  localization = 1,
  neighbors = NULL,
  similarity = "euclidean",
  sparsity = 0,
  scale = TRUE,
  robust = FALSE,
  weight_function = "fair",
  robust_constant = NULL,
  max_iter = 30L,
  ...
)

## S3 method for class 'formula'
lwpls_fit(
  formula,
  data,
  num_comp = 2L,
  localization = 1,
  neighbors = NULL,
  similarity = "euclidean",
  sparsity = 0,
  scale = TRUE,
  robust = FALSE,
  weight_function = "fair",
  robust_constant = NULL,
  max_iter = 30L,
  ...
)

## S3 method for class 'recipe'
```

```
lwpls_fit(
  x,
  data,
  num_comp = 2L,
  localization = 1,
  neighbors = NULL,
  similarity = "euclidean",
  sparsity = 0,
  scale = TRUE,
  robust = FALSE,
  weight_function = "fair",
  robust_constant = NULL,
  max_iter = 30L,
  ...
)
```

Arguments

x	Depending on the context: • A **data frame** or **matrix** of numeric predictors. • A **recipe** specifying a set of preprocessing steps created from `recipes::recipe()`.
...	Not currently used, but required for extensibility.
y	When x is a **data frame** or **matrix**, y is the outcome: • A **factor** (or a one-column data frame with a factor) for classification. • A **numeric vector**, or a **matrix/data frame** with one or more numeric columns, for regression.
num_comp	The number of PLS components in each local model. It is capped at the maximum possible value, `min(ncol(x), nrow(x) - 1)`.
localization	A positive number: the localization parameter of the similarity weights (see `lwpls()`). Smaller values give more local models.
neighbors	Either `NULL` (default), to use all training samples, or the number of nearest training samples used to build each local model.
similarity	The similarity index used to weight the training samples: `"euclidean"` (default) or `"covariance"` (covariance-based LW-PLS, CbLW-PLS). See `lwpls()` for details.
sparsity	A number in `[0, 1)`: the sparsity threshold η of the local models (Hoffmann et al., 2015). Zero (default) gives ordinary local PLS models.
scale	A logical: should the predictors (and numeric outcomes) be standardized to unit variance? Predictors are always centered, which does not affect the predictions. For robust models, the center and scale are the median and the median absolute deviation (MAD).
robust	A logical: should the local models be robust to outliers (partial robust M-regression, Serneels et al., 2005)?
weight_function	The weight function of the robust models: `"fair"` (default) or `"hampel"`.

`robust_constant`
: The tuning constant(s) of the weight function: for `"fair"`, the constant c of the Fair function (default 4); for `"hampel"`, three increasing probabilities that define the cutoffs (default `c(0.95, 0.975, 0.999)`). `NULL` uses the defaults.

`max_iter`
: The maximum number of iterations of the robust models.

`formula`
: A formula specifying the outcome terms on the left-hand side and the predictor terms on the right-hand side. Multiple numeric outcomes can be given as `y1 + y2 ~ .`.

`data`
: When a **recipe** or **formula** is used, `data` is a data frame containing both the predictors and the outcome(s).

Value

A `lwpls_fit` object, which stores the standardized training data, the standardization constants and the hyperparameters.

See Also

`predict.lwpls_fit()`, `lwpls()`

Examples

```
train <- mtcars[-(1:5), ]
test <- mtcars[1:5, ]

# XY interface
fit <- lwpls_fit(train[, -1], train$mpg, num_comp = 3, localization = 0.5)
fit
predict(fit, test[, -1])

# Formula interface, with two outcomes
fit2 <- lwpls_fit(mpg + qsec ~ ., data = train, num_comp = 3)
predict(fit2, test)

# Classification
fit3 <- lwpls_fit(Species ~ ., data = iris, num_comp = 2)
predict(fit3, iris[c(1, 51, 101), ], type = "prob")

# Robust and sparse local models
fit4 <- lwpls_fit(mpg ~ ., data = train, num_comp = 3, robust = TRUE, sparsity = 0.3)
fit4
predict(fit4, test)
```

`lwpls_local` *Local models of LW-PLS predictions*

Description

`lwpls_local()` returns the local models that `lwpls()` builds to predict new samples, with the diagnostics that matter in practice, such as for spectroscopic calibration:

- **Reliability** of each prediction: the effective number of neighbors, the distance to the nearest training sample, and the Hotelling T^2 and Q statistics of the query in its local model, compared with theoretical and empirical limits and ranked among the training samples of the local model. Queries beyond the limits are extrapolations of their local model. See `plot_reliability()`.
- **Regression coefficients** and variable importance in projection (VIP) of each local model, on the original scale of the predictors and outcomes. See `plot_coefficients()`.
- **Selected predictors** of sparse local models. See `plot_selection()`.
- **Robust weights** of the training samples, averaged over the local models in which they take part. See `plot_robust_weights()`.

Usage

```
lwpls_local(object, ...)

## Default S3 method:
lwpls_local(object, ...)

## S3 method for class 'lwpls_fit'
lwpls_local(
  object,
  new_data,
  num_comp = NULL,
  wavelength = NULL,
  limits = c("empirical", "theoretical"),
  level = 0.95,
  ...
)

## S3 method for class 'model_fit'
lwpls_local(object, new_data, ...)

## S3 method for class 'workflow'
lwpls_local(object, new_data, ...)
```

Arguments

`object`	A fitted `lwpls()` model: a `lwpls_fit` object from `lwpls_fit()`, a parsnip model fit, or a fitted workflow.
`...`	Not currently used.
`new_data`	A data frame or matrix of new samples (the queries).
`num_comp`	The number of components of the local models. Defaults to the value used to fit the model.

`wavelength`	An optional numeric vector with the position of each predictor on the spectral axis (wavelengths, wavenumbers, ...). By default, the positions are the numbers at the end of the predictor names (for example `nm_1650` or `x_001`) when they are all present and unique, and the column numbers otherwise.
`limits`	The limits that define the `status` and the ratios of the reliability table: `"empirical"` (default) or `"theoretical"`. Both are always computed; see Details.
`level`	The coverage of the limits, and the percentile rank beyond which a query is flagged. Defaults to 0.95.

Details

For a query with local model weights w_i (the similarity weights, multiplied by the robust weights for robust models), the effective number of neighbors is $(\sum w_i)^2 / \sum w_i^2$.

The T^2 statistic of a sample is its squared distance to the weighted mean of the local scores, scaled by their weighted variances, and its Q statistic is the squared norm of its residual (on the standardized scale). They are computed for the query and for the training samples of its local model, and the query is compared with them in three ways:

- **Theoretical limits** (`t2_limit_theoretical`, `q_limit_theoretical`): the T^2 limit of a new observation for a model with a components fitted on n samples, $a(n^2-1)/(n(n-a))F_{level}(a, n-a)$, where n is the effective number of samples of the local model, and the Box approximation of the weighted distribution of the Q of the training samples (Nomikos and MacGregor, 1995). They assume normally distributed scores and residuals, and they are derived for ordinary, unweighted models.
- **Empirical limits** (`t2_limit_empirical`, `q_limit_empirical`): the `level` quantiles of the T^2 and Q of the training samples of the local model, weighted by their weights in the model. They make no distributional assumption. The training samples were used to fit the model, so their statistics are slightly smaller than those of new samples, which makes these limits slightly conservative (more queries are flagged).
- **Percentile ranks** (`t2_percentile`, `q_percentile`): the weighted fraction of the training samples of the local model whose statistic is not larger than that of the query. A rank above `level` is equivalent to exceeding the empirical limit, on a continuous scale.

A query is flagged as `"few neighbors"` when its local model has too few effective samples to assess it (no more than $a+1$), whatever the limits.

Value

An object of class `lwpls_local`, a list with:

- `reliability`: a tibble with one row per query: `.row`, the prediction(s), `n_eff`, `nearest_distance`, `num_comp`, `t2`, `t2_limit_theoretical`, `t2_limit_empirical`, `t2_percentile`, `q`, `q_limit_theoretical`, `q_limit_empirical`, `q_percentile`, `t2_ratio` and `q_ratio` (the statistics divided by the limits chosen by `limits`), and `status`.
- `coefficients`: a tibble with one row per query, outcome and predictor: `.row`, `outcome`, `predictor`, `position`, `coefficient`, `vip` and `selected`.

- `selection`: a tibble with one row per component and predictor: `component`, `predictor`, `position`, `frequency` (the fraction of the local models with this component whose weight vector selects the predictor) and `queries` (the number of those local models). Sparse models select predictors per component; a predictor is used by a local model (`selected` in `coefficients`) if any component selects it.
- `training`: a tibble with one row per training sample: `.row`, the outcome(s), `influence` (the sum of its similarity weights over the queries) and `robust_weight` (the average of its robust weights, weighted by the similarity weights).
- `info`: the settings of the model and of the spectral axis.

References

Nomikos, P. and MacGregor, J. F. (1995). Multivariate SPC charts for monitoring batch processes. *Technometrics*, 37(1), 41–59. doi:10.1080/00401706.1995.10485888

See Also

`plot_reliability()`, `plot_coefficients()`, `plot_robust_weights()`, `plot_selection()`

Examples

```
fit <- lwpls_fit(mpg ~ ., data = mtcars[-(1:5), ], num_comp = 3)
local <- lwpls_local(fit, mtcars[1:5, ])
local
local$reliability

# Theoretical limits at 99%
lwpls_local(fit, mtcars[1:5, ], limits = "theoretical", level = 0.99)
```

---

lwpls_plots *Plots of LW-PLS local models*

Description

Publication-ready plots of the local models returned by `lwpls_local()`:

- `plot_reliability()`: the T^2 and Q statistics of the queries divided by their theoretical or empirical limits, or their percentile ranks among the training samples of their local models (see `lwpls_local()`). Queries beyond a limit are extrapolations of their local model, and their predictions should be checked.
- `plot_coefficients()`: the regression coefficients (or VIP) of the local models along the spectral axis, as one line per query colored by its prediction, or as a heatmap of the queries ordered by their prediction. They show which bands drive the predictions and how they change across the range of the outcome.
- `plot_robust_weights()`: the robust weights of the training samples (robust models), averaged over the local models in which they take part. Samples with low weights, often wrong reference values, are labeled.

- `plot_selection()`: how often each predictor is selected by the sparse local models (sparse models): for each component (default), or by the local models as a whole (any component), with the mean spectrum for reference. On smooth spectra, the components of a sparse model often select different bands, so that almost all predictors are used by the model as a whole: the selection per component is then more informative.

All plots are ggplot2 objects that can be modified further.

Usage

```
plot_reliability(object, reference = NULL, labels = 5, log = FALSE, ...)

plot_coefficients(
  object,
  type = c("coefficients", "vip"),
  style = c("lines", "heatmap"),
  outcome = NULL,
  reverse = FALSE,
  axis_label = NULL,
  ...
)

plot_robust_weights(object, labels = 5, outcome = NULL, ...)

plot_selection(
  object,
  by = c("component", "model"),
  spectrum = TRUE,
  reverse = FALSE,
  axis_label = NULL,
  ...
)

## S3 method for class 'lwpls_local'
autoplot(
  object,
  type = c("reliability", "coefficients", "robust_weights", "selection"),
  ...
)
```

Arguments

`object`	An `lwpls_local` object from `lwpls_local()`.
`reference`	For `plot_reliability()`: `"empirical"` or `"theoretical"` to plot the statistics divided by these limits, or `"percentile"` to plot their percentile ranks. Defaults to the `limits` used in `lwpls_local()`.
`labels`	The number of points to label with their row number: the queries with the largest statistics beyond the limits, or the training samples with the smallest robust weights.

log	A logical: should the axes be on a log scale? Not available for percentile ranks.
...	Arguments passed to the plot function, for autoplot(). Not used otherwise.
type	For autoplot(), the plot: "reliability", "coefficients", "robust_weights" or "selection". For plot_coefficients(), "coefficients" (default) or "vip".
style	"lines" (default) or "heatmap".
outcome	The outcome (or class) whose coefficients are shown, or that is used on the x axis of plot_robust_weights(). Defaults to the first.
reverse	A logical: should the spectral axis be reversed, as is usual for wavenumbers?
axis_label	The label of the spectral axis. Defaults to "Wavelength" or "Predictor", see lwpls_local().
by	For plot_selection(), "component" (default) for the selection frequency of each predictor in each component, or "model" for the fraction of local models that use each predictor in any component.
spectrum	A logical: should the mean spectrum of the training data, rescaled to the range of the selection frequencies, be drawn? Only for by = "model".

Value

A ggplot object.

Examples

```
fit <- lwpls_fit(mpg ~ ., data = mtcars[-(1:5), ], num_comp = 3, robust = TRUE, sparsity = 0.3)
local <- lwpls_local(fit, mtcars[1:5, ])

plot_reliability(local)
plot_coefficients(local)
plot_robust_weights(local)
plot_selection(local)
```

---

multi_predict._lwpls_fit

Predictions for several numbers of components

Description

multi_predict() returns the predictions of a fitted lwpls() model for several values of num_comp at once. All values are obtained from a single pass over the data.

Usage

```
## S3 method for class '`_lwpls_fit`'
multi_predict(object, new_data, type = NULL, num_comp = NULL, ...)
```

Arguments

object	A parsnip::model_fit object created from a lwpls() specification.
new_data	A rectangular data object, such as a data frame.
type	A single character value or NULL. Possible values are "numeric", "class", or "prob". When NULL, "numeric" is used for regression and "class" for classification.
num_comp	An integer vector with the numbers of components. Defaults to the value used to fit the model.
...	Not currently used.

Value

A tibble with the same number of rows as new_data and a list column .pred of tibbles, each with a num_comp column and the prediction column(s).

Examples

```
fit <-
  lwpls(num_comp = 4) |>
  parsnip::set_mode("regression") |>
  parsnip::fit(mpg ~ ., data = mtcars[-(1:5), ])

preds <- parsnip::multi_predict(fit, mtcars[1:5, ], num_comp = 1:4)
preds
preds$.pred[[1]]
```

---

predict.lwpls_fit *Predict from a locally-weighted PLS model*

Description

For every row of new_data, a weighted PLS model is fitted around that sample and used to predict it.

Usage

```
## S3 method for class 'lwpls_fit'
predict(object, new_data, type = NULL, num_comp = NULL, ...)
```

Arguments

object	A lwpls_fit object created by lwpls_fit().
new_data	A data frame or matrix of new predictors.
type	A single character string, or NULL. One of: • "numeric" for numeric predictions (regression; the default).

- `"class"` for hard class predictions (classification; the default).
- `"prob"` for class probabilities (classification).
- `"raw"` for an array with the predictions of the local models with `1, ..., num_comp` components (see Value).

`num_comp` The number of PLS components to use. Defaults to the value used in `lwpls_fit()`. Since the model is lazy, any value can be used; values larger than the maximum possible number of components give the same predictions as that maximum.

`...` Not used, but required for extensibility.

Value

A tibble of predictions with one row per row of `new_data`, following the tidymodels conventions: `.pred` (or `.pred_{outcome}` for multiple outcomes), `.pred_class`, or `.pred_{level}` columns.

For `type = "raw"`, a numeric array of dimension `nrow(new_data) x q x num_comp`, where `q` is the number of outcomes (or of classes, for the predicted class indicators), on the original outcome scale.

Rows of `new_data` with missing values get missing predictions.

See Also

`lwpls_fit()`, `multi_predict._lwpls_fit()`

Examples

```
train <- mtcars[-(1:5), ]
test <- mtcars[1:5, ]

fit <- lwpls_fit(mpg ~ ., data = train, num_comp = 3, localization = 0.5)
predict(fit, test)
predict(fit, test, num_comp = 1)
predict(fit, test, type = "raw")
```

`similarity` *Similarity index for LW-PLS*

Description

The similarity index determines how the distance between a query sample and the training samples is measured before it is turned into weights by `lwpls()`:

Usage

```
similarity(values = values_similarity)

values_similarity
```

Arguments

values	A character vector of possible similarity indexes.

Details

- "euclidean": the Euclidean distance between the (standardized) predictors, as in the original LW-PLS.
- "covariance": the distance after projecting the samples on the covariance direction $\Gamma = X^\top Y / \|X^\top Y\|$ (covariance-based LW-PLS, CbLW-PLS; Hazama and Kano, 2015). It accounts for the relationships among the predictors and between the predictors and the outcome(s).

See vignette("similarity") for how to choose between them.

Value

A qual_param object (see dials::new_qual_param()).

References

Hazama, K. and Kano, M. (2015). Covariance-based locally weighted partial least squares for high-performance adaptive modeling. *Chemometrics and Intelligent Laboratory Systems*, 146, 55–62. doi:10.1016/j.chemolab.2015.05.007

Examples

```
similarity()
values_similarity
```

---

sparsity *Sparsity of the LW-PLS local models*

Description

The sparsity threshold η of lwpls(): the weight vector of each component of a local model is soft-thresholded at $\eta \max_j |w_j|$ (sparse NIPALS, Hoffmann et al., 2015). Zero gives ordinary local PLS models; larger values drop more predictors.

Usage

```
sparsity(range = c(0, 0.9), trans = NULL)
```

Arguments

range	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.
trans	A trans object from the scales package, or NULL (the default) for no transformation.

Value

A quant_param object (see dials::new_quant_param()).

References

Hoffmann, I., Serneels, S., Filzmoser, P. and Croux, C. (2015). Sparse partial robust M regression. *Chemometrics and Intelligent Laboratory Systems*, 149, 50–59. doi:10.1016/j.chemolab.2015.09.019

Examples

```
sparsity()
dials::value_seq(sparsity(), 4)
```

---

trimmed_rmse *Trimmed root mean squared error*

Description

The root mean squared error of the 1 - trim fraction of the observations with the smallest absolute errors. When the reference values contain outliers, they also appear in the assessment sets of a resampling scheme, where their errors would dominate the usual RMSE and favor non-robust models. The trimmed RMSE leaves them out, like the trimmed standard error of prediction used to tune partial robust M-regression (Serneels et al., 2005), so it is the metric to use when tuning robust LW-PLS models (lwpls() with robust = TRUE) on contaminated data.

Usage

```
trimmed_rmse(data, ...)

## S3 method for class 'data.frame'
trimmed_rmse(
  data,
  truth,
  estimate,
  trim = 0.2,
  na_rm = TRUE,
  case_weights = NULL,
  ...
)

trimmed_rmse_vec(
  truth,
  estimate,
  trim = 0.2,
  na_rm = TRUE,
  case_weights = NULL,
  ...
)
```

Arguments

`data`	A data frame containing the columns given by `truth` and `estimate`.
`...`	Not currently used.
`truth`	The column identifier for the true results (numeric).
`estimate`	The column identifier for the predicted results (numeric).
`trim`	The fraction of observations with the largest absolute errors to leave out, in `[0, 1)`. The default, 0.2, leaves out 20%.
`na_rm`	A logical: should missing values be removed?
`case_weights`	The optional column identifier for case weights.

Value

A tibble with columns `.metric`, `.estimator` and `.estimate` and one row (or one row per group), or, for `trimmed_rmse_vec()`, a single number.

References

Serneels, S., Croux, C., Filzmoser, P. and Van Espen, P. J. (2005). Partial robust M-regression. *Chemometrics and Intelligent Laboratory Systems*, 79(1–2), 55–64. doi:10.1016/j.chemolab.2005.04.007

Examples

```
df <- data.frame(truth = c(1, 2, 3, 4, 5), estimate = c(1.1, 2.1, 2.9, 4.2, 15))
trimmed_rmse(df, truth, estimate)
yardstick::rmse(df, truth, estimate)

# In a metric set, for example with tune::tune_grid()
metrics <- yardstick::metric_set(trimmed_rmse, yardstick::rmse)
metrics(df, truth, estimate)
```

`update.lwpls` *Update a locally-weighted PLS specification*

Description

Update a locally-weighted PLS specification

Usage

```
## S3 method for class 'lwpls'
update(
  object,
  parameters = NULL,
  num_comp = NULL,
  localization = NULL,
  neighbors = NULL,
```

```
  similarity = NULL,
  sparsity = NULL,
  fresh = FALSE,
  ...
)
```

Arguments

`object`	A `lwpls()` model specification.
`parameters`	A 1-row tibble or named list with *main* parameters to update. Use **either** `parameters` **or** the main arguments directly when updating. If the main arguments are used, these will supersede the values in `parameters`. Also, using engine arguments in this object will result in an error.
`num_comp`	The number of PLS components (latent variables) in each local model (engine default: 2).
`localization`	A positive number controlling how local the models are (engine default: 1). Small values give large weights to the closest training samples only; large values give all samples similar weights, so that LW-PLS tends to a global PLS model. See Details.
`neighbors`	The number of nearest training samples used for each local model. The default (`NULL`) uses all training samples, as in the original LW-PLS algorithm.
`similarity`	The similarity index: `"euclidean"` (engine default) or `"covariance"` for covariance-based LW-PLS (CbLW-PLS). See Details and `similarity()`.
`sparsity`	A number in `[0, 1)`: the sparsity threshold η of the local models (engine default: 0, no sparsity). See Details and `sparsity()`.
`fresh`	A logical for whether the arguments should be modified in-place or replaced wholesale.
`...`	Not used for `update()`.

Value

An updated model specification.

Examples

```
spec <- lwpls(num_comp = 3)
spec
update(spec, localization = 0.25)
update(spec, localization = 0.25, fresh = TRUE)
```

Index

autoplot.lwpls_local *(*lwpls_plots*)*, 13

dials::neighbors(), *6*
dials::new_qual_param(), *18*
dials::new_quant_param(), *3*, *19*
dials::num_comp(), *6*

localization, 2
localization(), *6*
lwpls, 3
lwpls(), *2*, *7*, *9–11*, *15–19*, *21*
lwpls_fit, 7
lwpls_fit(), *6*, *11*, *16, 17*
lwpls_local, 10
lwpls_local(), *13–15*
lwpls_plots, 13

multi_predict._lwpls_fit, 15
multi_predict._lwpls_fit(), *6*, *17*

parsnip, *3*
parsnip::model_fit, *16*
plot_coefficients *(*lwpls_plots*)*, 13
plot_coefficients(), *11*, *13*
plot_reliability *(*lwpls_plots*)*, 13
plot_reliability(), *11*, *13*
plot_robust_weights *(*lwpls_plots*)*, 13
plot_robust_weights(), *11*, *13*
plot_selection *(*lwpls_plots*)*, 13
plot_selection(), *11*, *13*
predict.lwpls_fit, 16
predict.lwpls_fit(), *7*, *10*

recipes::recipe(), *9*

scales::transform_log10(), *3*
scales::transform_reciprocal(), *3*
similarity, 17
similarity(), *4*, *6*, *21*
sparsity, 18
sparsity(), *4*, *6*, *21*

trimmed_rmse, 19
trimmed_rmse.data.frame *(*trimmed_rmse*)*, 19
trimmed_rmse_vec *(*trimmed_rmse*)*, 19
tune::tune_grid(), *5*

update.lwpls, 20

values_similarity *(*similarity*)*, 17