Package 'resemble'

May 9, 2026

Type Package

Title Similarity Retrieval and Local Learning for Spectral Chemometrics

Version 3.0.0

Date 2026-04-15

BugReports https://github.com/l-ramirez-lopez/resemble/issues

Description Functions for dissimilarity analysis and machine learning in complex spectral data sets, including memory-based learning (MBL), optimal subset search and selection, and retrieval-based modelling with model libraries. Supports local learning, optimisation of spectral libraries, and ensemble prediction from precomputed models.
Most of these functions are based on the methods presented in Ramirez-Lopez et al. (2013) <doi:10.1016/j.geoderma.2012.12.014>.

License MIT + file LICENSE

URL https://l-ramirez-lopez.github.io/resemble/,
https://github.com/l-ramirez-lopez/resemble

Depends R (>= 4.2.0)

Imports foreach, iterators, RhpcBLASctl, Rcpp (>= 1.0.3), mathjaxr (>= 1.0), lifecycle (>= 0.2.0)

Suggests prospectr, parallel, doParallel, testthat, quarto, knitr

LinkingTo Rcpp, RcppArmadillo

RdMacros mathjaxr

VignetteBuilder quarto

NeedsCompilation yes

Repository CRAN

RoxygenNote 7.3.3

Encoding UTF-8

Config/VersionName vortex

Author Leonardo Ramirez-Lopez [aut, cre] (ORCID:
<https://orcid.org/0000-0002-5369-5120>),
Antoine Stevens [aut, ctb] (ORCID:
<https://orcid.org/0000-0002-1588-7519>),
Claudio Orellano [ctb]

Maintainer Leonardo Ramirez-Lopez <ramirez.lopez.leo@gmail.com>

Date/Publication 2026-04-20 20:40:02 UTC

Contents

`resemble-package` *Overview of the functions in the resemble package*

Description

Maturing

Functions for dissimilarity assessment, nearest-neighbour search, memory-based learning, local expert libraries, and evolutionary training subset search in spectral chemometrics.

Details

This is the version 3.0.0 – vortex of the package. It implements a number of functions useful for modeling complex spectral spectra (e.g. NIR, IR). The package includes functions for dimensionality reduction, computing spectral dissimilarity matrices, nearest neighbor search, and modeling spectral data using memory-based learning and evolutionary search of optimal training subsets in large and complex datasets. This package builds upon the methods presented in Ramirez-Lopez et al. (2013a) doi:10.1016/j.geoderma.2012.12.014, Ramirez-Lopez et al. (2026a) and Ramirez-Lopez et al. (2026b).

Development versions can be found in the github repository of the package at https://github.com/l-ramirez-lopez/resemble.

The functions available for computing dissimilarity matrices are:

- `dissimilarity`: Computes a dissimilarity matrix based on a specified method.
- `diss_pca`: constructor for principal components-based dissimilarity method.
- `diss_pls`: constructor for partial least squares-based dissimilarity method.
- `diss_correlation`: constructor for correlation-based dissimilarity method.
- `diss_euclidean`: constructor for euclidean distance-based dissimilarity method.
- `diss_mahalanobis`: constructor for Mahalanobis distance-based dissimilarity method.
- `diss_cosine`: constructor for cosine-based dissimilarity method.

The functions available for evaluating dissimilarity matrices are:

- `diss_evaluate`: Evaluates the effectiveness of a dissimilarity matrix using side information.

The functions available for nearest neighbor search:

- `search_neighbors`: Search for nearest neighbors of a query spectrum in a reference dataset based on a specified dissimilarity method.

The functions available for modeling spectral data:

- `mbl`: Memory-based learning for modeling spectral data.
- `gesearch`: An evolutionary method to search optimal samples in large spectral datasets.
- `liblex`: Builds a library of reusable localized models.

The functions available for dimensionality reduction are:

- `ortho_projection`: Computes an orthogonal projection of the data based on either principal components or partial least squares.
- `predict.ortho_projection`

Author(s)

Maintainer / Creator: Leonardo Ramirez-Lopez <ramirez.lopez.leo@gmail.com>

Authors:

- Leonardo Ramirez-Lopez (ORCID)
- Antoine Stevens (ORCID)
- Claudio Orellano

References

Ramirez-Lopez, L., Viscarra Rossel, R., Behrens, T., Orellano, C., Perez-Fernandez, E., Kooijman, L., Wadoux, A. M. J.-C., Breure, T., Summerauer, L., Safanelli, J. L., & Plans, M. (2026a). When spectral libraries are too complex to search: Evolutionary subset selection for domain-adaptive calibration. *Analytica Chimica Acta*, under review.

Ramirez-Lopez, L., Metz, M., Lesnoff, M., Orellano, C., Perez-Fernandez, E., Plans, M., Breure, T., Behrens, T., Viscarra Rossel, R., & Peng, Y. (2026b). Rethinking local spectral modelling: From per-query refitting to model libraries. *Analytica Chimica Acta*, under review.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. (2013a). The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex data sets. *Geoderma* 195-196, 268-279.

See Also

Useful links:

- https://github.com/l-ramirez-lopez/resemble
- Report bugs at https://github.com/l-ramirez-lopez/resemble/issues

---

`dissimilarity` *Compute dissimilarity matrices*

Description

Computes dissimilarity matrices between observations using various methods. This is the main interface for dissimilarity computation in the resemble package.

Usage

```
dissimilarity(Xr, Xu = NULL, diss_method = diss_pca(), Yr = NULL)
```

Arguments

`Xr`	A numeric matrix of reference observations (rows) and variables (columns).
`Xu`	Optional matrix of additional observations with the same variables.
`diss_method`	A dissimilarity method object created by one of: • `diss_pca()`: Mahalanobis distance in PCA space • `diss_pls()`: Mahalanobis distance in PLS space • `diss_correlation()`: Correlation-based dissimilarity • `diss_euclidean()`: Euclidean distance • `diss_mahalanobis()`: Mahalanobis distance • `diss_cosine()`: Cosine dissimilarity Default is `diss_pca()`.
`Yr`	Optional response matrix. Required for PLS methods and when using `ncomp_by_opc()`.

Details

The function dispatches to the appropriate internal computation based on the class of `diss_method`. Each method constructor (e.g., `diss_pca()`) encapsulates all method-specific parameters including component selection, centering, scaling, and whether to return projections.

Output dimensions: When only `Xr` is provided, the function computes pairwise dissimilarities among all observations in `Xr`, returning a symmetric `nrow(Xr)` $\times$ `nrow(Xr)` matrix.

When both `Xr` and `Xu` are provided, the function computes dissimilarities between each observation in `Xr` and each observation in `Xu`, returning a `nrow(Xr)` $\times$ `nrow(Xu)` matrix where element (i, j) is the dissimilarity between the i-th observation in `Xr` and the j-th observation in `Xu`.

Mahalanobis distance: Note that `diss_mahalanobis()` computes Mahalanobis distance directly on the input variables. This requires the covariance matrix to be invertible, which fails when the number of variables exceeds the number of observations or when variables are highly correlated (common in spectral data). For such cases, use `diss_pca()` or `diss_pls()` instead.

Value

A list of class `"dissimilarity"` containing:

dissimilarity The computed dissimilarity matrix. Dimensions are `nrow(Xr)` $\times$ `nrow(Xr)` when `Xu = NULL`, or `nrow(Xr)` $\times$ `nrow(Xu)` otherwise.

diss_method The `diss_*` constructor object used for computation.

center Vector used to center the data.

scale Vector used to scale the data.

ncomp Number of components used (for projection methods).

projection If `return_projection = TRUE` in the method constructor, the `ortho_projection` object.

Author(s)

Leonardo Ramirez-Lopez

References

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. 2013a. The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex data sets. Geoderma 195-196, 268-279.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Viscarra Rossel, R., Dematte, J.A.M., Scholten, T. 2013b. Distance and similarity-search metrics for use with soil vis-NIR spectra. Geoderma 199, 43-53.

See Also

`diss_pca`, `diss_pls`, `diss_correlation`, `diss_euclidean`, `diss_mahalanobis`, `diss_cosine`

Examples

```
library(prospectr)
data(NIRsoil)

# Preprocess
sg <- savitzkyGolay(NIRsoil$spc, m = 1, p = 4, w = 15)

Xr <- sg[as.logical(NIRsoil$train), ]
Xu <- sg[!as.logical(NIRsoil$train), ]
Yr <- NIRsoil$CEC[as.logical(NIRsoil$train)]
Yu <- NIRsoil$CEC[!as.logical(NIRsoil$train)]

Xu <- Xu[!is.na(Yu), ]
Xr <- Xr[!is.na(Yr), ]
Yr <- Yr[!is.na(Yr)]

# PCA-based dissimilarity with variance-based selection
d1 <- dissimilarity(Xr, Xu, diss_method = diss_pca())

# PCA with OPC selection (requires Yr)
d2 <- dissimilarity(Xr, Xu,
  Yr = Yr,
  diss_method = diss_pca(
    ncomp = ncomp_by_opc(30),
    return_projection = TRUE
  )
)

# PLS-based dissimilarity
d3 <- dissimilarity(
  Xr, Xu,
  Yr = Yr,
  diss_method = diss_pls(
    ncomp = ncomp_by_opc(30)
  )
)

# Euclidean distance
d4 <- dissimilarity(Xr, Xu, diss_method = diss_euclidean())

# Correlation dissimilarity with moving window
d5 <- dissimilarity(Xr, Xu, diss_method = diss_correlation(ws = 41))

# Mahalanobis distance (use only when n > p and low collinearity)
# d6 <- dissimilarity(Xr[, 1:20], Xu[, 1:20],
#                     diss_method = diss_mahalanobis())
```

`diss_correlation` *Correlation dissimilarity method constructor*

Description

Creates a configuration object that fully specifies a correlation (or moving correlation) dissimilarity method. Pass the result to `dissimilarity()` to compute the dissimilarity matrix.

Usage

```
diss_correlation(ws = NULL, center = TRUE, scale = FALSE)
```

Arguments

`ws`	Either `NULL` (default) or an odd integer greater than 2. When `NULL`, standard Pearson correlation dissimilarity is used. When an odd integer is provided, a moving (rolling) correlation dissimilarity is computed using a window of that size.
`center`	Logical. Should the data be mean-centered before computing dissimilarities? Centering is applied jointly to `Xr` and `Xu` (if provided) based on their combined column means. Default is `TRUE`.
`scale`	Logical. Should the data be scaled (divided by column standard deviations) before computing dissimilarities? Scaling is applied jointly to `Xr` and `Xu` (if provided). Default is `FALSE`.

Details

The correlation dissimilarity between two observations x_i and x_j is:

$$d(x_i, x_j) = \frac{1}{2}(1 - \rho(x_i, x_j))$$

where ρ is the Pearson correlation coefficient. This is used when `ws = NULL`.

When `ws` is specified, the moving correlation dissimilarity is:

$$d(x_i, x_j; ws) = \frac{1}{2\,ws} \sum_{k=1}^{p-ws} \left(1 - \rho(x_{i,(k:k+ws)},\, x_{j,(k:k+ws)})\right)$$

where ws is the window size and p is the number of variables.

Value

An object of class `c("diss_correlation", "diss_method")` — a list holding the validated method parameters. Intended to be passed to `dissimilarity()`, not used directly.

Parallel execution

The underlying C++ implementation uses OpenMP for parallel computation. Thread count is controlled by the `OMP_NUM_THREADS` environment variable. To limit threads (e.g., when calling from within a parallel backend):

```
Sys.setenv(OMP_NUM_THREADS = 1)
```

Author(s)

Leonardo Ramirez-Lopez

See Also

dissimilarity, diss_euclidean, diss_mahalanobis, diss_cosine

Examples

```
# Standard correlation dissimilarity
m <- diss_correlation()

# Moving correlation with window size 41
m <- diss_correlation(ws = 41)

# Without centering
m <- diss_correlation(center = FALSE)
```

---

diss_cosine *Cosine dissimilarity method constructor*

Description

Creates a configuration object for computing cosine dissimilarity (also known as spectral angle mapper). Pass the result to `dissimilarity()` to compute the dissimilarity matrix.

The cosine dissimilarity between two observations x_i and x_j is:

$$c(x_i, x_j) = \cos^{-1} \frac{\sum_{k=1}^{p} x_{i,k}\, x_{j,k}}{\sqrt{\sum_{k=1}^{p} x_{i,k}^2}\, \sqrt{\sum_{k=1}^{p} x_{j,k}^2}}$$

where p is the number of variables.

Usage

```
diss_cosine(center = TRUE, scale = FALSE)
```

Arguments

center	Logical. Center the data before computing dissimilarities? Applied jointly to Xr and Xu if both are provided. Default TRUE.
scale	Logical. Scale the data before computing dissimilarities? Applied jointly to Xr and Xu if both are provided. Default FALSE.

Value

An object of class `c("diss_cosine", "diss_method")`.

Author(s)

Leonardo Ramirez-Lopez

See Also

dissimilarity, diss_euclidean, diss_mahalanobis

Examples

```
m <- diss_cosine()
m <- diss_cosine(center = FALSE)
```

---

diss_euclidean *Euclidean dissimilarity method constructor*

Description

Creates a configuration object for computing Euclidean dissimilarity. Pass the result to `dissimilarity()` to compute the dissimilarity matrix.

The scaled Euclidean dissimilarity between two observations x_i and x_j is:

$$d(x_i, x_j) = \sqrt{\frac{1}{p}\sum_{k=1}^{p}(x_{i,k} - x_{j,k})^2}$$

where p is the number of variables. Results are equivalent to `stats::dist()` but scaled by $1/p$.

Usage

```
diss_euclidean(center = TRUE, scale = FALSE)
```

Arguments

`center`	Logical. Center the data before computing distances? Applied jointly to `Xr` and `Xu` if both are provided. Default `TRUE`.
`scale`	Logical. Scale the data before computing distances? Applied jointly to `Xr` and `Xu` if both are provided. Default `FALSE`.

Value

An object of class `c("diss_euclidean", "diss_method")`.

Author(s)

Leonardo Ramirez-Lopez

See Also

dissimilarity, diss_mahalanobis, diss_cosine

Examples

```
m <- diss_euclidean()
m <- diss_euclidean(center = FALSE, scale = TRUE)
```

---

diss_evaluate *Evaluate dissimilarity matrices*

Description

[Stable]

Evaluates a dissimilarity matrix by comparing each observation to its nearest neighbor based on side information. For continuous variables, RMSD and correlation are computed; for categorical variables, the kappa index is used.

Usage

```
diss_evaluate(diss, side_info)

sim_eval(d, side_info)
```

Arguments

diss	A symmetric dissimilarity matrix. Alternatively, a vector containing the lower triangle values (as returned by dist).
side_info	A matrix of side information corresponding to the observations. Can be numeric (one or more columns) or character (single column for categorical data).
d	Deprecated. Use diss in diss_evaluate() instead.

Details

This function assesses whether a dissimilarity matrix captures meaningful structure by examining the side information of nearest neighbor pairs (Ramirez-Lopez et al., 2013). If observations that are similar in the dissimilarity space also have similar side information values, the dissimilarity is considered effective.

For numeric side_info, the root mean square of differences (RMSD) between each observation and its nearest neighbor is computed:

$$j(i) = NN(x_i, X^{-i})$$

$$RMSD = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - y_{j(i)})^2}$$

where $NN(x_i, X^{-i})$ returns the index of the nearest neighbor of observation i (excluding itself), y_i is the side information value for observation i, and m is the number of observations.

For categorical `side_info`, the kappa index is computed:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

where p_o is the observed agreement and p_e is the agreement expected by chance.

Value

A list with the following components:

eval For numeric side information: a matrix with columns `rmsd` and `r` (correlation). For categorical: a matrix with column `kappa`.

global_eval If multiple numeric side information variables are provided, summary statistics across variables.

first_nn A matrix with the original side information and the side information of each observation's nearest neighbor.

Author(s)

Leonardo Ramirez-Lopez

References

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. 2013a. The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex datasets. Geoderma 195-196, 268-279.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Viscarra Rossel, R., Dematte, J.A.M., Scholten, T. 2013b. Distance and similarity-search metrics for use with soil vis-NIR spectra. Geoderma 199, 43-53.

See Also

`dissimilarity`, `ncomp_by_opc`

Examples

```
library(prospectr)
data(NIRsoil)

sg <- savitzkyGolay(NIRsoil$spc, p = 3, w = 11, m = 0)
NIRsoil$spc <- sg

Yr <- NIRsoil$Nt[as.logical(NIRsoil$train)]
Xr <- NIRsoil$spc[as.logical(NIRsoil$train), ]

# Compute PCA-based dissimilarity
d <- dissimilarity(Xr, diss_method = diss_pca(ncomp = 8))
```

```
# Evaluate using side information
ev <- diss_evaluate(d$dissimilarity, side_info = as.matrix(Yr))
ev$eval

# Evaluate with multiple side information variables
Yr_2 <- NIRsoil$CEC[as.logical(NIRsoil$train)]
ev_2 <- diss_evaluate(d$dissimilarity, side_info = cbind(Yr, Yr_2))
ev_2$eval
ev_2$global_eval
```

`diss_mahalanobis` *Mahalanobis dissimilarity method constructor*

Description

Creates a configuration object for computing Mahalanobis dissimilarity. Pass the result to `dissimilarity()` to compute the dissimilarity matrix.

The Mahalanobis distance is computed by first transforming the data into Mahalanobis space via a factorization of the inverse covariance matrix $M^{-1} = W^T W$ (using SVD), then applying Euclidean distance in that transformed space:

$$d(x_i, x_j) = \sqrt{\frac{1}{p}(x_i - x_j)M^{-1}(x_i - x_j)^T}$$

Usage

```
diss_mahalanobis(center = TRUE, scale = FALSE)
```

Arguments

`center`	Logical. Center the data before computing distances? Applied jointly to `Xr` and `Xu` if both are provided. Default `TRUE`.
`scale`	Logical. Scale the data before computing distances? Applied jointly to `Xr` and `Xu` if both are provided. Default `FALSE`.

Value

An object of class `c("diss_mahalanobis", "diss_method")`.

Important limitations

The covariance matrix will be singular — and the distance therefore uncomputable — when the number of observations is smaller than the number of variables, or when variables are perfectly collinear. This is common with raw spectral data; consider using `diss_euclidean()` on PCA scores instead.

Author(s)

Leonardo Ramirez-Lopez

See Also

dissimilarity, diss_euclidean, diss_cosine

Examples

```
m <- diss_mahalanobis()
m <- diss_mahalanobis(center = TRUE, scale = TRUE)
```

---

diss_pca *PCA dissimilarity method constructor*

Description

Creates a configuration object for computing dissimilarity based on Mahalanobis distance in PCA score space.

Usage

```
diss_pca(
  ncomp = ncomp_by_var(0.01),
  method = c("pca", "pca_nipals"),
  center = TRUE,
  scale = FALSE,
  return_projection = FALSE
)
```

Arguments

ncomp	Component selection method. Can be: • A positive integer for a fixed number of components • ncomp_fixed(n): explicit fixed selection • ncomp_by_var(min_var): retain components explaining at least min_var variance each (default: ncomp_by_var(0.01)) • ncomp_by_cumvar(min_cumvar): retain components until cumulative variance reaches min_cumvar • ncomp_by_opc(): optimize using side information (Yr required in dissimilarity())
method	Character. PCA algorithm: "pca" (default, SVD-based) or "pca_nipals" (NIPALS algorithm).
center	Logical. Center data before projection? Default TRUE.
scale	Logical. Scale data before projection? Default FALSE.
return_projection	Logical. Return the projection object? Default FALSE.

Value

An object of class `c("diss_pca", "diss_method")`.

See Also

Component selection: `ncomp_by_var`, `ncomp_by_cumvar`, `ncomp_by_opc`, `ncomp_fixed`

Other dissimilarity methods: `diss_pls`, `diss_correlation`, `diss_euclidean`, `diss_cosine`, `diss_mahalanobis`

Examples

```
# Fixed number of components
diss_pca(ncomp = 10)
diss_pca(ncomp = ncomp_fixed(10))

# Retain components explaining >= 1% variance each (default)
diss_pca(ncomp = ncomp_by_var(0.01))

# Retain components until 99% cumulative variance
diss_pca(ncomp = ncomp_by_cumvar(0.99))

# Optimize using side information (requires Yr)
diss_pca(ncomp = ncomp_by_opc(40))
diss_pca(ncomp = ncomp_by_opc())

# NIPALS algorithm (useful for very large matrices)
diss_pca(ncomp = 10, method = "pca_nipals")
```

---

diss_pls *PLS dissimilarity method constructor*

Description

Creates a configuration object for computing dissimilarity based on Mahalanobis distance in PLS score space. Requires `Yr` in `dissimilarity()`.

Usage

```
diss_pls(
  ncomp = ncomp_by_opc(),
  method = c("pls", "mpls"),
  scale = FALSE,
  return_projection = FALSE
)
```

Arguments

ncomp	Component selection method. Can be: • A positive integer for a fixed number of components • ncomp_fixed(n): explicit fixed selection • ncomp_by_var(min_var): retain components explaining at least min_var variance each • ncomp_by_cumvar(min_cumvar): retain components until cumulative variance reaches min_cumvar • ncomp_by_opc(): optimize using side information (default; recommended for PLS since Yr is already required)
method	Character. PLS algorithm: "pls" (default) or "mpls" (modified PLS, Shenk & Westerhaus 1991).
scale	Logical. Scale data? Default FALSE. Note: PLS always centers internally.
return_projection	Logical. Return projection object? Default FALSE.

Value

An object of class c("diss_pls", "diss_method").

Author(s)

Leonardo Ramirez-Lopez

See Also

Component selection: ncomp_by_var, ncomp_by_cumvar, ncomp_by_opc, ncomp_fixed

Other dissimilarity methods: diss_pca, diss_correlation, diss_euclidean, diss_cosine, diss_mahalanobis

Examples

```
# Default: OPC optimization (recommended)
diss_pls()

# Fixed number of components
diss_pls(ncomp = 15)

# Custom opc settings
diss_pls(ncomp = ncomp_by_opc(max_ncomp = 50))

# Modified PLS
diss_pls(ncomp = 10, method = "mpls")
```

`fit_methods` *Local fitting method constructors*

Description

These functions create configuration objects that specify how local regression models are fitted within the `mbl` function.

Usage

```
fit_pls(ncomp, method = c("pls", "mpls", "simpls"),
        scale = FALSE, max_iter = 100L, tol = 1e-6)

fit_wapls(min_ncomp, max_ncomp, method = c("mpls", "pls", "simpls"),
          scale = FALSE, max_iter = 100L, tol = 1e-6)

fit_gpr(noise_variance = 0.001, center = TRUE, scale = TRUE)
```

Arguments

`ncomp`	an integer indicating the number of PLS components to use in local regressions when `fit_pls` is used.
`min_ncomp`	an integer indicating the minimum number of PLS components to use in local regressions when `fit_wapls` is used. See details.
`max_ncomp`	an integer indicating the maximum number of PLS components to use in local regressions when `fit_wapls` is used. See details.
`method`	a character string indicating the PLS algorithm to use. Options are: • `'pls'`: standard PLS using covariance between X and Y for weight computation (NIPALS algorithm). • `'mpls'`: modified PLS using correlation between X and Y for weight computation (NIPALS algorithm). See Shenk and Westerhaus (1991). • `'simpls'`: SIMPLS algorithm (de Jong, 1993). Computationally faster as it avoids iterative X deflation. Parameters `max_iter` and `tol` are ignored when this method is used. Default is `'pls'` for `fit_pls` and `'mpls'` for `fit_wapls`.
`scale`	logical indicating whether predictors must be scaled. Default is `FALSE` for PLS methods and `TRUE` for GPR.
`max_iter`	an integer indicating the maximum number of iterations for convergence in the NIPALS algorithm. Only used when `method = 'pls'` or `method = 'mpls'`. Default is 100.
`tol`	a numeric value indicating the convergence tolerance for calculating scores in the NIPALS algorithm. Only used when `method = 'pls'` or `method = 'mpls'`. Default is 1e-6.

`noise_variance` a numeric value indicating the variance of the noise for Gaussian process local regressions (`fit_gpr`). Default is 0.001.

`center` logical indicating whether predictors should be centered before fitting. Only used for `fit_gpr`. Default is `TRUE`.

Details

These functions create configuration objects that are passed to `mbl` to specify how local regression models are fitted.

There are three fitting methods available:

Partial least squares (`fit_pls`): Uses orthogonal scores partial least squares regression. Three algorithm variants are available:

- **Standard PLS** (`method = 'pls'`): Uses the NIPALS algorithm with covariance-based weights.
- **Modified PLS** (`method = 'mpls'`): Uses the NIPALS algorithm with correlation-based weights. Proposed by Shenk and Westerhaus (1991), this approach gives equal influence to all predictors regardless of their variance scale.
- **SIMPLS** (`method = 'simpls'`): Uses the SIMPLS algorithm (de Jong, 1993), which deflates the cross-product matrix rather than X itself. This is computationally faster, especially for wide matrices, and produces identical predictions to standard PLS.

The only parameter to optimise is the number of PLS components (`ncomp`).

Weighted average PLS (`fit_wapls`): This method was developed by Shenk et al. (1997) and is used as the regression method in the LOCAL algorithm. It fits multiple PLS models using different numbers of components (from `min_ncomp` to `max_ncomp`). The final prediction is a weighted average of predictions from all models, where the weight for component j is:

$$w_j = \frac{1}{s_{1:j} \times g_j}$$

where $s_{1:j}$ is the root mean square of the spectral reconstruction error of the target observation(s) when j PLS components are used, and g_j is the root mean square of the squared regression coefficients for the jth component.

The same algorithm variants (`'pls'`, `'mpls'`, `'simpls'`) are available. The default is `'mpls'` following the original LOCAL implementation.

Gaussian process regression (`fit_gpr`): Gaussian process regression is a non-parametric Bayesian method characterised by a mean and covariance function. This implementation uses a dot product covariance.

The prediction vector A is computed from training data (X, Y) as:

$$A = (XX^T + \sigma^2 I)^{-1} Y$$

where σ^2 is the noise variance and I is the identity matrix. Prediction for a new observation x_u is:

$$\hat{y}_u = x_u X^T A$$

The only parameter is the noise variance (`noise_variance`).

Value

An object of class `c("fit_<method>", "fit_method")` containing the specified parameters. This object is passed to `mbl` to configure local model fitting.

Author(s)

Leonardo Ramirez-Lopez

References

de Jong, S. (1993). SIMPLS: An alternative approach to partial least squares regression. Chemometrics and Intelligent Laboratory Systems, 18(3), 251-263.

Rasmussen, C.E., Williams, C.K. (2006). Gaussian Processes for Machine Learning. MIT Press.

Shenk, J.S., & Westerhaus, M.O. (1991). Populations structuring of near infrared spectra and modified partial least squares regression. Crop Science, 31(6), 1548-1555.

Shenk, J., Westerhaus, M., & Berzaghi, P. (1997). Investigation of a LOCAL calibration procedure for near infrared instruments. Journal of Near Infrared Spectroscopy, 5, 223-232.

Westerhaus, M. (2014). Eastern Analytical Symposium Award for outstanding achievements in near infrared spectroscopy: my contributions to near infrared spectroscopy. NIR news, 25(8), 16-20.

See Also

`mbl`

Examples

```
# PLS with 10 components using standard algorithm
fit_pls(ncomp = 10)

# PLS with modified algorithm (correlation-based weights)
fit_pls(ncomp = 10, method = "mpls")

# PLS with SIMPLS (faster, no iteration)
fit_pls(ncomp = 10, method = "simpls")

# Weighted average PLS (LOCAL-style)
fit_wapls(min_ncomp = 3, max_ncomp = 12)

# Weighted average PLS with SIMPLS
fit_wapls(min_ncomp = 3, max_ncomp = 15, method = "simpls")

# Gaussian process regression
fit_gpr()
fit_gpr(noise_variance = 0.01)
```

gesearch *Evolutionary sample search for context-specific calibrations*

Description

Implements an evolutionary search algorithm that selects a subset from large reference datasets (e.g., spectral libraries) to build context-specific calibrations. The algorithm iteratively removes weak or non-informative samples based on prediction error, spectral reconstruction error, or dissimilarity criteria. This implementation is based on the methods proposed in Ramirez-Lopez et al. (2026a).

Usage

```
## Default S3 method:
gesearch(Xr, Yr, Xu, Yu = NULL, Yu_lims = NULL,
         k, b, retain = 0.95, target_size = k,
         fit_method = fit_pls(ncomp = 10),
         optimization = "reconstruction",
         group = NULL, control = gesearch_control(),
         intermediate_models = FALSE,
         verbose = TRUE, seed = NULL, pchunks = 1L, ...)

## S3 method for class 'formula'
gesearch(formula, train, test, k, b, target_size, fit_method,
         ..., na_action = na.pass)

## S3 method for class 'gesearch'
predict(object, newdata, type = "response",
         what = c("final", "all_generations"), ...)

## S3 method for class 'gesearch'
plot(x, which = c("weakness", "removed"), ...)
```

Arguments

Xr	A numeric matrix of predictor variables for the reference data (observations in rows, variables in columns).
Yr	A numeric vector or single-column matrix of response values corresponding to Xr. Only one response variable is supported.
Xu	A numeric matrix of predictor variables for target observations (same structure as Xr).
Yu	An optional numeric vector or single-column matrix of response values for Xu. Required when optimization includes "response". Default is NULL.
Yu_lims	A numeric vector of length 2 specifying expected response limits for the target population. Used with optimization = "range".

k — An integer specifying the number of samples in each resampling subset (gene size).

b — An integer specifying the target average number of times each training sample is evaluated per iteration. Higher values (e.g., >40) produce more stable results but increase computation time.

retain — A numeric value in (0, 1] specifying the proportion of samples retained per iteration. Default is 0.95. Values >0.9 are recommended for stability. See gesearch_control for retention strategy.

target_size — An integer specifying the target number of selected samples (gene pool size). Must be >= k. Default is k.

fit_method — A fit method object created with fit_pls. Specifies the regression model and scaling used during the search. Currently only fit_pls() is supported.

optimization — A character vector specifying optimization criteria:

- "reconstruction": (default) Retains samples based on spectral reconstruction error of Xu in PLS space.
- "response": Retains samples based on RMSE of predicting Yu. Requires Yu.
- "similarity": Retains samples based on Mahalanobis distance between Xu and training samples in PLS score space.
- "range": Removes samples producing predictions outside Yu_lims.

Multiple criteria can be combined, e.g., c("reconstruction", "similarity").

group — An optional factor assigning group labels to training observations. Used for leave-group-out cross-validation to avoid pseudo-replication.

control — A list created with gesearch_control containing additional algorithm parameters.

intermediate_models — A logical indicating whether to store models for each intermediate generation. Default is FALSE.

verbose — A logical indicating whether to print progress information. Default is TRUE.

seed — An integer for random number generation to ensure reproducibility. Default is NULL.

pchunks — An integer specifying the chunk size used for memory-efficient parallel processing. Larger values divide the workload into smaller pieces, which can help reduce memory pressure. Default is 1L.

formula — A formula defining the model.

train — A data.frame containing training data with model variables.

test — A data.frame containing test data with model variables.

na_action — A function for handling missing values in training data. Default is na.pass.

object — A fitted gesearch object (for predict).

newdata — A matrix or data.frame of new observations. For formula-fitted models, a data.frame containing all predictor variables is accepted. For non-formula models, a matrix is required.

type	A character string specifying the prediction type. Currently only "response" is supported.
what	A character string specifying which models to use for prediction: "final" (default) for predictions from final models only, or "all_generations" for predictions from all intermediate generations plus the final models.
x	A gesearch object (for plot).
which	Character string specifying what to plot: "weakness" (maximum weakness scores per generation) or "removed" (cumulative samples removed).
...	Additional arguments passed to methods.

Details

The gesearch algorithm requires a large reference dataset (Xr) where the sample search is conducted, target observations (Xu), and three tuning parameters: k, b, and retain.

The target observations (Xu) should represent the population of interest. These may be selected via algorithms like Kennard-Stone when response values are unavailable.

The algorithm iteratively removes weak samples from Xr based on:

- Increased RMSE when predicting Yu
- Increased PLS reconstruction error on Xu
- Increased dissimilarity to Xu in PLS space

A resampling scheme identifies samples that consistently appear in high-error subsets. These are labeled weak and removed. The process continues until approximately target_size samples remain.

The gesearch() function also returns a final model fitted on the selected samples, which can be used for prediction. This model is internally validated by cross-validation using only the selected samples from the training/reference set. If Yu is available, a model fitted only on the selected reference samples is first used to predict the target samples. The final model is then refitted using both the selected reference samples and the target samples used to guide the search, provided that response values are available for those target samples.

Parameter guidance:

- k: Number of samples per resampling subset. See Lobsey et al. (2017) for guidance.
- b: Resampling intensity. Higher values increase stability but computational cost.
- retain: Proportion retained per iteration. Values >0.9 recommended.

Prediction:

The predict method generates predictions from a fitted gesearch object. If the model was fitted with a formula, newdata is validated and transformed to the appropriate model matrix.

When what = "all_generations", the return value is a named list with one element per generation, where each element contains a prediction matrix. This option requires intermediate_models = TRUE during fitting.

Value

For `gesearch`: A list of class `"gesearch"` containing:

- `x_local`: Matrix of predictors for selected samples.
- `y_local`: Vector of responses for selected samples.
- `indices`: Indices of selected samples from original training set.
- `complete_iter`: Number of completed iterations.
- `iter_weakness`: List with iteration-level weakness statistics.
- `samples`: List of sample indices retained at each iteration.
- `n_removed`: data.frame of samples removed per iteration.
- `control`: Copy of control parameters.
- `fit_method`: Fit constructor from `fit_method`.
- `validation_results`: Cross-validation in the training only set validation on the test set using models built only with the samples found.
- `final_models`: Final PLS model containing coefficients, loadings, scores, VIP, and selectivity ratios.
- `intermediate_models`: List of models per generation (if `intermediate_models = TRUE`).
- `seed`: RNG seed used.

For `predict.gesearch`:

- If `what = "final"`: a prediction matrix with `nrow(newdata)` rows and one column per PLS component.
- If `what = "all_generations"`: a named list of generations, where each generation contains a prediction matrix as above.

Author(s)

Leonardo Ramirez-Lopez, Claudio Orellano, Craig Lobsey, Raphael Viscarra Rossel

References

Lobsey, C.R., Viscarra Rossel, R.A., Roudier, P., Hedley, C.B. 2017. rs-local data-mines information from spectral libraries to improve local calibrations. European Journal of Soil Science 68:840-852.

Kennard, R.W., Stone, L.A. 1969. Computer aided design of experiments. Technometrics 11:137-148.

Rajalahti, T., Arneberg, R., Berven, F.S., Myhr, K.M., Ulvik, R.J., Kvalheim, O.M. 2009. Biomarker discovery in mass spectral profiles by means of selectivity ratio plot. Chemometrics and Intelligent Laboratory Systems 95:35-48.

Ramirez-Lopez, L., Viscarra Rossel, R., Behrens, T., Orellano, C., Perez-Fernandez, E., Kooijman, L., Wadoux, A. M. J.-C., Breure, T., Summerauer, L., Safanelli, J. L., & Plans, M. (2026a). When spectral libraries are too complex to search: Evolutionary subset selection for domain-adaptive calibration. *Analytica Chimica Acta*, under review.

See Also

fit_pls, gesearch_control, mbl

Examples

```
## Not run:
library(prospectr)
data(NIRsoil)

# Preprocess
sg_det <- savitzkyGolay(
  detrend(NIRsoil$spc, wav = as.numeric(colnames(NIRsoil$spc))),
  m = 1, p = 1, w = 7
)
NIRsoil$spc_pr <- sg_det

# Split data
train_x <- NIRsoil$spc_pr[NIRsoil$train == 1 & !is.na(NIRsoil$Ciso), ]
train_y <- NIRsoil$Ciso[NIRsoil$train == 1 & !is.na(NIRsoil$Ciso)]
test_x <- NIRsoil$spc_pr[NIRsoil$train == 0 & !is.na(NIRsoil$Ciso), ]
test_y <- NIRsoil$Ciso[NIRsoil$train == 0 & !is.na(NIRsoil$Ciso)]

# Basic search with reconstruction and similarity optimizations
gs <- gesearch(
  Xr = train_x, Yr = train_y,
  Xu = test_x, Yu = test_y,
  k = 50, b = 100, retain = 0.97,
  target_size = 200,
  fit_method = fit_pls(ncomp = 15, method = "mpls"),
  optimization = c("reconstruction", "similarity"),
  control = gesearch_control(retain_by = "probability"),
  seed = 42
)

# Predict
preds <- predict(gs, test_x)

# Plot progress
plot(gs)
plot(gs, which = "removed")

# With reconstruction and response optimization (requires Yu)
gs_response <- gesearch(
  Xr = train_x, Yr = train_y,
  Xu = test_x, Yu = test_y,
  k = 50, b = 100, retain = 0.97,
  target_size = 200,
  fit_method = fit_pls(ncomp = 15),
  optimization = c("reconstruction", "response"),
  seed = 42
)
```

```
# Parallel processing
library(doParallel)
n_cores <- min(2, parallel::detectCores() - 1)
cl <- makeCluster(n_cores)
registerDoParallel(cl)

gs_parallel <- gesearch(
  Xr = train_x, Yr = train_y,
  Xu = test_x,
  k = 50, b = 100, retain = 0.97,
  target_size = 200,
  fit_method = fit_pls(ncomp = 15),
  pchunks = 3,
  seed = 42
)

stopCluster(cl)
registerDoSEQ()

## End(Not run)
```

`gesearch_control` *Control parameters for gesearch*

Description

Creates a control object specifying algorithm parameters for `gesearch`.

Usage

```
gesearch_control(
  retain_by = c("probability", "proportion"),
  percentile_type = 7L,
  tune = FALSE,
  number = 10L,
  p = 0.75,
  stagnation_limit = 5L,
  allow_parallel = TRUE,
  blas_threads = 1L
)
```

Arguments

`retain_by` A character string specifying how training observations are selected at each iteration:

`"probability"` (default) Retains observations with errors below a percentile estimated from a given probability. More robust to outliers.

"proportion" Retains a fixed proportion of observations with lowest errors.

`percentile_type`
: An integer between 1 and 9 specifying the quantile algorithm when `retain_by = "probability"`. Passed to the `type` argument of `quantile`. Default is 7. Ignored when `retain_by = "proportion"`.

`tune`
: A logical indicating whether to tune regression parameters (e.g., number of PLS components) via cross-validation at each iteration. Increases computation time substantially. Default is `FALSE`.

`number`
: An integer specifying the number of groups for leave-group-out cross-validation. Default is 10. This is used for validating the final models built the samples found. When `tune = TRUE`, this value is also used in the internal CV for tuning regression parameters at each iteration.

`p`
: A numeric value in (0, 1) specifying the proportion of observations per group in leave-group-out cross-validation. Default is 0.75. When `tune = TRUE`, this value is also used in the internal CV for tuning regression parameters at each iteration.

`stagnation_limit`
: An integer specifying the maximum number of consecutive iterations with no change in gene pool size before early termination. Prevents infinite loops when target size cannot be reached. Default is 5.

`allow_parallel`
: A logical indicating whether to enable parallel processing for internal resampling and calibration. The parallel backend must be registered by the user. Default is `TRUE`.

`blas_threads`
: An integer specifying the number of BLAS threads to use during computation. Default is 1, which avoids multi-threaded OpenBLAS overhead on Linux. Requires **RhpcBLASctl**. See Details.

Details

Retention strategies:

When `retain_by = "probability"` (default), observations with errors below a percentile threshold are retained. The percentile is computed using `quantile` with `probs` set to the `retain` value from `gesearch`. This approach is more robust when outlier observations have extreme error values.

When `retain_by = "proportion"`, a fixed fraction of observations (specified by the `retain` argument in `gesearch`) with the lowest associated errors are kept at each iteration.

Cross-validation for tuning:

When `tune = TRUE`, leave-group-out cross-validation is used to select optimal regression parameters at each iteration. The `number` argument controls how many CV groups are formed, and `p` controls the proportion of observations in each group.

BLAS threading:

On Linux systems with multi-threaded OpenBLAS, the default thread count can cause significant overhead for algorithms that perform many small matrix operations (like the iterative PLS fits in `gesearch`). Setting `blas_threads = 1` (the default) eliminates this overhead.

This setting requires the **RhpcBLASctl** package. If not installed, the parameter is ignored and a message is displayed. The original thread count is restored when `gesearch` completes.

Value

A list of class "gesearch_control" containing the specified parameters.

Author(s)

Leonardo Ramirez-Lopez

References

Lobsey, C.R., Viscarra Rossel, R.A., Roudier, P., Hedley, C.B. 2017. rs-local data-mines information from spectral libraries to improve local calibrations. European Journal of Soil Science 68:840-852.

See Also

gesearch, quantile

Examples

```
# Default parameters (probability-based retention)
gesearch_control()

# Proportion-based retention
gesearch_control(retain_by = "proportion")

# Enable parameter tuning with custom CV settings
gesearch_control(tune = TRUE, number = 5, p = 0.8)
```

---

liblex *Build a precomputed library of localised experts using memory-based learning*

Description

Constructs a library of local predictive models based on memory-based learning (MBL). For each anchor observation, a local regression model is fitted using its nearest neighbors from the reference set. This implementation is based on the methods proposed in Ramirez-Lopez et al. (2026b).

Usage

```
liblex(Xr, Yr, neighbors,
       diss_method = diss_pca(ncomp = ncomp_by_opc()),
       fit_method = fit_wapls(min_ncomp = 3, max_ncomp = 15),
       anchor_indices = NULL, gh = TRUE, group = NULL,
       control = liblex_control(), verbose = TRUE, ...)

## S3 method for class 'liblex'
```

```
predict(object, newdata, diss_method = NULL,
        weighting = c("gaussian", "tricube", "triweight", "triangular",
                      "quartic", "parabolic", "cauchy", "none"),
        adaptive_bandwidth = TRUE, reliability_weighting = TRUE,
        range_prediction_limits = FALSE, residual_cutoff = NULL,
        enforce_indices = NULL, probs = c(0.05, 0.25, 0.5, 0.75, 0.95),
        verbose = TRUE, allow_parallel = TRUE, blas_threads = 1L, ...)

## S3 method for class 'liblex'
plot(x, ...)
```

Arguments

`Xr`	A numeric matrix of predictor variables with dimensions `n × p` (observations in rows, variables in columns). Column names are required.
`Yr`	A numeric vector or single-column matrix of length `n` containing the response variable values corresponding to `Xr`. Missing values are allowed (see Details).
`neighbors`	A neighbor selection object specifying how to select neighbors. Use `neighbors_k()` for fixed k-nearest neighbors or `neighbors_diss()` for dissimilarity threshold-based selection.
`diss_method`	For `liblex`: either a `diss_*` object specifying the dissimilarity method, or a precomputed numeric dissimilarity matrix. If a `diss_*` object (e.g., `diss_pca()`, `diss_pls()`, `diss_correlation()`, `diss_euclidean()`), dissimilarities are computed internally. Additional parameters (centering, scaling, number of components) are controlled within the object. If a matrix is provided: • When `anchor_indices = NULL`: must be square (`n × n`) with zeros on the diagonal. • When `anchor_indices` is specified (length `m`): must have dimensions `n × m`, with `diss_method[anchor_indices, ]` having zeros on the diagonal. Default is `diss_pca(ncomp = ncomp_by_opc())`. For `predict.liblex`: a dissimilarity method object created by `diss_*()` constructors. If not provided, uses the method stored in `object`. Required if `object` was built with a precomputed dissimilarity matrix.
`fit_method`	A `local_fit` object specifying the local regression method. Currently supported: `fit_pls()` and `fit_wapls()`. `fit_gpr()` is not yet supported. Default is `fit_wapls(min_ncomp = 3, max_ncomp = 15)`.
`anchor_indices`	An optional integer vector specifying row indices of `Xr` around which to build local models. If `NULL` (default), models are built for all observations. See Details.
`gh`	Logical indicating whether to compute the GH distance (Mahalanobis distance in PLS score space) for each anchor observation. Default is `TRUE`.
`group`	An optional factor assigning group labels to observations in `Xr`. Used for leave-group-out validation to avoid pseudo-replication. When an observation is selected for validation, all observations from the same group are excluded from model fitting. Default is `NULL` (each observation is its own group).
`control`	A list of control parameters created by `liblex_control()`. Default is `liblex_control()`.

verbose — Logical indicating whether to display progress messages. Default is TRUE.

... — Additional arguments (currently unused).

object — A fitted object of class "liblex", created by liblex (for predict).

newdata — A numeric matrix or data frame containing new predictor values. Must include all predictors used in object.

weighting — Character string specifying the kernel weighting function applied to neighbours when combining predictions. Options are: "gaussian" (default), "tricube", "triweight", "triangular", "quartic", "parabolic", "cauchy", or "none" (equal weights). See Details for kernel definitions.

adaptive_bandwidth — Logical indicating whether to use adaptive bandwidth for kernel weighting. When TRUE (default), the bandwidth is set to the maximum dissimilarity within the neighborhood of each observation, so weights adapt to local density. When FALSE, a fixed global bandwidth is used across all predictions, which may result in uniform weights in sparse regions or overly concentrated weights in dense regions.

reliability_weighting — Logical indicating whether to weight expert predictions by their estimated reliability. When TRUE (default), the contribution of each experrt is additionally weighted by the inverse of its cross-validation residual variance, giving more influence to models that performed well during fitting. When FALSE, only dissimilarity-based kernel weights are used.

probs — A numeric vector of probabilities in $[0, 1]$ for computing weighted quantiles of expert predictions. Default is c(0.05, 0.25, 0.5, 0.75, 0.95).

range_prediction_limits — Logical. If TRUE, predictions falling outside the 5th–95th percentile range of neighbour response values are clipped to those limits. Default is FALSE.

residual_cutoff — Numeric threshold for excluding models. Models with absolute residuals exceeding this value are penalized during neighbour selection. Default is NULL (no exclusion).

enforce_indices — Optional integer vector specifying model indices that must always be included in each prediction neighborhood. These models are assigned the minimum dissimilarity of the neighborhood to ensure selection. Default is NULL (no enforced models).

allow_parallel — Logical indicating whether parallel computation is permitted if a backend is registered. Default is TRUE.

blas_threads — Integer specifying the number of BLAS threads to use. Default is 1L. Requires the **RhpcBLASctl** package for thread control.

x — An object of class "liblex" as returned by liblex().

Details

By default, local models are constructed for all n observations in the reference set. Alternatively, specify a subset of m observations (m < n) via anchor_indices to reduce computation.

Each local model uses neighbors selected from the full reference set, but models are only built for anchor observations. This is useful for large datasets where building models for all observations is computationally prohibitive.

When dissimilarity methods depend on `Yr` (e.g., PLS-based distances), the response values of anchor observations are excluded during dissimilarity computation for efficiency. However, anchor response values are always used when fitting local models.

The number of anchors must not exceed 90% of `nrow(Xr)`; to build models for all observations, use `anchor_indices = NULL`.

Relationship between anchors and neighborhood size:

The `neighbors` argument controls the neighborhood size (`k`) used both for fitting local models and for retrieving experts during prediction. When `anchor_indices` is specified, the number of available experts equals the number of anchors. If `max(k)` exceeds the number of anchors and tuning selects a large optimal `k`, prediction will retrieve fewer experts than specified. For reliable predictions, ensure the number of anchors is at least as large as the maximum `k` value being evaluated.

Missing values in Yr:

Missing values in `Yr` are permitted. Observations with missing response values can still serve as neighbors but are excluded from model fitting as target observations.

GH distance:

The GH distance is computed independently from `diss_method` using a PLS projection with optimized component selection. This provides a measure of how far each observation lies from the center of the reference set in the PLS score space.

Validation and tuning:

When `control$mode = "validate"` or `control$tune = TRUE`, nearest-neighbor cross-validation is performed. For each anchor observation, its nearest neighbor is excluded, a model is fitted on remaining neighbors, and the excluded neighbor's response is predicted. This provides validation statistics for parameter selection.

Prediction:

For each observation in `newdata`, the `predict` method:

1. Computes dissimilarities to anchor observations (or their neighbourhood centres) stored in `object`.
2. Selects the `k` nearest neighbours based on the optimal `k` determined during model fitting.
3. Applies kernel weighting based on dissimilarity.
4. Combines expert predictions using weighted averaging.

Kernel weighting functions:

The weighting functions follow Cleveland and Devlin (1988). Let d be the normalised dissimilarity (scaled to $[0, 1]$ within the neighbourhood when `adaptive_bandwidth = TRUE`). The available kernels are:

- `"gaussian"`: $w = \exp(-d^2)$
- `"tricube"`: $w = (1 - d^3)^3$
- `"triweight"`: $w = (1 - d^2)^3$

- "triangular": $w = 1 - d$
- "quartic": $w = (1 - d^2)^2$
- "parabolic": $w = 1 - d^2$
- "cauchy": $w = 1/(1 + d^2)$
- "none": $w = 1$ (equal weights)

Value

For liblex**:** A list of class "liblex" (when control$mode = "build") or "liblex_validation" (when control$mode = "validate") containing:

- dissimilarity: List containing the dissimilarity method and matrix.
- fit_method: Fit constructor from fit_method.
- gh: If gh = TRUE, a list with GH distances and the PLS projection.
- results: Data frame of validation statistics for each parameter combination (if validation was performed).
- best: The optimal parameter combination based on control$metric.
- optimal_params: List with optimal k and ncomp values.
- residuals: Residuals from predictions using optimal parameters.
- coefficients: (Build mode only) List of regression coefficients: B0 (intercepts), B (slopes).
- vips: (Build mode only) Variable importance in projection scores.
- selectivity_ratios: (Build mode only) Selectivity ratios for each predictor.
- scaling: (Build mode only) Centering and scaling vectors for prediction.
- neighborhood_stats: Statistics (response quantiles) for each neighborhood size.
- anchor_indices: The anchor indices used.
- neighbors: The object passed to neighbors.

For predict.liblex**:** A list with the following components:

- predictions: A data frame containing:
  - pred: Weighted mean predictions.
  - pred_sd: Weighted standard deviation of expert predictions.
  - q*: Weighted quantiles at probabilities specified by probs.
  - gh: Global Mahalanobis distance (if computed during fitting).
  - min_yr: Minimum response value (5th percentile) across neighbours.
  - max_yr: Maximum response value (95th percentile) across neighbours.
  - below_min: Logical indicating prediction below min_yr.
  - above_max: Logical indicating prediction above max_yr.
- neighbors: A list with:
  - indices: Matrix of neighbour indices (models) for each observation.
  - dissimilarities: Matrix of corresponding dissimilarity scores.
- expert_predictions: A list with:
  - weights: Matrix of kernel weights applied to each expert.
  - predictions: Matrix of raw predictions from each expert.
  - weighted: Matrix of weighted predictions from each expert.

Author(s)

Leonardo Ramirez-Lopez

References

Cleveland, W. S., & Devlin, S. J. (1988). Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403), 596–610.

Naes, T., Isaksson, T., & Kowalski, B. (1990). Locally weighted regression and scatter correction for near-infrared reflectance data. *Analytical Chemistry*, 62(7), 664–673.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. (2013). The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex datasets. Geoderma, 195-196, 268-279.

Ramirez-Lopez, L., Metz, M., Lesnoff, M., Orellano, C., Perez-Fernandez, E., Plans, M., Breure, T., Behrens, T., Viscarra Rossel, R., & Peng, Y. (2026b). Rethinking local spectral modelling: From per-query refitting to model libraries. *Analytica Chimica Acta*, under review.

Rajalahti, T., Arneberg, R., Berven, F.S., Myhr, K.M., Ulvik, R.J., Kvalheim, O.M. (2009). Biomarker discovery in mass spectral profiles by means of selectivity ratio plot. Chemometrics and Intelligent Laboratory Systems, 95(1), 35-48.

See Also

`liblex_control()` for control parameters, `neighbors_k()` for neighborhood specification, `diss_pca()`, `diss_pls()`, `diss_correlation()` for dissimilarity methods, `fit_pls()`, `fit_wapls()` for fitting methods.

Examples

```
## Not run:
library(prospectr)
data(NIRsoil)

# Preprocess spectra
NIRsoil$spc_pr <- savitzkyGolay(
  detrend(NIRsoil$spc, wav = as.numeric(colnames(NIRsoil$spc))),
  m = 1, p = 1, w = 7
)

# Missing values in the response are allowed
train_x <- NIRsoil$spc_pr[NIRsoil$train == 1, ]
train_y <- NIRsoil$Ciso[NIRsoil$train == 1]
test_x  <- NIRsoil$spc_pr[NIRsoil$train == 0, ]
test_y  <- NIRsoil$Ciso[NIRsoil$train == 0]

# Build library
model_library <- liblex(
  Xr = train_x,
  Yr = train_y,
  neighbors = neighbors_k(c(30, 40)),
  diss_method = diss_correlation(ws = 27, scale = TRUE),
```

```
  fit_method = fit_wapls(
    min_ncomp = 4,
    max_ncomp = 17,
    scale = FALSE,
    method = "mpls"
  ),
  control = liblex_control(tune = TRUE)
)

# Visualise neighborhood centroids and samples to predict
matplot(
  as.numeric(colnames(model_library$scaling$local_x_center)),
  t(test_x),
  col = rgb(1, 0, 0, 0.3),
  lty = 1,
  type = "l",
  xlab = "Wavelength (nm)",
  ylab = "First derivative detrended absorbance"
)
matlines(
  as.numeric(colnames(model_library$scaling$local_x_center)),
  t(model_library$scaling$local_x_center),
  col = rgb(0, 0, 1, 0.3),
  lty = 1,
  type = "l"
)
grid(lty = 1)
legend(
  "topright",
  legend = c("Samples to predict", "Neighborhood centroids"),
  col = c(rgb(1, 0, 0, 0.8), rgb(0, 0, 1, 0.8)),
  lty = 1,
  lwd = 2,
  bty = "n"
)

# Predict new observations
y_hat_liblex <- predict(model_library, test_x)

# Predicted versus observed values
lims <- range(y_hat_liblex$predictions$pred, test_y, na.rm = TRUE)
plot(
  y_hat_liblex$predictions$pred,
  test_y,
  pch = 16,
  col = rgb(0, 0, 0, 0.5),
  xlab = "Predicted",
  ylab = "Observed",
  xlim = lims,
  ylim = lims
)
abline(a = 0, b = 1, col = "red")
grid(lty = 1)
```

```
## run liblex in parallel (requires a parallel backend, e.g., doParallel)
library(doParallel)
n_cores <- min(2, parallel::detectCores() - 1)
clust <- makeCluster(n_cores)
registerDoParallel(clust)

model_library2 <- liblex(
  Xr = train_x,
  Yr = train_y,
  neighbors = neighbors_k(c(30, 40)),
  fit_method = fit_wapls(min_ncomp = 4, max_ncomp = 17, method = "simpls")
)

y_hat_liblex2 <- predict(model_library2, test_x)
registerDoSEQ()
try(stopCluster(clust))

## End(Not run)
```

`liblex_control` *Control parameters for liblex*

Description

Specifies control parameters for the `liblex` function, including output options, validation settings, tuning behavior, and parallel execution.

Usage

```
liblex_control(
  return_dissimilarity = FALSE,
  mode = c("build", "validate"),
  tune = FALSE,
  metric = c("rmse", "r2"),
  chunk_size = 1L,
  allow_parallel = TRUE,
  blas_threads = 1L
)
```

Arguments

`return_dissimilarity`
: A logical indicating whether the dissimilarity matrix should be returned in the output. Default is `FALSE`. Setting to `TRUE` can be useful for diagnostics but increases memory usage for large libraries.

`mode`
: A character string specifying the operation mode:

"build" (default) Builds the library of local models. If tune = TRUE, validation is performed first to find optimal parameters, then the library is built using those parameters. If tune = FALSE, the library is built directly using the parameters provided to liblex.

"validate" Performs validation only without building the library. Useful for parameter exploration or when testing different configurations before committing to the full library build.

tune — A logical indicating whether to optimize parameters via nearest-neighbor validation. Default is FALSE. When TRUE, the function evaluates all combinations of k values and PLS component ranges, selecting the combination that minimizes RMSE (or maximizes R^2, depending on metric). See Details.

metric — A character string specifying the performance metric used for parameter selection when tune = TRUE. Options are:

"rmse" (default) Root mean squared error (minimized).

"r2" Coefficient of determination (maximized).

Ignored when tune = FALSE.

chunk_size — An integer specifying the number of local models to process per parallel task. Default is 1L. Increasing this value reduces parallel overhead but may cause load imbalance. For large libraries, values between 10 and 50 often provide a good trade-off between overhead and efficiency.

allow_parallel — A logical indicating whether parallel execution is permitted. Default is TRUE. Parallelization is applied to the model fitting loop using the **foreach** package. Requires a registered parallel backend (e.g., via **doParallel**).

blas_threads — An integer specifying the number of threads for BLAS operations. Default is 1L, which avoids thread contention when using parallel processing. Requires the **RhpcBLASctl** package to take effect. On Linux systems with multi-threaded BLAS (e.g., OpenBLAS), setting this to 1 can substantially improve performance when allow_parallel = TRUE.

Details

Nearest-neighbor validation: When tune = TRUE or mode = "validate", the function performs nearest-neighbor validation (NNv) to assess model performance. For each observation in the reference set (or anchor set, if specified), the procedure:

1. Identifies the k nearest neighbors of the target observation.
2. Excludes the target observation (and any observations in the same group, if group is specified) from the neighbor set.
3. Fits a local model using the remaining neighbors.
4. Predicts the response value for the excluded target observation.
5. Computes prediction errors across all observations.

This leave-one-out style validation provides an estimate of prediction performance without requiring a separate test set. When tune = TRUE, the parameter combination (number of neighbors and PLS component range) yielding the best performance according to metric is selected for building the final library.

Mode and tune combinations:

mode	tune	Behavior
"build"	FALSE	Build library using parameters as provided
"build"	TRUE	Validate, find optimal parameters, build library
"validate"	FALSE	Validate only, report performance statistics
"validate"	TRUE	Validate, report statistics with optimal parameters identified

Parallel chunk size: The chunk_size parameter controls granularity of parallel work distribution. When allow_parallel = TRUE and a parallel backend is registered:

- chunk_size = 1: Each local model is a separate parallel task. Maximum parallelism but higher scheduling overhead.
- chunk_size > 1: Multiple models are processed sequentially within each parallel task. Reduces overhead and improves memory locality, but may cause load imbalance if the number of models is not evenly divisible.

When allow_parallel = FALSE, chunk_size has no effect.

Value

A list of class liblex_control containing the validated control parameters.

Author(s)

Leonardo Ramirez-Lopez

See Also

liblex, mbl_control

Examples

```
# Default settings: build library without tuning
liblex_control()

# Tune parameters before building
liblex_control(tune = TRUE)

# Validate only for parameter exploration
liblex_control(mode = "validate", tune = TRUE)

# Include dissimilarity matrix in output
liblex_control(return_dissimilarity = TRUE)

# Larger chunks for reduced parallel overhead
liblex_control(chunk_size = 20L)

# Parallel settings for Linux with OpenBLAS
liblex_control(allow_parallel = TRUE, blas_threads = 1L)
```

mbl *Memory-based learning (mbl)*

Description

Memory-based learning (a.k.a. instance-based learning or local regression) is a non-linear lazy learning approach for predicting a response variable from predictor variables. For each observation in a prediction set, a local regression is fitted using a subset of similar observations (nearest neighbors) from a reference set. This function does not produce a global model.

Usage

```
mbl(Xr, Yr, Xu, Yu = NULL,
    neighbors,
    diss_method = diss_pca(ncomp = ncomp_by_opc()),
    diss_usage = c("none", "predictors", "weights"),
    fit_method = fit_wapls(min_ncomp = 3, max_ncomp = 15),
    spike = NULL, group = NULL,
    gh = FALSE,
    control = mbl_control(),
    verbose = TRUE, seed = NULL, ...)

## S3 method for class 'mbl'
plot(x, what = c("validation", "gh"), metric = "rmse", ncomp = c(1, 2), ...)

get_predictions(x)

## S3 method for class 'mbl'
plot(x, what = c("validation", "gh"), metric = "rmse", ncomp = c(1, 2), ...)
```

Arguments

Xr	A matrix of predictor variables for the reference data (observations in rows, variables in columns). Column names are required.
Yr	A numeric vector or single-column matrix of response values corresponding to Xr. NA values are not permitted.
Xu	A matrix of predictor variables for the data to be predicted (observations in rows, variables in columns). Must have the same column names as Xr.
Yu	An optional numeric vector or single-column matrix of response values corresponding to Xu. Used for computing prediction statistics. Default is NULL.
neighbors	A neighbor selection object specifying how to select neighbors. Use neighbors_k() for fixed k-nearest neighbors or neighbors_diss() for dissimilarity threshold-based selection.
diss_method	A dissimilarity method object or a precomputed dissimilarity matrix. Available constructors:

- `diss_pca()`: Mahalanobis distance in PCA score space. This is the default where the number of components is optimized using side information (see `ncomp_by_opc()`).
- `diss_pls()`: Mahalanobis distance in PLS score space
- `diss_euclidean()`: Euclidean distance
- `diss_mahalanobis()`: Mahalanobis distance
- `diss_cosine()`: Cosine dissimilarity
- `diss_correlation()`: Correlation-based dissimilarity

A precomputed matrix can also be passed. When `diss_usage = "predictors"`, it must be square with dimensions `(nrow(Xr) + nrow(Xu))` and zeros on the diagonal. Otherwise, it must have `nrow(Xr)` rows and `nrow(Xu)` columns.

`diss_usage` How dissimilarity information is used in local models:

- `"none"` (default): dissimilarities used only for neighbor selection
- `"predictors"`: local dissimilarity matrix columns added as predictors
- `"weights"`: neighbors weighted by dissimilarity using a tricubic function

`fit_method` A local fitting method object. Available constructors:

- `fit_pls()`: Partial least squares regression
- `fit_wapls()`: Weighted average PLS (default)
- `fit_gpr()`: Gaussian process regression

`spike` An integer vector indicating indices of observations in `Xr` to force into (positive values) or exclude from (negative values) all neighborhoods. Default is `NULL`. Spiking does not change neighborhood size; forced observations displace the most distant neighbors.

`group` An optional factor assigning group labels to `Xr` observations (e.g., measurement batches). Used to avoid pseudo-replication in cross-validation: when one observation is held out, all observations from its group are also removed.

`gh` Logical indicating whether to compute global Mahalanobis (GH) distances. Default is `FALSE`. GH distances measure how far each observation lies from the center of the reference set in PLS score space. The computation uses a fixed methodology: PLS projection with the number of components selected via `ncomp_by_opc()` (capped at 40). This is independent of the `diss_method` argument.

`control` A list from `mbl_control()` specifying validation type, tuning options, and other settings.

`verbose` Logical indicating whether to display a progress bar. Default is `TRUE`. Not shown during parallel execution.

`seed` An integer for random number generation, enabling reproducible cross-validation results. Default is `NULL`.

`...` Additional arguments (currently unused).

`x` An object of class `mbl` (as returned by `mbl`).

`what` Character vector specifying what to plot. Options are `"validation"` (validation statistics) and/or `"gh"` (PLS scores used for GH distance computation). Default is both.

metric	Character string specifying which validation statistic to plot. Options are "rmse", "st_rmse", or "r2". Only used when "validation" is in what.
ncomp	Integer vector of length 1 or 2 specifying which PLS components to plot. Default is c(1, 2). Only used when "gh" is in what.

Details

Spiking:

The spike argument forces specific reference observations into or out of neighborhoods. Positive indices are always included; negative indices are always excluded. When observations are forced in, the most distant neighbors are displaced to maintain neighborhood size. See Guerrero et al. (2010).

Dissimilarity usage:

When diss_usage = "predictors", the local dissimilarity matrix columns are appended as additional predictor variables, which can improve predictions (Ramirez-Lopez et al., 2013a).

When diss_usage = "weights", neighbors are weighted using a tricubic function (Cleveland and Devlin, 1988; Naes et al., 1990):

$$W_j = (1 - v^3)^3$$

where $v = d(xr_i, xu_j) / \max(d)$.

GH distance:

The global Mahalanobis distance (GH) measures how far each observation lies from the center of the reference set. It is always computed using a PLS projection with the number of components optimized via ncomp_by_opc() (maximum 40 components or nrow(Xr), whichever is smaller). This methodology is fixed and independent of the diss_method specified for neighbor selection.

GH distances are useful for identifying extrapolation: observations with high GH values lie far from the calibration space and may yield unreliable predictions.

Grouping:

The group argument enables leave-group-out cross-validation. When validation_type = "local_cv" in mbl_control(), the p parameter refers to the proportion of groups (not observations) retained per iteration.

Deprecated arguments:

The following arguments from previous versions of resemble are no longer supported and will throw an error if used: k, k_diss, k_range, method, pc_selection, center, scale, and documentation. See the current argument list for their replacements.

Value

mbl:

For mbl(), a list of class mbl containing:

- control: control parameters from control
- fit_method: fit constructor from fit_method
- Xu_neighbors: list with neighbor indices and dissimilarities

- dissimilarities: dissimilarity method and matrix (if return_dissimilarity = TRUE in control)
- n_predictions: number of predictions made
- gh: GH distances for Xr and Xu (if gh = TRUE)
- validation_results: validation statistics by method
- results: list of data.frame objects with predictions, one per neighborhood size
- seed: the seed value used

Each results table contains:

- o_index: observation index
- k: number of neighbors used
- k_diss, k_original: (neighbors_diss only) threshold and original count
- ncomp: (fit_pls only) number of PLS components
- min_ncomp, max_ncomp: (fit_wapls only) component range
- yu_obs, pred: observed and predicted values
- yr_min_obs, yr_max_obs: response range in neighborhood
- index_nearest_in_Xr, index_farthest_in_Xr: neighbor indices
- y_nearest, y_farthest: neighbor response values
- diss_nearest, diss_farthest: neighbor dissimilarities
- y_nearest_pred: (NNv validation) leave-one-out prediction
- loc_rmse_cv, loc_st_rmse_cv: (local_cv validation) CV statistics
- loc_ncomp: (local dissimilarity only) components used locally

Get predictions:

The get_predictions() function extracts predicted values from an object of class mbl. It returns a data.frame containing the predictions.

Author(s)

Leonardo Ramirez-Lopez and Antoine Stevens

References

Cleveland, W. S., and Devlin, S. J. 1988. Locally weighted regression: an approach to regression analysis by local fitting. Journal of the American Statistical Association 83:596-610.

Guerrero, C., Zornoza, R., Gomez, I., Mataix-Beneyto, J. 2010. Spiking of NIR regional models using observations from target sites: Effect of model size on prediction accuracy. Geoderma 158:66-77.

Naes, T., Isaksson, T., Kowalski, B. 1990. Locally weighted regression and scatter correction for near-infrared reflectance data. Analytical Chemistry 62:664-673.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. 2013a. The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex data sets. Geoderma 195-196:268-279.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Viscarra Rossel, R., Dematte, J.A.M., Scholten, T. 2013b. Distance and similarity-search metrics for use with soil vis-NIR spectra. Geoderma 199:43-53.

Rasmussen, C.E., Williams, C.K. 2006. Gaussian Processes for Machine Learning. MIT Press.

Shenk, J., Westerhaus, M., Berzaghi, P. 1997. Investigation of a LOCAL calibration procedure for near infrared instruments. Journal of Near Infrared Spectroscopy 5:223-232.

See Also

mbl_control, neighbors_k, neighbors_diss, diss_pca, diss_pls, fit_pls, fit_wapls, fit_gpr, search_neighbors

Examples

```
## Not run:
library(prospectr)
data(NIRsoil)

# Preprocess: detrend + first derivative with Savitzky-Golay
sg_det <- savitzkyGolay(
  detrend(NIRsoil$spc, wav = as.numeric(colnames(NIRsoil$spc))),
  m = 1, p = 1, w = 7
)
NIRsoil$spc_pr <- sg_det

# Split data
test_x <- NIRsoil$spc_pr[NIRsoil$train == 0 & !is.na(NIRsoil$CEC), ]
test_y <- NIRsoil$CEC[NIRsoil$train == 0 & !is.na(NIRsoil$CEC)]
train_x <- NIRsoil$spc_pr[NIRsoil$train == 1 & !is.na(NIRsoil$CEC), ]
train_y <- NIRsoil$CEC[NIRsoil$train == 1 & !is.na(NIRsoil$CEC)]

# Example 1: Spectrum-based learner (Ramirez-Lopez et al., 2013)
ctrl <- mbl_control(validation_type = "NNv")

sbl <- mbl(
  Xr = train_x,
  Yr = train_y,
  Xu = test_x,
  neighbors = neighbors_k(seq(40, 140, by = 20)),
  diss_method = diss_pca(ncomp = ncomp_by_opc(40)),
  fit_method = fit_gpr(),
  control = ctrl
)
sbl
plot(sbl)
get_predictions(sbl)

# Example 2: With known Yu
sbl_2 <- mbl(
  Xr = train_x,
  Yr = train_y,
  Xu = test_x,
  Yu = test_y,
  neighbors = neighbors_k(seq(40, 140, by = 20)),
  fit_method = fit_gpr(),
```

```
  control = ctrl
)
plot(sbl_2)

# Example 3: LOCAL algorithm (Shenk et al., 1997)
local_algo <- mbl(
  Xr = train_x,
  Yr = train_y,
  Xu = test_x,
  Yu = test_y,
  neighbors = neighbors_k(seq(40, 140, by = 20)),
  diss_method = diss_correlation(),
  diss_usage = "none",
  fit_method = fit_wapls(min_ncomp = 3, max_ncomp = 15),
  control = ctrl
)
plot(local_algo)

# Example 4: Using dissimilarity as predictors
local_algo_2 <- mbl(
  Xr = train_x,
  Yr = train_y,
  Xu = test_x,
  Yu = test_y,
  neighbors = neighbors_k(seq(40, 140, by = 20)),
  diss_method = diss_pca(ncomp = ncomp_by_opc(40)),
  diss_usage = "predictors",
  fit_method = fit_wapls(min_ncomp = 3, max_ncomp = 15),
  control = ctrl
)
plot(local_algo_2)

# Example 5: Parallel execution
library(doParallel)
n_cores <- min(2, parallel::detectCores() - 1)
clust <- makeCluster(n_cores)
registerDoParallel(clust)

local_algo_par <- mbl(
  Xr = train_x,
  Yr = train_y,
  Xu = test_x,
  Yu = test_y,
  neighbors = neighbors_k(seq(40, 140, by = 20)),
  diss_method = diss_correlation(),
  fit_method = fit_wapls(min_ncomp = 3, max_ncomp = 15),
  control = ctrl
)

registerDoSEQ()
try(stopCluster(clust))

## End(Not run)
```

mbl_control *Control parameters for memory-based learning*

Description

This function controls various aspects of the memory-based learning process in the mbl function.

Usage

```
mbl_control(
  return_dissimilarity = FALSE,
  validation_type = "NNv",
  tune_locally = TRUE,
  number = 10,
  p = 0.75,
  range_prediction_limits = TRUE,
  allow_parallel = TRUE,
  blas_threads = 1L
)
```

Arguments

return_dissimilarity
: Logical indicating whether to return the dissimilarity matrix between Xr and Xu. Default is FALSE.

validation_type
: Character vector specifying validation method(s):
  - "NNv": Leave-nearest-neighbor-out cross-validation (default, faster)
  - "local_cv": Local leave-group-out cross-validation
  - "none": No validation

  Multiple methods can be specified (e.g., c("NNv", "local_cv")). Default is "NNv".

tune_locally
: Logical indicating whether to tune PLS components locally when validation_type = "local_cv" and using fit_pls or fit_wapls. Default is TRUE.

number
: Integer specifying the number of sampling iterations for "local_cv" validation. Default is 10.

p
: Numeric value between 0 and 1 indicating the proportion of observations retained at each "local_cv" iteration. Default is 0.75.

range_prediction_limits
: Logical indicating whether predictions should be constrained to the range of response values in each neighborhood. Default is TRUE.

allow_parallel
: Logical indicating whether parallel execution is allowed via the **foreach** package. Default is TRUE.

`blas_threads` Integer specifying the number of BLAS threads to use during `mbl()` execution. Default is `1L`, which avoids thread overhead from repeated small matrix operations. Requires the **RhpcBLASctl** package to take effect. The original thread count is restored after `mbl()` completes. See Details.

Details

Validation methods:

Leave-nearest-neighbor-out cross-validation (`"NNv"`): For each target observation, the nearest neighbor is excluded from the local model, which then predicts that neighbor's value. This is faster than `"local_cv"`. If the nearest neighbor belongs to a group (specified via the `group` argument in `mbl`), all group members are excluded.

Local leave-group-out cross-validation (`"local_cv"`): The neighborhood is partitioned into subsets via stratified random sampling. Each subset serves as validation data while the remainder fits the model. This repeats `number` times, with `p` controlling the training proportion. The final error is the average local RMSE.

BLAS threading:

On Linux systems with multi-threaded OpenBLAS, the default thread count can cause significant overhead for algorithms like `mbl()` that perform many small matrix operations. Setting `blas_threads = 1` (the default) eliminates this overhead.

This setting requires the **RhpcBLASctl** package. If not installed, the parameter is ignored and a message is displayed. The original thread count is restored when `mbl()` completes.

Windows systems typically use single-threaded BLAS by default, so this setting has no effect there.

Value

A list of class `mbl_control` with the specified control parameters.

Author(s)

Leonardo Ramirez-Lopez and Antoine Stevens

References

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. 2013a. The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex data sets. Geoderma 195-196:268-279.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Viscarra Rossel, R., Dematte, J.A.M., Scholten, T. 2013b. Distance and similarity-search metrics for use with soil vis-NIR spectra. Geoderma 199:43-53.

See Also

`mbl`, `neighbors_k`, `neighbors_diss`

Examples

```
# Default control parameters (NNv validation)
mbl_control()

# Both validation methods
mbl_control(validation_type = c("NNv", "local_cv"))

# No validation
mbl_control(validation_type = "none")

# NNv validation only, no parallel
mbl_control(validation_type = "NNv", allow_parallel = FALSE)

# Allow more BLAS threads (if needed for other computations)
mbl_control(blas_threads = 4)
```

---

model *Global spectral calibration model*

Description

Fits a global calibration model for spectral data using partial least squares (PLS) or Gaussian process regression (GPR). Unlike mbl, which builds local models for each prediction, model() fits a single global model to the reference data.

Usage

```
model(Xr, Yr, fit_method, control = model_control(), verbose = TRUE)

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

Arguments

Xr	A numeric matrix of predictor variables, with observations in rows and variables in columns. Typically spectral data.
Yr	A numeric matrix with one column containing the response variable values corresponding to the observations in Xr.
fit_method	An object created by fit_pls or fit_gpr specifying the regression method and its parameters, including centring and scaling options. fit_wapls() is not supported for global models because it requires target observations to compute weights.
control	A list created by model_control() specifying the cross-validation settings. The default is model_control().
verbose	Logical indicating whether progress information should be printed. Default is TRUE.

object	A resemble_model object returned by model.
newdata	A numeric matrix of new observations with the same number of columns as the training data.
ncomp	For PLS models, the number of components to use for prediction. The default is the number used during fitting. Ignored for GPR models. For prediction, this can be a vector of integers representing the predictions coming from models with the requested components.
...	Additional arguments, currently unused.

Details

model() provides a straightforward interface for fitting global calibration models, in contrast to the local modelling approach implemented in mbl. This is useful when:

- the relationship between spectra and the response is approximately linear across the full dataset
- a single portable model is needed for deployment
- computational efficiency is prioritised over predictive performance in heterogeneous datasets

Cross-validation: When validation_type = "lgo", stratified random sampling is used to create training-validation splits that preserve the distribution of the response variable. Cross-validation results include RMSE, R^2, and standardised RMSE for each component in PLS models, or overall in GPR models.

PLS models: When fit_pls() is used, the function fits a PLS model with the specified number of components. If cross-validation is enabled, the optimal number of components is selected based on the minimum RMSE.

GPR models: When fit_gpr() is used, the function fits a Gaussian process regression model with a dot-product covariance function. The noise variance parameter controls regularisation.

Value

For model(), an object of class resemble_model containing:

- fit_method: Fitting method object used.
- control: Model control object used.
- model: Fitted model object.
- cv_results: Cross-validation results, if validation_type = "lgo".
- n_obs: Number of observations used.
- n_vars: Number of predictor variables.

For predict.resemble_model(), a numeric matrix of predictions. For PLS models, columns correspond to different numbers of components.

Author(s)

Leonardo Ramirez-Lopez

References

de Jong, S. (1993). SIMPLS: An alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, 18(3), 251–263.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Shenk, J. S., & Westerhaus, M. O. (1991). Population structuring of near infrared spectra and modified partial least squares regression. *Crop Science*, 31(6), 1548–1555.

See Also

`fit_pls` and `fit_gpr` for specifying the fitting method; `model_control` for controlling aspects of the modelling process; `mbl` for memory-based (local) learning.

Examples

```
library(prospectr)
data(NIRsoil)

# Preprocess spectra
Xr <- savitzkyGolay(NIRsoil$spc, m = 1, p = 2, w = 11)
Yr <- NIRsoil$CEC

# Remove missing values
ok <- !is.na(Yr)
Xr <- Xr[ok, ]
Yr <- as.matrix(Yr[ok])

# Fit a PLS model with 10 components and cross-validation
# Scaling is controlled via fit_pls()
pls_mod <- model(
  Xr = Xr,
  Yr = Yr,
  fit_method = fit_pls(ncomp = 10, scale = FALSE),
  control = model_control(validation_type = "lgo", number = 10)
)

# View cross-validation results
pls_mod$cv_results

# Fit a GPR model (centring/scaling controlled via fit_gpr())
gpr_mod <- model(
  Xr = Xr,
  Yr = Yr,
  fit_method = fit_gpr(noise_variance = 0.001, scale = TRUE),
  control = model_control(validation_type = "lgo")
)
```

model_control *Control parameters for global model fitting*

Description

Specifies cross-validation settings for the model function.

Usage

```
model_control(validation_type = c("lgo", "none"), number = 10L, p = 0.75)
```

Arguments

validation_type	a character string specifying the validation method: • "lgo": Leave-group-out cross-validation. At each iteration, a proportion p of observations is retained for training and the remainder is used for validation. This is repeated number times. • "none": No cross-validation is performed.
number	an integer indicating the number of cross-validation iterations. Only used when validation_type = "lgo". Default is 10.
p	a numeric value between 0 and 1 indicating the proportion of observations to retain for training at each cross-validation iteration. Only used when validation_type = "lgo". Default is 0.75.

Value

A list of class "model_control" containing the specified parameters.

Author(s)

Leonardo Ramirez-Lopez

See Also

model

Examples

```
# Default settings (leave-group-out CV with 10 iterations)
model_control()

# No cross-validation
model_control(validation_type = "none")

# Custom CV settings
model_control(validation_type = "lgo", number = 20, p = 0.80)
```

`ncomp_selection` *Component selection methods*

Description

Constructor functions for specifying how to select the number of components in projection-based dissimilarity methods (`diss_pca()`, `diss_pls()`).

Usage

```
ncomp_by_var(min_var = 0.01, max_ncomp = 40L)

ncomp_by_cumvar(min_cumvar = 0.99, max_ncomp = 40L)

ncomp_by_opc(max_ncomp = 40L)

ncomp_fixed(ncomp)
```

Arguments

`min_var`	Numeric in (0, 1]. Minimum variance a single component must explain to be retained.
`max_ncomp`	Positive integer. Maximum number of components to compute or evaluate.
`min_cumvar`	Numeric in (0, 1]. Minimum cumulative variance that the retained components must explain.
`ncomp`	Positive integer. Exact number of components to use.

Details

Four selection methods are available:

`ncomp_by_var()` Retains components that individually explain at least `min_var` proportion of variance.

`ncomp_by_cumvar()` Retains the minimum number of components whose combined explained variance reaches `min_cumvar`.

`ncomp_by_opc()` Optimized principal component selection based on side information (Ramirez-Lopez et al., 2013). The optimal number of components minimizes the RMSD between each observation's response and its nearest neighbor's response in the projected space. Requires `Yr`.

`ncomp_fixed()` Uses exactly `ncomp` components with no automatic selection. Equivalent to passing an integer directly.

At runtime, `max_ncomp` is capped at `min(max_ncomp, nrow(X), ncol(X))`.

Value

An object of class "ncomp_selection" with a subclass indicating the method:

- ncomp_by_var: class c("ncomp_by_var", "ncomp_selection")
- ncomp_by_cumvar: class c("ncomp_by_cumvar", "ncomp_selection")
- ncomp_by_opc: class c("ncomp_by_opc", "ncomp_selection")
- ncomp_fixed: class c("ncomp_fixed", "ncomp_selection")

Author(s)

Leonardo Ramirez-Lopez

References

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. 2013. The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex data sets. Geoderma 195-196, 268-279.

See Also

diss_pca(), diss_pls(), dissimilarity()

Examples

```
# Retain components explaining >= 1% variance each
ncomp_by_var(0.01)

# Retain enough components for 99% cumulative variance
ncomp_by_cumvar(0.99)

# Optimize using side information (requires Yr)
ncomp_by_opc(max_ncomp = 40)

# Fix at exactly 10 components
ncomp_fixed(10)

# Usage in dissimilarity constructors
diss_pca(ncomp = ncomp_by_var(0.01))
diss_pca(ncomp = ncomp_by_opc())
diss_pca(ncomp = 10)
```

neighbors *Neighbor selection methods*

Description

These functions create configuration objects that specify how neighbors are selected for memory-based learning in mbl an liblex.

Usage

```
neighbors_k(k)

neighbors_diss(threshold, k_min = 4L, k_max = Inf)
```

Arguments

k	Integer vector. One or more neighborhood sizes to evaluate. Values will be sorted in ascending order. Minimum allowed value is 4.
threshold	Numeric vector. One or more dissimilarity thresholds. Neighbors are selected if their dissimilarity to the target observation is below the threshold. Values will be sorted in ascending order.
k_min	Integer. Minimum number of neighbors to retain, regardless of threshold. Default 4L.
k_max	Integer or Inf. Maximum number of neighbors to retain, regardless of threshold. Default Inf (no upper bound other than the size of the reference set).

Details

Two strategies are available for neighbor selection:

Fixed-k selection (neighbors_k)

A fixed number of nearest neighbors is selected for each target observation. Multiple values of k can be provided to evaluate different neighborhood sizes.

Dissimilarity-threshold selection (neighbors_diss)

Neighbors are selected based on a dissimilarity threshold. All reference observations with dissimilarity below the threshold are included. The k_min and k_max arguments provide bounds to ensure a reasonable neighborhood size regardless of the threshold. Multiple thresholds can be provided to evaluate different settings.

Value

An object of class c("neighbors_k", "neighbors") or c("neighbors_diss", "neighbors"), containing the validated parameters. Intended to be passed to mbl.

See Also

mbl

Examples

```
# Fixed neighborhood sizes
neighbors_k(k = 50)
neighbors_k(k = c(40, 60, 80, 100, 120))

# Dissimilarity threshold with default bounds
neighbors_diss(threshold = 0.3)

# Dissimilarity threshold with custom bounds
neighbors_diss(threshold = c(0.1, 0.2, 0.3), k_min = 10, k_max = 150)
```

---

ortho_projection *Orthogonal projections*

Description

Performs orthogonal projections of high-dimensional data matrices using principal component analysis (PCA) or partial least squares (PLS).

Usage

```
ortho_projection(
  Xr, Xu = NULL, Yr = NULL,
  ncomp = ncomp_by_var(0.01),
  method = c("pca", "pca_nipals", "pls", "mpls", "simpls"),
  center = TRUE,
  scale = FALSE,
  tol = 1e-6,
  max_iter = 1000L,
  pc_selection = deprecated(),
  ...
)

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

## S3 method for class 'ortho_projection'
plot(x, col = "#3B82F6", ...)

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

Arguments

Xr	A numeric matrix of reference observations (rows) and variables (columns).
Xu	An optional matrix of additional observations to project.

Yr	An optional response matrix. Required for PLS methods ("pls", "mpls", "simpls") and when using ncomp_by_opc().
ncomp	Component selection method. Either: • A positive integer (equivalent to ncomp_fixed(n)) • An ncomp_selection object: ncomp_by_var(), ncomp_by_cumvar(), ncomp_by_opc(), or ncomp_fixed() Default is ncomp_by_var(0.01).
method	A character string specifying the projection method: • "pca": PCA via singular value decomposition (default) • "pca_nipals": PCA via NIPALS algorithm • "pls": PLS via NIPALS algorithm • "mpls": Modified PLS via NIPALS (Shenk and Westerhaus, 1991) • "simpls": PLS via SIMPLS algorithm (de Jong, 1993)
center	A logical indicating whether to center the data. Default is TRUE. PLS methods always center internally regardless of this setting.
scale	A logical indicating whether to scale the data to unit variance. Default is FALSE.
tol	Convergence tolerance for the NIPALS algorithm. Default is 1e-6. Ignored when method = "simpls".
max_iter	Maximum number of iterations for NIPALS. Default is 1000. Ignored when method = "simpls".
pc_selection	**[Deprecated]** Use ncomp instead.
...	Additional arguments (currently unused).
object	Object of class "ortho_projection".
newdata	Matrix of new observations to project.
x	An object of class ortho_projection (as returned by ortho_projection).
col	Color for the plot elements. Default is "#3B82F6".

Details

PCA methods:

When method = "pca", singular value decomposition factorizes the data matrix X as:

$$X = UDV^T$$

where U and V are orthogonal matrices (left and right singular vectors), and D is a diagonal matrix of singular values. The score matrix is UD and the loadings are V.

When method = "pca_nipals", the non-linear iterative partial least squares (NIPALS) algorithm is used instead.

PLS methods:

Three PLS variants are available:

- "pls": Standard PLS using the NIPALS algorithm with covariance-based weights.

- "mpls": Modified PLS using the NIPALS algorithm with correlation-based weights, giving equal influence to all predictors regardless of variance (Shenk and Westerhaus, 1991).
- "simpls": SIMPLS algorithm (de Jong, 1993), which deflates the cross-product matrix rather than X itself. Computationally faster than NIPALS, especially for wide matrices.

Component selection:

When ncomp_by_opc() is used, component selection minimizes RMSD (for continuous Yr) or maximizes kappa (for categorical Yr) between observations and their nearest neighbors. See diss_evaluate.

Value

An object of class "ortho_projection" containing:

- scores: Matrix of projected scores for Xr (and Xu).
- X_loadings: Matrix of X loadings.
- Y_loadings: Matrix of Y loadings (PLS only).
- weights: Matrix of PLS weights (PLS only).
- projection_mat: Projection matrix for new data (PLS only).
- variance: List with original and explained variance.
- scores_sd: Standard deviation of scores.
- ncomp: Number of components retained.
- center: Centering vector used.
- scale: Scaling vector used.
- method: Projection method used.
- ncomp_method: The value passed to the ncomp argument.
- opc_evaluation: opc optimization results (if applicable).

Author(s)

Leonardo Ramirez-Lopez

References

de Jong, S. 1993. SIMPLS: An alternative approach to partial least squares regression. Chemometrics and Intelligent Laboratory Systems 18:251-263.

Martens, H. 1991. Multivariate calibration. John Wiley & Sons.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. 2013a. The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex data sets. Geoderma 195-196:268-279.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Viscarra Rossel, R., Dematte, J.A.M., Scholten, T. 2013b. Distance and similarity-search metrics for use with soil vis-NIR spectra. Geoderma 199:43-53.

Shenk, J.S., Westerhaus, M.O. 1991. Populations structuring of near infrared spectra and modified partial least squares regression. Crop Science 31:1548-1555.

See Also

ncomp_by_var, ncomp_by_opc, diss_evaluate, mbl

Examples

```
library(prospectr)
data(NIRsoil)

# Preprocess
sg_det <- savitzkyGolay(
  detrend(NIRsoil$spc, wav = as.numeric(colnames(NIRsoil$spc))),
  m = 1, p = 1, w = 7
)

# Split data
train_x <- sg_det[NIRsoil$train == 1 & !is.na(NIRsoil$CEC), ]
train_y <- NIRsoil$CEC[NIRsoil$train == 1 & !is.na(NIRsoil$CEC)]
test_x <- sg_det[NIRsoil$train == 0 & !is.na(NIRsoil$CEC), ]

# PCA with fixed components
proj <- ortho_projection(train_x, ncomp = 5)

plot(proj)

# PCA with variance-based selection
proj <- ortho_projection(train_x, ncomp = ncomp_by_var(0.01))

# PCA with OPC optimization
proj <- ortho_projection(train_x, Xu = test_x, Yr = train_y,
                         ncomp = ncomp_by_opc(40))

#' plot(proj)

# PLS projection (NIPALS)
proj <- ortho_projection(train_x, Xu = test_x, Yr = train_y,
                         method = "pls", ncomp = ncomp_by_opc(40))

# Modified PLS
proj <- ortho_projection(train_x, Yr = train_y,
                         method = "mpls", ncomp = 10)

# SIMPLS (faster for wide matrices)
proj <- ortho_projection(train_x, Yr = train_y,
                         method = "simpls", ncomp = 10)
```

---

search_neighbors *Search neighbors in a reference set*

Description

Searches for the nearest neighbors of observations in a reference set or between two sets of observations.

Usage

```
search_neighbors(Xr, Xu = NULL,
                 diss_method = diss_pca(), Yr = NULL,
                 neighbors, spike = NULL,
                 return_dissimilarity = FALSE,
                 k, k_diss, k_range, pc_selection,
                 center, scale, documentation, ...
                 )
```

Arguments

`Xr`	A numeric matrix of reference observations (rows) and variables (columns) where the neighbor search is conducted.
`Xu`	Optional matrix of observations for which neighbors are to be searched in `Xr`.
`diss_method`	A dissimilarity method object created by one of: • `diss_pca()`: Mahalanobis distance in PCA space • `diss_pls()`: Mahalanobis distance in PLS space • `diss_correlation()`: Correlation-based dissimilarity • `diss_euclidean()`: Euclidean distance • `diss_mahalanobis()`: Mahalanobis distance • `diss_cosine()`: Cosine dissimilarity Default is `diss_pca()`.
`Yr`	Optional response matrix. Required for PLS methods and when using `ncomp_by_opc()`.
`neighbors`	A neighbor selection object created by: • `neighbors_k()`: Select k nearest neighbors • `neighbors_diss()`: Select neighbors by dissimilarity threshold
`spike`	Optional integer vector indicating observations in `Xr` to force into (positive indices) or exclude from (negative indices) neighborhoods.
`return_dissimilarity`	Logical indicating whether to return the dissimilarity matrix. Default is `FALSE`.
`k`	Deprecated.
`k_diss`	Deprecated.
`k_range`	Deprecated.
`pc_selection`	Deprecated.
`center`	Deprecated.
`scale`	Deprecated.
`documentation`	Deprecated.
`...`	Additional arguments (currently unused).

Details

This function is useful for reducing large reference sets by identifying only relevant neighbors before running `mbl`.

If `Xu` is not provided, the function searches for neighbors within `Xr` itself (excluding self-matches). If `Xu` is provided, neighbors of each observation in `Xu` are searched in `Xr`.

The `spike` argument allows forcing specific observations into or out of all neighborhoods. Positive indices are always included; negative indices are always excluded.

Value

A list containing:

neighbors Matrix of `Xr` indices for each query observation's neighbors, sorted by dissimilarity (columns = query observations).

neighbors_diss Matrix of dissimilarity scores corresponding to `neighbors`.

unique_neighbors Vector of unique `Xr` indices that appear in any neighborhood.

k_diss_info If `neighbors_diss()` was used, a `data.frame` with columns for observation index, number of neighbors found, and final number after applying bounds.

dissimilarity If `return_dissimilarity = TRUE`, the full dissimilarity object.

projection If the dissimilarity method includes `return_projection = TRUE`, the projection object.

gh If the dissimilarity method includes `gh = TRUE`, the GH distances.

Author(s)

Leonardo Ramirez-Lopez

References

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Stevens, A., Dematte, J.A.M., Scholten, T. 2013a. The spectrum-based learner: A new local approach for modeling soil vis-NIR spectra of complex data sets. Geoderma 195-196, 268-279.

Ramirez-Lopez, L., Behrens, T., Schmidt, K., Viscarra Rossel, R., Dematte, J.A.M., Scholten, T. 2013b. Distance and similarity-search metrics for use with soil vis-NIR spectra. Geoderma 199, 43-53.

See Also

`dissimilarity`, `mbl`, `neighbors_k`, `neighbors_diss`

Examples

```
library(prospectr)
data(NIRsoil)

Xu <- NIRsoil$spc[!as.logical(NIRsoil$train), ]
Yu <- NIRsoil$CEC[!as.logical(NIRsoil$train)]
Yr <- NIRsoil$CEC[as.logical(NIRsoil$train)]
```

```
Xr <- NIRsoil$spc[as.logical(NIRsoil$train), ]

Xu <- Xu[!is.na(Yu), ]
Yu <- Yu[!is.na(Yu)]
Xr <- Xr[!is.na(Yr), ]
Yr <- Yr[!is.na(Yr)]

# Correlation-based neighbor search with k neighbors
ex1 <- search_neighbors(
  Xr = Xr, Xu = Xu,
  diss_method = diss_correlation(),
  neighbors = neighbors_k(40)
)

# PCA-based with OPC selection
ex2 <- search_neighbors(
  Xr = Xr, Xu = Xu,
  diss_method = diss_pca(
    ncomp = ncomp_by_opc(40),
    scale = TRUE,
    return_projection = TRUE
  ),
  Yr = Yr,
  neighbors = neighbors_k(50)
)

# Observations not in any neighborhood
setdiff(seq_len(nrow(Xr)), ex2$unique_neighbors)

# Dissimilarity threshold-based selection
ex3 <- search_neighbors(
  Xr = Xr, Xu = Xu,
  diss_method = diss_pls(
    ncomp = ncomp_by_opc(40),
    scale = TRUE
  ),
  Yr = Yr,
  neighbors = neighbors_diss(threshold = 0.5, k_min = 10, k_max = 100)
)
```

`sid` *A function for computing the spectral information divergence between spectra (sid)*

Description

Experimental

This function computes the spectral information divergence/dissimilarity between spectra based on the kullback-leibler divergence algorithm (see details).

Usage

```
sid(Xr, Xu = NULL,
    mode = "density",
    center = FALSE, scale = FALSE,
    kernel = "gaussian",
    n = if(mode == "density") round(0.5 * ncol(Xr)),
    bw = "nrd0",
    reg = 1e-04,
    ...)
```

Arguments

Xr	a matrix containing the spectral (reference) data.
Xu	an optional matrix containing the spectral data of a second set of observations.
mode	the method to be used for computing the spectral information divergence. Options are "density" (default) for computing the divergence values on the density distributions of the spectral observations, and "feature" for computing the divergence vales on the spectral variables. See details.
center	a logical indicating if the computations must be carried out on the centred X and Xu (if specified) matrices. If mode = "feature" centring is not carried out since this option does not accept negative values which are generated after centring the matrices. Default is FALSE. See details.
scale	a logical indicating if the computations must be carried out on the variance scaled X and Xu (if specified) matrices. Default is TRUE.
kernel	if mode = "density" a character string indicating the smoothing kernel to be used. It must be one of "gaussian" (default), "rectangular", "triangular", "epanechnikov", "biweight", "cosine" or "optcosine". See the density function of the stats package.
n	if mode = "density" a numerical value indicating the number of equally spaced points at which the density is to be estimated. See the density function of the stats package for further details. Default is round(0.5 * ncol(X)).
bw	if mode = "density" a numerical value indicating the smoothing kernel bandwidth to be used. Optionally the character string "nrd0" can be used, it computes the bandwidth using the bw.nrd0 function of the stats package (see bw.nrd0). See the density and the bw.nrd0 functions for more details. By default "nrd0" is used, in this case the bandwidth is computed as bw.nrd0(as.vector(X)), if Xu is specified the bandwidth is computed as bw.nrd0(as.vector(rbind(X, Xu))).
reg	a numerical value larger than 0 which indicates a regularization parameter. Values (probabilities) below this threshold are replaced by this value for numerical stability. Default is 1e-4.
...	additional arguments to be passed to the density function of the base package.

Details

This function computes the spectral information divergence (distance) between spectra. When `mode = "density"`, the function first computes the probability distribution of each spectrum which result in a matrix of density distribution estimates. The density distributions of all the observations in the datasets are compared based on the kullback-leibler divergence algorithm. When `mode = "feature"`, the kullback-leibler divergence between all the observations is computed directly on the spectral variables. The spectral information divergence (SID) algorithm (Chang, 2000) uses the Kullback-Leibler divergence (KL) or relative entropy (Kullback and Leibler, 1951) to account for the vis-NIR information provided by each spectrum. The SID between two spectra (x_i and x_j) is computed as follows:

$$sid(x_i, x_j) = KL(x_i \,||\, x_j) + KL(x_j \,||\, x_i)$$

$$sid(x_i, x_j) = \sum_{l=1}^{k} p_l \; log(\frac{p_l}{q_l}) + \sum_{l=1}^{k} q_l \; log(\frac{q_l}{p_l})$$

where k represents the number of variables or spectral features, p and q are the probability vectors of x_i and x_i respectively which are calculated as:

$$p = \frac{x_i}{\sum_{l=1}^{k} x_{i,l}}$$

$$q = \frac{x_j}{\sum_{l=1}^{k} x_{j,l}}$$

From the above equations it can be seen that the original SID algorithm assumes that all the components in the data matrices are nonnegative. Therefore centering cannot be applied when `mode = "feature"`. If a data matrix with negative values is provided and `mode = "feature"`, the `sid` function automatically scales the matrix as follows:

$$X_s = \frac{X - min(X)}{max(X) - min(X)}$$

or

$$X_s = \frac{X - min(X, Xu)}{max(X, Xu) - min(X, Xu)}$$

$$Xu_s = \frac{Xu - min(X, Xu)}{max(X, Xu) - min(X, Xu)}$$

if `Xu` is specified. The 0 values are replaced by a regularization parameter (`reg` argument) for numerical stability. The default of the `sid` function is to compute the SID based on the density distributions of the spectra (`mode = "density"`). For each spectrum in `X` the density distribution is computed using the `density` function of the `stats` package. The 0 values of the estimated density distributions of the spectra are replaced by a regularization parameter (`"reg"` argument) for numerical stability. Finally the divergence between the computed spectral histogramas is computed using the SID algorithm. Note that if `mode = "density"`, the `sid` function will accept negative values and matrix centering will be possible.

Value

a `list` with the following components:

- `sid`: if only `"X"` is specified (i.e. `Xu = NULL`), a square symmetric matrix of SID distances between all the components in `"X"`. If both `"X"` and `"Xu"` are specified, a matrix of SID distances between the components in `"X"` and the components in `"Xu"`) where the rows represent the objects in `"X"` and the columns represent the objects in `"Xu"`
- `Xr`: the (centered and/or scaled if specified) spectral `X` matrix
- `Xu`: the (centered and/or scaled if specified) spectral `Xu` matrix
- `densityDisXr`: if `mode = "density"`, the computed density distributions of `Xr`
- `densityDisXu`: if `mode = "density"`, the computed density distributions of `Xu`

Author(s)

Leonardo Ramirez-Lopez

References

Chang, C.I. 2000. An information theoretic-based approach to spectral variability, similarity and discriminability for hyperspectral image analysis. IEEE Transactions on Information Theory 46, 1927-1932.

See Also

`density`

Examples

```
library(prospectr)

data(NIRsoil)

Xu <- NIRsoil$spc[!as.logical(NIRsoil$train), ]
Yu <- NIRsoil$CEC[!as.logical(NIRsoil$train)]
Yr <- NIRsoil$CEC[as.logical(NIRsoil$train)]
Xr <- NIRsoil$spc[as.logical(NIRsoil$train), ]

Xu <- Xu[!is.na(Yu), ]
Xr <- Xr[!is.na(Yr), ]

# Example 1
# Compute the SID distance between all the observations in Xr
xr_sid <- sid(Xr)
xr_sid

# Example 2
# Compute the SID distance between the observations in Xr and the observations
# in Xu
xr_xu_sid <- sid(Xr, Xu)
xr_xu_sid
```

Index