Package 'mlr3fda'

June 1, 2026

Title Extending 'mlr3' to Functional Data Analysis

Version 0.6.0

Description Extends the 'mlr3' ecosystem to functional analysis by adding support for irregular and regular functional data as defined in the 'tf' package. The package provides 'PipeOps' for preprocessing functional columns and for extracting scalar features, thereby allowing standard machine learning algorithms to be applied afterwards. Available operations include simple functional features such as the mean or maximum, smoothing, interpolation, flattening, and functional 'PCA'.

License LGPL-3

URL https://mlr3fda.mlr-org.com, https://github.com/mlr-org/mlr3fda

BugReports https://github.com/mlr-org/mlr3fda/issues

Depends mlr3 (>= 1.3.0), mlr3pipelines (>= 0.5.2), R (>= 4.1.0)

Imports checkmate, data.table (>= 1.18.0), lgr, mlr3misc (>= 0.19.0), paradox, R6, tf (>= 0.4.0)

Suggests FDboost, lme4, mboost, rpart, testthat (>= 3.3.0), tsfeatures, wavelets, withr

Config/roxygen2/markdown TRUE

Config/roxygen2/r6 TRUE

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Collate 'zzz.R' 'PipeOpFDABsignal.R' 'PipeOpFDACor.R' 'PipeOpFDADepth.R' 'PipeOpFDADerive.R' 'PipeOpFDAExtract.R' 'PipeOpFDAFlatten.R' 'PipeOpFDAFourier.R' 'PipeOpFDAInterpol.R' 'PipeOpFDARandomEffect.R' 'PipeOpFDARegister.R' 'PipeOpFDAScaleRange.R' 'PipeOpFDASmooth.R' 'PipeOpFDATsfeatures.R' 'PipeOpFDAWavelets.R' 'PipeOpFDAZoom.R' 'PipeOpFPCA.R' 'TaskClassif_phoneme.R' 'TaskRegr_dti.R' 'TaskRegr_fuel.R' 'bibentries.R' 'datasets.R' 'hash_input.R'

NeedsCompilation no

Author Maximilian Mücke [aut, cre] (ORCID:
<https://orcid.org/0009-0000-9432-9795>),
Sebastian Fischer [aut] (ORCID:
<https://orcid.org/0000-0002-9609-3197>),
Fabian Scheipl [ctb] (ORCID: <https://orcid.org/0000-0001-8172-3603>),
Bernd Bischl [ctb] (ORCID: <https://orcid.org/0000-0001-6002-6980>)

Maintainer Maximilian Mücke <muecke.maximilian@gmail.com>

Repository CRAN

Date/Publication 2026-06-01 10:40:02 UTC

Contents

`mlr3fda-package` *mlr3fda: Extending 'mlr3' to Functional Data Analysis*

Description

Extends the 'mlr3' ecosystem to functional analysis by adding support for irregular and regular functional data as defined in the 'tf' package. The package provides 'PipeOps' for preprocessing functional columns and for extracting scalar features, thereby allowing standard machine learning algorithms to be applied afterwards. Available operations include simple functional features such as the mean or maximum, smoothing, interpolation, flattening, and functional 'PCA'.

Data types

To extend mlr3 to functional data, two data types from the tf package are added:

- `tfd_irreg` - Irregular functional data, i.e. the functions are observed for potentially different inputs for each observation.
- `tfd_reg` - Regular functional data, i.e. the functions are observed for the same input for each individual.

Lang M, Binder M, Richter J, Schratz P, Pfisterer F, Coors S, Au Q, Casalicchio G, Kotthoff L, Bischl B (2019). "mlr3: A modern object-oriented machine learning framework in R." *Journal of Open Source Software*. doi:10.21105/joss.01903. https://joss.theoj.org/papers/10.21105/joss.01903.

Author(s)

Maintainer: Maximilian Mücke <muecke.maximilian@gmail.com> (ORCID)

Authors:

- Maximilian Mücke <muecke.maximilian@gmail.com> (ORCID)
- Sebastian Fischer <sebf.fischer@gmail.com> (ORCID)

Other contributors:

- Fabian Scheipl <fabian.scheipl@googlemail.com> (ORCID) [contributor]
- Bernd Bischl <bernd_bischl@gmx.net> (ORCID) [contributor]

See Also

Useful links:

- https://mlr3fda.mlr-org.com
- https://github.com/mlr-org/mlr3fda
- Report bugs at https://github.com/mlr-org/mlr3fda/issues

`mlr_pipeops_fda.bsignal` *B-spline Feature Extraction*

Description

This `PipeOp` extracts features from functional data using B-spline basis functions. The extracted features are B-spline coefficients that represent the functional data in the B-spline basis space. For more details, see `FDboost::bsignal()`, which is called internally.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreprocSimple, as well as the following parameters:

- inS :: character(1)
  Type of effect in the covariate index: one of "smooth", "linear", "constant". Default is "smooth".
- knots :: numeric()
  Either the number of interior knots or a vector of their positions.
- boundary.knots :: numeric(2)
  Boundary points at which to anchor the B-spline basis. Lower and upper boundary points for the spline basis. Defaults to the range of the data.
- degree :: integer(1)
  The degree of the regression spline. Default is 3L.
- differences :: integer(1)
  Order of difference penalty. Default is 1L.
- df :: numeric(1)
  Trace of the hat matrix, controlling smoothness. Default is 4.
- lambda :: any
  Smoothing parameter of the penalty term.
- center :: logical(1)
  Reparameterize the unpenalized part to zero-mean? Default is FALSE.
- cyclic :: logical(1)
  If true the fitted coefficient function coincides at the boundaries.
- Z :: any
  Custom transformation matrix for the spline design.
- penalty :: character(1)
  The penalty type: "ps" (P-spline) or "pss" (shrinkage). Default is "ps".
- check.ident :: logical(1)
  Use checks for identifiability of the effect. Default is FALSE.

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> mlr3pipelines::PipeOpTaskPreprocSimple -> PipeOpFDABsignal

Methods

Public methods:

- PipeOpFDABsignal$new()
- PipeOpFDABsignal$clone()

PipeOpFDABsignal$new(): Initializes a new instance of this Class.

Usage:

```
PipeOpFDABsignal$new(id = "fda.bsignal", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fda.bsignal"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDABsignal$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDABsignal$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_bsignal = po("fda.bsignal")
task_bsignal = po_bsignal$train(list(task))[[1L]]
task_bsignal$data()
```

`mlr_pipeops_fda.cor` *Cross-Correlation of Functional Data*

Description

Calculates the cross-correlation between two functional vectors using `tf::tf_crosscor()`. Note that it only operates on regular data and that the cross-correlation assumes that each column has the same domain.

To apply this `PipeOp` to irregular data, convert it to a regular grid first using `PipeOpFDAInterpol`. If you need to change the domain of the columns, use `PipeOpFDAScaleRange`.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreprocSimple`, as well as the following parameters:

- `arg` :: `numeric()`
  Grid to use for the cross-correlation.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `mlr3pipelines::PipeOpTaskPreprocSimple` -> `PipeOpFDACor`

Methods

Public methods:

- `PipeOpFDACor$new()`
- `PipeOpFDACor$clone()`

`PipeOpFDACor$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFDACor$new(id = "fda.cor", param_vals = list())
```

Arguments:

`id` (`character(1)`)
: Identifier of resulting object, default `"fda.cor"`.

`param_vals` (named `list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDACor$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDACor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
set.seed(1234L)
dt = data.table(y = 1:100, x1 = tf::tf_rgp(100L), x2 = tf::tf_rgp(100L))
task = as_task_regr(dt, target = "y")
po_cor = po("fda.cor")
task_cor = po_cor$train(list(task))[[1L]]
task_cor
```

---

mlr_pipeops_fda.depth *Functional Data Depth Features*

Description

Computes the data depth of functional features via `tf::tf_depth()`. Data depth measures how central each curve is relative to the others: values close to `1` indicate central curves and values close to `0` indicate extreme curves.

The depth is computed in-sample, i.e. each curve is scored relative to the other curves in the same data. The same operation is applied during training and prediction.

Irregular curves are interpolated to a common grid before the depth is computed. Curves that remain incomplete after interpolation (e.g. because they only cover part of the grid) are assigned a depth of `NA`.

Details

The "MHI" method ranks functions from lowest (0) to highest (1) instead of from most extreme to most central. The "RPD" method relies on random projections, so set a seed (e.g. via set.seed()) before training for reproducible results.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreprocSimple, as well as the following parameters:

- method :: character(1)
  The depth method to use. One of "MBD" (modified band depth, the default), "MHI" (modified hypograph index), "FM" (Fraiman-Muniz), "FSD" (functional spatial depth), or "RPD" (regularized projection depth). Initial value is "MBD".
- na.rm :: logical(1)
  Whether to remove missing observations before computing the depth. Initial value is TRUE.
- u :: numeric(1)
  Quantile level for the regularization. Only used when method is "RPD". Default is 0.01.
- n_projections :: integer(1)
  Number of projection directions. Only used when method is "RPD". Default is 5000.
- n_projections_beta :: integer(1)
  Number of directions for estimating the regularization parameter. Only used when method is "RPD". Default is 500.

Naming

The new names generally append a _depth to the corresponding column name. If a column was called "x", the corresponding new column will be called "x_depth".

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> mlr3pipelines::PipeOpTaskPreprocSimple -> PipeOpFDADepth

Methods

Public methods:

- PipeOpFDADepth$new()
- PipeOpFDADepth$clone()

PipeOpFDADepth$new(): Initializes a new instance of this Class.

Usage:

```
PipeOpFDADepth$new(id = "fda.depth", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of resulting object, default "fda.depth".

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDADepth$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDADepth$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_depth = po("fda.depth", method = "MBD")
task_depth = po_depth$train(list(task))[[1L]]
task_depth$data(cols = c("NIR_depth", "UVVIS_depth"))
```

`mlr_pipeops_fda.derive`

Derivatives of Functional Columns

Description

Computes derivatives of functional features via `tf::tf_derive()`. For `tfd` inputs derivatives are obtained by finite differencing of the function evaluations, for `tfb` inputs by finite differencing of the basis functions.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreprocSimple`, as well as the following parameters:

- `order` :: `integer(1)`
  Order of the derivative. Must be a positive integer. Initial value is `1`.
- `arg` :: `numeric()`
  Optional grid to use for the finite differences. If `NULL` (the default), the argument grid of each functional column is used. For `tfd_irreg` inputs, supplying `arg` interpolates the data to a common grid before differentiating.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `mlr3pipelines::PipeOpTaskPreprocSimple` -> `PipeOpFDADerive`

Methods

Public methods:

- PipeOpFDADerive$new()
- PipeOpFDADerive$clone()

`PipeOpFDADerive$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFDADerive$new(id = "fda.derive", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fda.derive"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDADerive$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDADerive$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_deriv = po("fda.derive", order = 1)
task_deriv = po_deriv$train(list(task))[[1L]]
task_deriv$data(cols = c("NIR", "UVVIS"))
```

`mlr_pipeops_fda.extract`

Extract Simple Features from Functional Columns

Description

This is the class that extracts simple features from functional columns. Note that it only operates on values that were actually observed and does not interpolate.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreprocSimple, as well as the following parameters:

- `drop` :: `logical(1)`
Whether to drop the original `functional` features and only keep the extracted features. Note that this does not remove the features from the backend, but only from the active column role `feature`. Initial value is `TRUE`.

- features :: list() | character()
  A list of features to extract. Each element can be either a function or a string. If the element is a function it requires the following arguments: arg and value and returns a numeric. For string elements, the following predefined features are available: "mean", "max", "min", "slope", "median", "var", "sd". Initial value is c("mean", "max", "min", "slope", "median", "var").
- left :: numeric()
  The left boundary of the window. Initial value is -Inf. The window is specified such that all values >=left and <=right are kept for the computations.
- right :: numeric()
  The right boundary of the window. Initial value is Inf.

Naming

The new names generally append a _{feature} to the corresponding column name. However this can lead to name clashes with existing columns. This is solved as follows: If a column was called "x" and the feature is "mean", the corresponding new column will be called "x_mean". In case of duplicates, unique names are obtained using make.unique() and a warning is given.

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> mlr3pipelines::PipeOpTaskPreprocSimple -> PipeOpFDAExtract

Methods

Public methods:

- PipeOpFDAExtract$new()
- PipeOpFDAExtract$clone()

PipeOpFDAExtract$new(): Initializes a new instance of this Class.

Usage:

```
PipeOpFDAExtract$new(id = "fda.extract", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of resulting object, default "fda.extract".

param_vals (named list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

PipeOpFDAExtract$clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDAExtract$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_fmean = po("fda.extract", features = "mean")
task_fmean = po_fmean$train(list(task))[[1L]]

# add more than one feature
pop = po("fda.extract", features = c("mean", "median", "var"))
task_features = pop$train(list(task))[[1L]]

# add a custom feature
po_custom = po("fda.extract",
  features = list(mean = function(arg, value) mean(value, na.rm = TRUE))
)
task_custom = po_custom$train(list(task))[[1L]]
task_custom
```

`mlr_pipeops_fda.flatten`

Flatten Functional Columns

Description

Convert regular functional features (e.g. all individuals are observed at the same time-points) to new columns, one for each input value to the function.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreprocSimple`.

Naming

The new names generally append `_1`, `_2`, ... to the corresponding column name. However this can lead to name clashes with existing columns. This is solved as follows: If a column was called `"x"`, the corresponding new columns will be called `"x_1"`, `"x_2"`, etc. In case of duplicates, unique names are obtained using `make.unique()` and a warning is given.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `mlr3pipelines::PipeOpTaskPreprocSimple` -> `PipeOpFDAFlatten`

Methods

Public methods:

- `PipeOpFDAFlatten$new()`
- `PipeOpFDAFlatten$clone()`

`PipeOpFDAFlatten$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFDAFlatten$new(id = "fda.flatten", param_vals = list())
```

Arguments:

`id` (`character(1)`)
: Identifier of resulting object, default `"fda.flatten"`.

`param_vals` (named `list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDAFlatten$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDAFlatten$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
task = tsk("fuel")
pop = po("fda.flatten")
task_flat = pop$train(list(task))[[1L]]
```

`mlr_pipeops_fda.fourier`

Fast Fourier Transform Features

Description

This `PipeOp` extracts Fourier coefficients from functional columns. For more details, see `stats::fft()`, which is called internally. Only the one-sided spectrum is returned since the input is real-valued (Oppenheim and Schafer, 2010).

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreprocSimple`, as well as the following parameters:

- `type` :: `character(1)`
  Which feature to extract from the Fourier coefficients. `"amplitude"` returns the magnitude. `"phase"` returns the phase shift in degrees (values in [-180, 180]). Initial value is `"phase"`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `mlr3pipelines::PipeOpTaskPreprocSimple` -> `PipeOpFDAFourier`

Methods

Public methods:

- PipeOpFDAFourier$new()
- PipeOpFDAFourier$clone()

`PipeOpFDAFourier$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFDAFourier$new(id = "fda.fourier", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fda.fourier"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDAFourier$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDAFourier$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Oppenheim, V A, Schafer, W R (2010). *Discrete-Time Signal Processing*, 3rd edition. Pearson. ISBN 978-0131988422.

Examples

```
task = tsk("fuel")
po_fourier = po("fda.fourier")
task_fourier = po_fourier$train(list(task))[[1L]]
task_fourier$data()
```

---

mlr_pipeops_fda.fpca *Functional Principal Component Analysis*

Description

This PipeOp applies a functional principal component analysis (FPCA) to functional columns and then extracts the principal components as features. This is done using a (truncated) weighted SVD.

To apply this PipeOp to irregular data, convert it to a regular grid first using PipeOpFDAInterpol.

For more details, see `tf::tfb_fpc()`, which is called internally.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as the following parameters:

- pve :: numeric(1)
  The percentage of variance explained that should be retained. Default is 0.995.
- n_components :: integer(1)
  The number of principal components to extract. This parameter is initialized to Inf.

Naming

The new names generally append a _pc_{number} to the corresponding column name. If a column was called "x" and there are three principal components, the corresponding new columns will be called "x_pc_1", "x_pc_2", "x_pc_3".

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> PipeOpFPCA

Methods

Public methods:

- PipeOpFPCA$new()
- PipeOpFPCA$clone()

PipeOpFPCA$new(): Initializes a new instance of this Class.

Usage:

```
PipeOpFPCA$new(id = "fda.fpca", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of resulting object, default "fda.fpca".

param_vals (named list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

PipeOpFPCA$clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpFPCA$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_fpca = po("fda.fpca", n_components = 3L)
task_fpca = po_fpca$train(list(task))[[1L]]
task_fpca$data()
```

`mlr_pipeops_fda.interpol` *Interpolate Functional Columns*

Description

Interpolate functional features (e.g. all individuals are observed at different time-points) to a common grid. This is useful if you want to compare functional features across observations. The interpolation is done using the `tf` package. See `tfd()` for details.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreprocSimple`, as well as the following parameters:

- `grid` :: `character(1)` | `numeric()`
  The grid to use for interpolation. If `grid` is numeric, it must be a sequence of values to use for the grid or a single value that specifies the number of points to use for the grid, requires `left` and `right` to be specified in the latter case. If `grid` is a character, it must be one of:
  - `"union"`: This option creates a grid based on the union of all argument points from the provided functional features. This means that if the argument points across features are \(t_1, t_2, ..., t_n\), then the grid will be the combined unique set of these points. This option is generally used when the argument points vary across observations and a common grid is needed for comparison or further analysis.
  - `"intersect"`: Creates a grid using the intersection of all argument points of a feature. This grid includes only those points that are common across all functional features, facilitating direct comparison on a shared set of points.
  - `"minmax"`: Generates a grid within the range of the maximum of the minimum argument points to the minimum of the maximum argument points across features. This bounded grid encapsulates the argument point range common to all features. Note: For regular functional data this has no effect as all argument points are the same. Initial value is `"union"`.
- `method` :: `character(1)`
  One of:
  - `"linear"`: applies linear interpolation without extrapolation (see `tf::tf_approx_linear()`).
  - `"spline"`: applies cubic spline interpolation (see `tf::tf_approx_spline()`).
  - `"fill_extend"`: applies linear interpolation with constant extrapolation (see `tf::tf_approx_fill_extend()`).
  - `"locf"`: applies "last observation carried forward" interpolation (see `tf::tf_approx_locf()`).
  - `"nocb"`: applies "next observation carried backward" interpolation (see `tf::tf_approx_nocb()`).

  Default is `"linear"`.
- `left` :: `numeric()`
  The left boundary of the window. The window is specified such that all values >=left and <=right are kept for the computations.
- `right` :: `numeric()`
  The right boundary of the window.

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> mlr3pipelines::PipeOpTaskPreprocSimple -> PipeOpFDAInterpol

Methods

Public methods:

- PipeOpFDAInterpol$new()
- PipeOpFDAInterpol$clone()

`PipeOpFDAInterpol$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFDAInterpol$new(id = "fda.interpol", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fda.interpol"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDAInterpol$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDAInterpol$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
task = tsk("fuel")
pop = po("fda.interpol")
task_interpol = pop$train(list(task))[[1L]]
task_interpol$data()
```

---

mlr_pipeops_fda.random_effect

Extract Random Effects from Functional Columns

Description

This is the class that extracts random effects, specifically random intercepts and random slopes, from functional columns. This PipeOp fits a linear mixed model, specifically a random intercept and random slope model, using the lme4::lmer() function. The target variable is the value of the functional feature which is regressed on the functional feature's argument while subject id determines the grouping structure. After model estimation, the random effects are extracted and assigned to the correct id.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreprocSimple.

Naming

The new names append _random_intercept and _random_slope to the corresponding column name of the functional feature.

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> mlr3pipelines::PipeOpTaskPreprocSimple -> PipeOpFDARandomEffect

Methods

Public methods:

- PipeOpFDARandomEffect$new()
- PipeOpFDARandomEffect$clone()

PipeOpFDARandomEffect$new(): Initializes a new instance of this Class.

Usage:

```
PipeOpFDARandomEffect$new(id = "fda.random_effect", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of resulting object, default "fda.random_effect".

param_vals (named list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

PipeOpFDARandomEffect$clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDARandomEffect$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
task = tsk("dti")
po_fre = po("fda.random_effect")
task_fre = po_fre$train(list(task))[[1L]]
```

`mlr_pipeops_fda.register`

Register (Align) Functional Columns

Description

Aligns functional features by estimating warping functions against a template using `tf::tf_register()`. Registration reduces phase (horizontal) variability while preserving amplitude (vertical) variability, which is useful when curves share a common shape but differ in the timing of their features.

During training, a template is learned for each functional column (either estimated iteratively from the data or supplied via `args`). The template is stored as part of the `$state` and reused at predict time so that new observations are aligned to the same reference as the training data.

Supported methods are `"srvf"`, `"cc"`, and `"affine"`. The `"landmark"` method from `tf::tf_register()` is not supported because it requires per-observation landmark positions for both training and prediction data, which does not fit a stateful preprocessing step. For landmark registration, use `tf::tf_register()` directly and feed the aligned data into the task.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreproc`, as well as the following parameters:

- `method` :: `character(1)`
  Registration method. One of:
  - `"srvf"`: elastic registration using the Square Root Velocity Framework via `fdasrvf::time_warping()`. Requires regular grids and the **fdasrvf** package. Default template is the Karcher mean.
  - `"cc"`: continuous-criterion registration with monotone spline warps. Requires regular grids. Default template is the arithmetic mean.
  - `"affine"`: affine (shift and/or scale) registration with warps of the form $h(t) = a \cdot t + b$. Supports regular and irregular grids. Default template is the arithmetic mean.

  Default is `"srvf"`.
- `args` :: named `list()`
  Method-specific arguments passed to `tf::tf_register()` via `...`. See the help page of `tf::tf_estimate_warps()` for valid arguments (e.g. `lambda` / `penalty_method` for `"srvf"`; `nbasis`, `lambda`, `crit`, `conv`, `iterlim` for `"cc"`; `type`, `shift_range`, `scale_range` for `"affine"`). An optional `template` entry is honored at training time and stored in the state.
- `max_iter` :: `integer(1)`
  Maximum number of Procrustes-style template refinement iterations during training. Default is `3`. Ignored at predict time because the stored template is used directly.
- `tol` :: `numeric(1)`
  Convergence tolerance for template refinement during training. Default is `0.01`.

State

`$state$templates` contains the learned template (as a length-1 `tf` vector) for each functional column.

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> PipeOpFDARegister

Methods

Public methods:

- PipeOpFDARegister$new()
- PipeOpFDARegister$clone()

`PipeOpFDARegister$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFDARegister$new(id = "fda.register", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fda.register"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDARegister$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDARegister$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
set.seed(1)
task = tsk("fuel")
po_reg = po("fda.register", method = "affine", args = list(type = "shift_scale"))
task_reg = po_reg$train(list(task))[[1L]]
task_reg$data(cols = c("NIR", "UVVIS"))
```

---

mlr_pipeops_fda.scalerange

Linearly Transform the Domain of Functional Data

Description

Linearly transform the domain of functional data so they are between `lower` and `upper`. The formula for this is $x' = offset + x * scale$, where $scale$ is $(upper - lower)/(max(x) - min(x))$ and $offset$ is $-min(x) * scale + lower$. The same transformation is applied during training and prediction.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreproc, as well as the following parameters:

- lower :: numeric(1)
  Target value of smallest item of input data. Initialized to 0.
- upper :: numeric(1)
  Target value of greatest item of input data. Initialized to 1.

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> PipeOpFDAScaleRange

Methods

Public methods:

- PipeOpFDAScaleRange$new()
- PipeOpFDAScaleRange$clone()

PipeOpFDAScaleRange$new(): Initializes a new instance of this Class.

Usage:

```
PipeOpFDAScaleRange$new(id = "fda.scalerange", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of resulting object, default "fda.scalerange".

param_vals (named list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

PipeOpFDAScaleRange$clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDAScaleRange$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_scale = po("fda.scalerange", lower = -1, upper = 1)
task_scale = po_scale$train(list(task))[[1L]]
task_scale$data()
```

`mlr_pipeops_fda.smooth` *Smooth Functional Columns*

Description

Smoothes functional data using `tf::tf_smooth()`. This preprocessing operator is similar to `PipeOpFDAInterpol`, however it does not interpolate to unobserved x-values, but rather smooths the observed values.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreprocSimple`, as well as the following parameters:

- `method` :: `character(1)`
  One of:
  - `"lowess"`: locally weighted scatterplot smoothing (default)
  - `"rollmean"`: rolling mean
  - `"rollmedian"`: rolling median
  - `"savgol"`: Savitzky-Golay filtering

  All methods but `"lowess"` ignore non-equidistant arg values.
- `args` :: named `list()`
  List of named arguments that is passed to `tf_smooth()`. See the help page of `tf_smooth()` for default values.
- `verbose` :: `logical(1)`
  Whether to print messages during the transformation. Is initialized to `FALSE`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `mlr3pipelines::PipeOpTaskPreprocSimple` -> `PipeOpFDASmooth`

Methods

Public methods:

- `PipeOpFDASmooth$new()`
- `PipeOpFDASmooth$clone()`

`PipeOpFDASmooth$new()`**:** Initializes a new instance of this Class.

Usage:

```
PipeOpFDASmooth$new(id = "fda.smooth", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of resulting object, default `"fda.smooth"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDASmooth$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDASmooth$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_smooth = po("fda.smooth", method = "rollmean", args = list(k = 5))
task_smooth = po_smooth$train(list(task))[[1L]]
task_smooth
task_smooth$data(cols = c("NIR", "UVVIS"))
```

`mlr_pipeops_fda.tsfeats`
Time Series Feature Extraction

Description

This `PipeOp` extracts time series features from functional columns.

For more details, see `tsfeatures::tsfeatures()`, which is called internally.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreprocSimple`, as well as the following parameters:

- `features` :: `character()`
  Function names which return numeric vectors of features. All features returned by these functions must be named if they return more than one feature. Default is `c("frequency", "stl_features", "entropy", "acf_features")`.
- `scale` :: `logical(1)`
  If `TRUE`, data is scaled to mean 0 and sd 1 before features are computed. Default is `TRUE`.
- `trim` :: `logical(1)`
  If `TRUE`, data is trimmed by `trim_amount` before features are computed. Values larger than `trim_amount` in absolute value are set to `NA`. Default is `FALSE`.
- `trim_amount` :: `numeric(1)`
  Default level of trimming. Default is `0.1`.
- `parallel` :: `logical(1)`
  If `TRUE`, the features are computed in parallel. Default is `FALSE`.

- multiprocess :: any
  The function from the future package to use for parallel processing. Default is future::multisession().
- na.action :: any
  A function to handle missing values. Default is stats::na.pass().

Naming

The new names generally append a _{feature} to the corresponding column name. If a column was called "x" and the feature is "trend", the corresponding new column will be called "x_trend".

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> mlr3pipelines::PipeOpTaskPreprocSimple -> PipeOpFDATsfeatures

Methods

Public methods:

- PipeOpFDATsfeatures$new()
- PipeOpFDATsfeatures$clone()

PipeOpFDATsfeatures$new(): Initializes a new instance of this Class.

Usage:

```
PipeOpFDATsfeatures$new(id = "fda.tsfeats", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of resulting object, default "fda.tsfeats".

param_vals (named list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

PipeOpFDATsfeatures$clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDATsfeatures$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_tsfeats = po("fda.tsfeats")
task_tsfeats = po_tsfeats$train(list(task))[[1L]]
task_tsfeats$data()
```

`mlr_pipeops_fda.wavelets` *Discrete Wavelet Transform Features*

Description

This `PipeOp` extracts discrete wavelet transform coefficients from functional columns. For more details, see `wavelets::dwt()`, which is called internally.

Parameters

The parameters are the parameters inherited from `PipeOpTaskPreprocSimple`, as well as the following parameters:

- `filter` :: `character(1)` | `numeric()` | `wavelets::wt.filter()`
  Specifies which filter should be used. Must be either a `wavelets::wt.filter()` object, an even numeric vector or a string. In case of a string must be one of `"d"|"la"|"bl"|"c"` followed by an even number for the level of the filter. The level of the filter needs to be smaller or equal than the time-series length. For more information and acceptable filters see `help(wt.filter)`. Default is `"la8"`.
- `n.levels` :: `integer(1)`
  An integer specifying the level of the decomposition.
- `boundary` :: `character(1)`
  Boundary to be used. `"periodic"` assumes circular time series, for `"reflection"` the series is extended to twice its length. Default is `"periodic"`.
- `fast` :: `logical(1)`
  Should the pyramid algorithm be calculated with an internal C function? Default is `TRUE`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `mlr3pipelines::PipeOpTaskPreprocSimple` -> `PipeOpFDAWavelets`

Methods

Public methods:

- `PipeOpFDAWavelets$new()`
- `PipeOpFDAWavelets$clone()`

`PipeOpFDAWavelets$new()`**:** Initializes a new instance of this Class.

Usage:

```
PipeOpFDAWavelets$new(id = "fda.wavelets", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fda.wavelets"`.

param_vals (named list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

PipeOpFDAWavelets$clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDAWavelets$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
task = tsk("fuel")
po_wavelets = po("fda.wavelets")
task_wavelets = po_wavelets$train(list(task))[[1L]]
task_wavelets$data()
```

---

mlr_pipeops_fda.zoom *Zoom In/Out on Functional Columns*

Description

Zoom in or out on functional features by restricting their domain to a specified window. This operation extracts a subset of each function by defining new lower and upper boundaries, effectively cropping the functional data to focus on a specific region of interest. Calls tf::tf_zoom() from package **tf**.

Parameters

The parameters are the parameters inherited from PipeOpTaskPreprocSimple, as well as the following parameters:

- begin :: numeric()
  The lower limit of the domain. Can be a single value applied to all functional columns, or a numeric of length equal to the number of observations. The window includes all values where argument >= begin. If not specified, defaults to the lower limit of each function's domain.
- end :: numeric()
  The upper limit of the domain.

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> mlr3pipelines::PipeOpTaskPreprocSimple -> PipeOpFDAZoom

Methods

Public methods:

- PipeOpFDAZoom$new()
- PipeOpFDAZoom$clone()

`PipeOpFDAZoom$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFDAZoom$new(id = "fda.zoom", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fda.zoom"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFDAZoom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFDAZoom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
task = tsk("fuel")
pop = po("fda.zoom", begin = 50, end = 100)
task_zoom = pop$train(list(task))[[1L]]
task_zoom$data()
```

---

mlr_tasks_dti *Diffusion Tensor Imaging (DTI) Regression Task*

Description

This dataset contains two functional covariates and three scalar covariates. The goal is to predict the PASAT score. `pasat` represents the PASAT score at each visit. `subject_id` represents the subject ID. `cca` represents the fractional anisotropy tract profiles from the corpus callosum. `sex` indicates subject's sex. `rcst` represents the fractional anisotropy tract profiles from the right corticospinal tract. Rows containing NAs are removed.

This is a subset of the full dataset, which is contained in the package `refund`.

Format

R6::R6Class inheriting from mlr3::TaskRegr.

Dictionary

This mlr3::Task can be instantiated via the dictionary mlr3::mlr_tasks or with the associated sugar function `mlr3::tsk()`:

```
mlr_tasks$get("dti")
tsk("dti")
```

Meta Information

- Task type: "regr"
- Dimensions: 340x4
- Properties: "groups"
- Has Missings: `FALSE`
- Target: "pasat"
- Features: "cca", "rcst", "sex"

References

Goldsmith, Jeff, Bobb, Jennifer, Crainiceanu, M C, Caffo, Brian, Reich, Daniel (2011). "Penalized functional regression." *Journal of Computational and Graphical Statistics*, **20**(4), 830–851.

Brain dataset courtesy of Gordon Kindlmann at the Scientific Computing and Imaging Institute, University of Utah, and Andrew Alexander, W. M. Keck Laboratory for Functional Brain Imaging and Behavior, University of Wisconsin-Madison.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from `https://www.openml.org`.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:
  - Unsupervised clustering: **mlr3cluster**
  - Probabilistic supervised regression and survival analysis: `https://mlr3proba.mlr-org.com/`.

Other Task: `mlr_tasks_fuel`, `mlr_tasks_phoneme`

`mlr_tasks_fuel` *Fuel Regression Task*

Description

This dataset contains two functional covariates and one scalar covariate. The goal is to predict the heat value of some fuel based on the ultraviolet radiation spectrum and infrared ray radiation and one scalar column called h2o.

This is a subset of the full dataset, which is contained in the package `FDboost`.

Format

R6::R6Class inheriting from mlr3::TaskRegr.

Dictionary

This mlr3::Task can be instantiated via the dictionary mlr3::mlr_tasks or with the associated sugar function `mlr3::tsk()`:

```
mlr_tasks$get("fuel")
tsk("fuel")
```

Meta Information

- Task type: "regr"
- Dimensions: 129x4
- Properties: -
- Has Missings: `FALSE`
- Target: "heatan"
- Features: "NIR", "UVVIS", "h2o"

References

Brockhaus, Sarah, Scheipl, Fabian, Hothorn, Torsten, Greven, Sonja (2015). "The functional linear array model." *Statistical Modelling*, **15**(3), 279–300.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html`
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from `https://www.openml.org`.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks

- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:
  - Unsupervised clustering: **mlr3cluster**
  - Probabilistic supervised regression and survival analysis: `https://mlr3proba.mlr-org.com/`.

Other Task: `mlr_tasks_dti`, `mlr_tasks_phoneme`

`mlr_tasks_phoneme` *Phoneme Classification Task*

Description

The task contains a single functional covariate and 5 equally big classes (aa, ao, dcl, iy, sh). The aim is to predict the class of the phoneme from the functional feature, which is a log-periodogram. This is a subset of the full dataset, which is contained in the package `fda.usc`.

Format

R6::R6Class inheriting from mlr3::TaskClassif.

Dictionary

This mlr3::Task can be instantiated via the dictionary mlr3::mlr_tasks or with the associated sugar function `mlr3::tsk()`:

```
mlr_tasks$get("phoneme")
tsk("phoneme")
```

Meta Information

- Task type: "classif"
- Dimensions: 250x2
- Properties: "multiclass"
- Has Missings: `FALSE`
- Target: "class"
- Features: "X"

References

Ferraty, Frédéric, Vieu, Philippe (2003). "Curves discrimination: a nonparametric functional approach." *Computational Statistics & Data Analysis*, **44**(1-2), 161–173.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from https://www.openml.org.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:
  - Unsupervised clustering: **mlr3cluster**
  - Probabilistic supervised regression and survival analysis: https://mlr3proba.mlr-org.com/.

Other Task: `mlr_tasks_dti`, `mlr_tasks_fuel`

Index