Package 'mlr3fda'

July 22, 2024

Title Extending 'mlr3' to Functional Data Analysis

Version 0.2.0

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 (>= 0.14.0), mlr3pipelines (>= 0.5.2), R (>= 3.1.0)

Imports checkmate, data.table, lgr, mlr3misc (>= 0.14.0), paradox, R6,
tf (>= 0.3.4)

Suggests rpart, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Collate 'zzz.R' 'PipeOpFDACor.R' 'PipeOpFDAExtract.R'
'PipeOpFDAFlatten.R' 'PipeOpFDAInterpol.R'
'PipeOpFDAScaleRange.R' 'PipeOpFDASmooth.R' 'PipeOpFPCA.R'
'TaskClassif_phoneme.R' 'TaskRegr_dti.R' 'TaskRegr_fuel.R'
'bibentries.R' 'datasets.R' 'hash_input.R'

NeedsCompilation no

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

Maintainer Sebastian Fischer <sebf.fischer@gmail.com>

Repository CRAN

Date/Publication 2024-07-22 11:40:02 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: Sebastian Fischer <`sebf.fischer@gmail.com`> ([ORCID](#))

Authors:

- Maximilian Mücke <`muecke.maximilian@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.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 irregualr 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()`

Method 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 other-
    wise be set during construction. Default list().

Method 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.extract
Extracts 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 if is 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"`. Initial is `c("mean", "max", "min", "slope", "median", "var")`

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

- `right :: numeric()`
  The right boundary of the window. Initial 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()](#)

Method `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 is `"fda.extract"`.

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

Method `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`

Flattens 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 a `_1`, ..., 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`](#) -> `PipeOpFDAFlatten`

Methods

Public methods:

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

Method `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()`.

Method `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))
```

---

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 irregualr 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 the there are three principcal 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()](#)

Method 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 is "fda.fpca".

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

Method 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)
  Defaults to "linear". 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()`).
- left :: numeric()
  The left boundary of the window. The window is specified such that the 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()](#)

Method 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 other-
    wise be set during construction. Default list().

Method 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.scalerange

Linearly Transform the Domain of Functional Data.

Description

Linearly transform the domain of functional data so they are between lower and upper. The for-
mula 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.

- uppper :: 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()](#)

Method 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 other-
    wise be set during construction. Default list().

Method 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

Smoothing 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 meadian
    - "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()`

Method 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 other-
    wise be set during construction. Default list().

Method 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_tasks_dti   *Diffusion Tensor Imaging (DTI) Regression Task*

Description

This dataset contains two functional covariates and three scalar covariate. The goal is to predict the
PASAT score. pasat represents the PASAT score at each vist. 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 Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function
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: 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 [Task](#) can be instantiated via the [dictionary](#) [mlr_tasks](#) or with the associated sugar function [tsk()](#):

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

Meta Information

- Task type: "regr"
- Dimensions: 129x4
- Properties: -
- Has Missings: FALSE
- Target: "heatan"
- Features: "NIR", "UVVIS", "h20"

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](#): [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 in the functional, 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 Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function 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, Frédric, 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: 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