Package: mlr3spatial (via r-universe)

August 29, 2024

Title Support for Spatial Objects Within the 'mlr3' Ecosystem

Version 0.5.0

Date 2024-03-09

Description Extends the 'mlr3' ML framework with methods for spatial objects. Data storage and prediction are supported for packages 'terra', 'raster' and 'stars'.

License LGPL-3

URL https://mlr3spatial.mlr-org.com, https://github.com/mlr-org/mlr3spatial

BugReports https://github.com/mlr-org/mlr3spatial/issues

Depends mlr3 (>= 0.14.0), R (>= 3.1.0)

Imports checkmate (>= 2.0.0), data.table (>= 1.14.0), lgr (>= 0.4.2), methods, mlr3misc (>= 0.11.0), R6 (>= 2.5.0), sf, terra (>= 1.6-3), utils

Suggests bench, future, future.callr, knitr, mlr3learners (>= 0.4.5), paradox, ranger, raster, rmarkdown, rpart, stars (>= 0.5-5), testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Collate 'DataBackendRaster.R' 'DataBackendVector.R' 'LearnerClassifSpatial.R' 'LearnerRegrSpatial.R' 'TaskRegrST.R' 'TaskClassifST.R' 'TaskClassif_leipzig.R' 'as_task_classif_st.R' 'as_task_regr_st.R' 'as_task_unsupervised.R' 'data.R' 'helper.R' 'predict_spatial.R' 'zzz.R'

Repository https://mlr-org.r-universe.dev

RemoteUrl https://github.com/mlr-org/mlr3spatial

RemoteRef v0.5.0

RemoteSha 6dd3417b02fcd36b1493449db189e3cf2f6f61f0

Contents

`mlr3spatial-package` *mlr3spatial: Support for Spatial Objects Within the 'mlr3' Ecosystem*

Description

Extends the 'mlr3' ML framework with methods for spatial objects. Data storage and prediction are supported for packages 'terra', 'raster' and 'stars'.

Learn mlr3

- Book on mlr3: `https://mlr3book.mlr-org.com`
- Use cases and examples gallery: `https://mlr3gallery.mlr-org.com`
- Cheat Sheets: `https://github.com/mlr-org/mlr3cheatsheets`

mlr3 extensions

- Preprocessing and machine learning pipelines: **mlr3pipelines**
- Analysis of benchmark experiments: **mlr3benchmark**
- More classification and regression tasks: **mlr3data**
- Connector to OpenML: **mlr3oml**
- Solid selection of good classification and regression learners: **mlr3learners**
- Even more learners: `https://github.com/mlr-org/mlr3extralearners`
- Tuning of hyperparameters: **mlr3tuning**
- Hyperband tuner: **mlr3hyperband**

- Visualizations for many **mlr3** objects: **mlr3viz**
- Survival analysis and probabilistic regression: **mlr3proba**
- Cluster analysis: **mlr3cluster**
- Feature selection filters: **mlr3filters**
- Feature selection wrappers: **mlr3fselect**
- Interface to real (out-of-memory) data bases: **mlr3db**
- Performance measures as plain functions: **mlr3measures**

Suggested packages

- Parallelization framework: **future**
- Progress bars: **progressr**
- Encapsulated evaluation: **evaluate**, **callr** (external process)

Package Options

- `"mlr3.debug"`: If set to `TRUE`, parallelization via **future** is disabled to simplify debugging and provide more concise tracebacks. Note that results computed with debug mode enabled use a different seeding mechanism and are not reproducible.
- `"mlr3.allow_utf8_names"`: If set to `TRUE`, checks on the feature names are relaxed, allowing non-ascii characters in column names. This is an experimental and temporal option to pave the way for text analysis, and will likely be removed in a future version of the package. analysis.

Author(s)

Maintainer: Marc Becker <marcbecker@posteo.de> (ORCID)

Authors:

- Patrick Schratz <patrick.schratz@gmail.com> (ORCID)

References

Becker M, Schratz P (2024). *mlr3spatial: Support for Spatial Objects Within the 'mlr3' Ecosystem*. https://mlr3spatial.mlr-org.com, https://github.com/mlr-org/mlr3spatial.

See Also

Useful links:

- https://mlr3spatial.mlr-org.com
- https://github.com/mlr-org/mlr3spatial
- Report bugs at https://github.com/mlr-org/mlr3spatial/issues

`as_data_backend.stars` *Coerce to spatial DataBackend*

Description

Wraps a DataBackend around spatial objects. Currently these S3 methods are only alternative ways for writing `DataBackendRaster$new()`. They do not support coercing from other backends yet.

Usage

```
## S3 method for class 'stars'
as_data_backend(data, primary_key = NULL, ...)

## S3 method for class 'SpatRaster'
as_data_backend(data, primary_key = NULL, ...)

## S3 method for class 'RasterBrick'
as_data_backend(data, primary_key = NULL, ...)

## S3 method for class 'RasterStack'
as_data_backend(data, primary_key = NULL, ...)

## S3 method for class 'sf'
as_data_backend(data, primary_key = NULL, keep_rownames = FALSE, ...)
```

Arguments

`data`	(terra::SpatRaster) The input terra::SpatRaster.
`primary_key`	(`character(1)` | `integer()`) Name of the primary key column, or integer vector of row ids.
`...`	(any) Not used.
`keep_rownames`	(`logical(1)` | `character(1)`) If `TRUE` or a single string, keeps the row names of `data` as a new column. The column is named like the provided string, defaulting to `"..rownames"` for `keep_rownames == TRUE`. Note that the created column will be used as a regular feature by the task unless you manually change the column role. Also see `data.table::as.data.table()`.

Value

DataBackend.

`as_task_classif_st` *Convert to a Spatiotemporal Classification Task*

Description

Convert object to a TaskClassifST. This is a S3 generic, specialized for at least the following objects:

1. TaskClassifST: Ensure the identity.
2. `data.frame()` and DataBackend: Provides an alternative to the constructor of TaskClassifST.
3. sf::sf: Extracts spatial meta data before construction.
4. TaskRegr: Calls `convert_task()`.

Usage

```
as_task_classif_st(x, ...)

## S3 method for class 'TaskClassifST'
as_task_classif_st(x, clone = FALSE, ...)

## S3 method for class 'data.frame'
as_task_classif_st(
  x,
  target,
  id = deparse(substitute(x)),
  positive = NULL,
  coordinate_names,
  crs = NA_character_,
  coords_as_features = FALSE,
  label = NA_character_,
  ...
)

## S3 method for class 'DataBackend'
as_task_classif_st(
  x,
  target,
  id = deparse(substitute(x)),
  positive = NULL,
  coordinate_names,
  crs,
  coords_as_features = FALSE,
  label = NA_character_,
  ...
)

## S3 method for class 'sf'
```

```
as_task_classif_st(
  x,
  target = NULL,
  id = deparse(substitute(x)),
  positive = NULL,
  coords_as_features = FALSE,
  label = NA_character_,
  ...
)

## S3 method for class 'TaskRegrST'
as_task_classif_st(
  x,
  target = NULL,
  drop_original_target = FALSE,
  drop_levels = TRUE,
  ...
)
```

Arguments

`x`	(any) Object to convert.
`...`	(any) Additional arguments.
`clone`	(`logical(1)`) If `TRUE`, ensures that the returned object is not the same as the input `x`.
`target`	(`character(1)`) Name of the target column.
`id`	(`character(1)`) Id for the new task. Defaults to the (deparsed and substituted) name of the data argument.
`positive`	(`character(1)`) Level of the positive class. See TaskClassif.
`coordinate_names`	(`character(1)`) The column names of the coordinates in the data.
`crs`	(`character(1)`) Coordinate reference system. WKT2 or EPSG string.
`coords_as_features`	(`logical(1)`) If `TRUE`, coordinates are used as features.
`label`	(`character(1)`) Label for the new instance.

drop_original_target
: (logical(1))
If FALSE (default), the original target is added as a feature. Otherwise the original target is dropped.

drop_levels
: (logical(1))
If TRUE (default), unused levels of the new target variable are dropped.

Value

TaskClassifST

---

as_task_regr_st *Convert to a Spatiotemporal Regression Task*

Description

Convert object to a TaskRegrST. This is a S3 generic, specialized for at least the following objects:

1. TaskRegrST: Ensure the identity.
2. `data.frame()` and DataBackend: Provides an alternative to the constructor of TaskRegrST.
3. sf::sf: Extracts spatial meta data before construction.
4. TaskClassif: Calls `convert_task()`.

Usage

```
as_task_regr_st(x, ...)

## S3 method for class 'TaskRegrST'
as_task_regr_st(x, clone = FALSE, ...)

## S3 method for class 'data.frame'
as_task_regr_st(
  x,
  target,
  id = deparse(substitute(x)),
  coordinate_names,
  crs = NA_character_,
  coords_as_features = FALSE,
  label = NA_character_,
  ...
)

## S3 method for class 'DataBackend'
as_task_regr_st(
  x,
  target,
```

```
  id = deparse(substitute(x)),
  coordinate_names,
  crs,
  coords_as_features = FALSE,
  label = NA_character_,
  ...
)

## S3 method for class 'sf'
as_task_regr_st(
  x,
  target = NULL,
  id = deparse(substitute(x)),
  coords_as_features = FALSE,
  label = NA_character_,
  ...
)

## S3 method for class 'TaskClassifST'
as_task_regr_st(
  x,
  target = NULL,
  drop_original_target = FALSE,
  drop_levels = TRUE,
  ...
)
```

Arguments

x	(any) Object to convert.
...	(any) Additional arguments.
clone	(logical(1)) If TRUE, ensures that the returned object is not the same as the input x.
target	(character(1)) Name of the target column.
id	(character(1)) Id for the new task. Defaults to the (deparsed and substituted) name of the data argument.
coordinate_names	(character(1)) The column names of the coordinates in the data.
crs	(character(1)) Coordinate reference system. WKT2 or EPSG string.
coords_as_features	(logical(1)) If TRUE, coordinates are used as features.

`label` (`character(1)`)
Label for the new instance.

`drop_original_target` (`logical(1)`)
If `FALSE` (default), the original target is added as a feature. Otherwise the original target is dropped.

`drop_levels` (`logical(1)`)
If `TRUE` (default), unused levels of the new target variable are dropped.

Value

TaskRegrST

`DataBackendRaster` *DataBackend for Raster Objects*

Description

mlr3::DataBackend for terra::SpatRaster raster objects.

Read mode

There are two different ways the reading of values is performed internally:

- "Block mode" reads complete rows of the raster file and subsets the requested cells. This mode is faster than "cell mode" if the complete raster file is iterated over.
- "Cell mode" reads individual cells. This is faster than "block mode" if only a few cells are sampled.

"Block mode" is activated if `$data(rows)` is used with a increasing integer sequence e.g. `200:300`. If only a single cell is requested, "cell mode" is used.

Super class

`mlr3::DataBackend` -> `DataBackendRaster`

Active bindings

`rownames (integer())`
Returns vector of all distinct row identifiers, i.e. the contents of the primary key column.

`colnames (character())`
Returns vector of all column names.

`nrow (integer(1))`
Number of rows (observations).

`ncol (integer(1))`
Number of columns (variables).

`stack (SpatRaster)`
Raster stack.

Methods

Public methods:

- DataBackendRaster$new()
- DataBackendRaster$data()
- DataBackendRaster$head()
- DataBackendRaster$distinct()
- DataBackendRaster$missings()
- DataBackendRaster$coordinates()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
DataBackendRaster$new(data)
```

Arguments:

`data` (terra::SpatRaster)
The input terra::SpatRaster.

Method `data()`**:** Returns a slice of the raster in the specified format. Currently, the only supported formats is `"data.table"`.

The rows must be addressed as vector of cells indices, columns must be referred to via layer names. Queries for rows with no matching row id and queries for columns with no matching column name are silently ignored.

Rows are guaranteed to be returned in the same order as `rows`, columns may be returned in an arbitrary order. Duplicated row ids result in duplicated rows, duplicated column names lead to an exception.

Usage:

```
DataBackendRaster$data(rows, cols, data_format = "data.table")
```

Arguments:

`rows integer()`
Row indices. Row indices start with 1 in the upper left corner in the raster, increase from left to right and then from top to bottom. The last cell is in the bottom right corner and the row index equals the number of cells in the raster.

`cols character()`
Column names.

`data_format (character(1))`
Desired data format. Currently only `"data.table"` supported.

Method `head()`**:** Retrieve the first `n` rows.

Usage:

```
DataBackendRaster$head(n = 6L)
```

Arguments:

`n (integer(1))`
Number of rows.

Returns: `data.table::data.table()` of the first `n` rows.

Method `distinct()`**:** Returns a named list of vectors of distinct values for each column specified. If `na_rm` is `TRUE`, missing values are removed from the returned vectors of distinct values. Non-existing rows and columns are silently ignored.

Usage:

```
DataBackendRaster$distinct(rows, cols, na_rm = TRUE)
```

Arguments:

`rows integer()`
: Row indices. Row indices start with 1 in the upper left corner in the raster, increase from left to right and then from top to bottom. The last cell is in the bottom right corner and the row index equals the number of cells in the raster.

`cols character()`
: Column names.

`na_rm logical(1)`
: Whether to remove NAs or not.

Returns: Named `list()` of distinct values.

Method `missings()`**:** Returns the number of missing values per column in the specified slice of data. Non-existing rows and columns are silently ignored.

Usage:

```
DataBackendRaster$missings(rows, cols)
```

Arguments:

`rows integer()`
: Row indices. Row indices start with 1 in the upper left corner in the raster, increase from left to right and then from top to bottom. The last cell is in the bottom right corner and the row index equals the number of cells in the raster.

`cols character()`
: Column names.

Returns: Total of missing values per column (named `numeric()`).

Method `coordinates()`**:** Returns the coordinates of `rows`. If `rows` is missing, all coordinates are returned.

Usage:

```
DataBackendRaster$coordinates(rows)
```

Arguments:

`rows integer()`
: Row indices. Row indices start with 1 in the upper left corner in the raster, increase from left to right and then from top to bottom. The last cell is in the bottom right corner and the row index equals the number of cells in the raster.

Returns: `data.table::data.table()` of coordinates of `rows`.

DataBackendVector *DataBackend for Vector Objects*

Description

mlr3::DataBackend for sf::sf vector objects.

Super classes

`mlr3::DataBackend` -> `mlr3::DataBackendDataTable` -> `DataBackendVector`

Active bindings

`sfc` (sf::sfc)
Returns the sfc object.

Methods

Public methods:

- `DataBackendVector$new()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
DataBackendVector$new(data, primary_key)
```

Arguments:

`data` (`sf`)
A raster object.

`primary_key` (`character(1)|integer()`)
Name of the primary key column, or integer vector of row ids.

leipzig *Leipzig Land Cover Task*

Description

Point survey of land cover in Leipzig. Includes Sentinel-2 spectral bands and NDVI.

Source

Copernicus Sentinel Data (2021). Retrieved from Copernicus Open Access Hub and processed by European Space Agency.

Examples

```
if (requireNamespace("sf")) {
  library(sf)
  data("leipzig", package = "mlr3spatial")
  print(leipzig)
}
```

`predict_spatial` *Predict on Spatial Objects with mlr3 Learners*

Description

This function allows to directly predict mlr3 learners on various spatial objects.

Usage

```
predict_spatial(
  newdata,
  learner,
  chunksize = 200L,
  format = "terra",
  filename = NULL
)
```

Arguments

`newdata`	(terra::SpatRaster | `stars::stars` | sf::sf | `raster::RasterStack` | `raster::RasterBrick`). New data to predict on. All spatial data formats convertible by `as_data_backend()` are supported e.g. terra::SpatRaster or sf::sf.
`learner`	(Learner). Learner with trained model.
`chunksize`	(`integer(1)`) The chunksize determines in how many subparts the prediction task will be split into. The value can be roughly thought of as megabyte of a raster file on disk. For example, if a prediction on a 1 GB file would be carried out with `chunksize = 100L`, the prediction would happen in 10 chunks. The default of `chunksize = 1000L` might be a good compromise between speed and memory usage. If you find yourself running out of memory, reduce this value.
`format`	(`character(1)`) Output class of the resulting object. Accepted values are `"raster"`, `"stars"` and `"terra"` if the input is a raster. Note that when choosing something else than `"terra"`, the spatial object is converted into the respective format which might cause overhead both in runtime and memory allocation. For vector data only `"sf"` is supported.
`filename`	(`character(1)`) Path where the spatial object should be written to.

Value

Spatial object of class given in argument `format`.

Examples

```
library(terra, exclude = "resample")

# fit rpart on training points
task_train = tsk("leipzig")
learner = lrn("classif.rpart")
learner$train(task_train)

# load raster
stack = rast(system.file("extdata", "leipzig_raster.tif", package = "mlr3spatial"))

# predict land cover classes
pred = predict_spatial(stack, learner, chunksize = 1L)
```

TaskClassifST *Spatiotemporal Classification Task*

Description

This task specializes TaskClassif for spatiotemporal classification problems.

A spatial example task is available via `tsk("ecuador")`.

The coordinate reference system passed during initialization must match the one which was used during data creation, otherwise offsets of multiple meters may occur. By default, coordinates are not used as features. This can be changed by setting `coords_as_features = TRUE`.

Super classes

`mlr3::Task` -> `mlr3::TaskSupervised` -> `mlr3::TaskClassif` -> `TaskClassifST`

Active bindings

`crs (character(1))`
: Returns coordinate reference system of task.

`coordinate_names (character())`
: Returns coordinate names.

`coords_as_features (logical(1))`
: If `TRUE`, coordinates are used as features.

Methods

Public methods:

- TaskClassifST$new()
- TaskClassifST$coordinates()
- TaskClassifST$print()
- TaskClassifST$clone()

Method `new()`**:** Creates a new instance of this R6 class. The function `as_task_classif_st()` provides an alternative way to construct classification tasks.

Usage:

```
TaskClassifST$new(
  id,
  backend,
  target,
  positive = NULL,
  label = NA_character_,
  coordinate_names,
  crs = NA_character_,
  coords_as_features = FALSE,
  extra_args = list()
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the new instance.

`backend` (DataBackend)
: Either a DataBackend, or any object which is convertible to a DataBackend with `as_data_backend()`. E.g., am `sf` will be converted to a DataBackendDataTable.

`target` (`character(1)`)
: Name of the target column.

`positive` (`character(1)`)
: Only for binary classification: Name of the positive class. The levels of the target columns are reordered accordingly, so that the first element of `$class_names` is the positive class, and the second element is the negative class.

`label` (`character(1)`)
: Label for the new instance.

`coordinate_names` (`character(1)`)
: The column names of the coordinates in the data.

`crs` (`character(1)`)
: Coordinate reference system. WKT2 or EPSG string.

`coords_as_features` (`logical(1)`)
: If `TRUE`, coordinates are used as features.

`extra_args` (named `list()`)
: Named list of constructor arguments, required for converting task types via `convert_task()`.

Method `coordinates()`**:** Returns coordinates of observations.

Usage:

```
TaskClassifST$coordinates(row_ids = NULL)
```

Arguments:

`row_ids (integer())`
Vector of rows indices as subset of `task$row_ids`.

Returns: `data.table::data.table()`

Method `print()`**:** Print the task.

Usage:

```
TaskClassifST$print(...)
```

Arguments:

`...` Arguments passed to the `$print()` method of the superclass.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`TaskRegrST` *Spatiotemporal Regression Task*

Description

This task specializes TaskRegr for spatiotemporal regression problems.

A spatial example task is available via `tsk("cookfarm_mlr3")`.

The coordinate reference system passed during initialization must match the one which was used during data creation, otherwise offsets of multiple meters may occur. By default, coordinates are not used as features. This can be changed by setting `coords_as_features = TRUE`.

Super classes

`mlr3::Task` -> `mlr3::TaskSupervised` -> `mlr3::TaskRegr` -> `TaskRegrST`

Active bindings

`crs (character(1))`
Returns coordinate reference system of the task.

`coordinate_names (character())`
Returns coordinate names.

`coords_as_features (logical(1))`
If `TRUE`, coordinates are used as features.

Methods

Public methods:

- TaskRegrST$new()
- TaskRegrST$coordinates()
- TaskRegrST$print()
- TaskRegrST$clone()

Method `new()`**:** Creates a new instance of this R6 class. The function `as_task_regr_st()` provides an alternative way to construct classification tasks.

Usage:

```
TaskRegrST$new(
  id,
  backend,
  target,
  label = NA_character_,
  coordinate_names,
  crs = NA_character_,
  coords_as_features = FALSE,
  extra_args = list()
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the new instance.

`backend` (DataBackend)
: Either a DataBackend, or any object which is convertible to a DataBackend with `as_data_backend()`. E.g., am `sf` will be converted to a DataBackendDataTable.

`target` (`character(1)`)
: Name of the target column.

`label` (`character(1)`)
: Label for the new instance.

`coordinate_names` (`character(1)`)
: The column names of the coordinates in the data.

`crs` (`character(1)`)
: Coordinate reference system. WKT2 or EPSG string.

`coords_as_features` (`logical(1)`)
: If `TRUE`, coordinates are used as features.

`extra_args` (named `list()`)
: Named list of constructor arguments, required for converting task types via `convert_task()`.

Method `coordinates()`**:** Returns coordinates of observations.

Usage:

```
TaskRegrST$coordinates(row_ids = NULL)
```

Arguments:

`row_ids` (`integer()`)
: Vector of rows indices as subset of `task$row_ids`.

Returns: `data.table::data.table()`

Method `print()`**:** Print the task.

Usage:

```
TaskRegrST$print(...)
```

Arguments:

`...` Arguments passed to the `$print()` method of the superclass.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index