Package 'mlr3spatial'

March 6, 2022

Title Support for Spatial Objects Within the 'mlr3' Ecosystem

Version 0.1.2

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 (>= 0.12.0), R (>= 3.1.0)

Imports checkmate (>= 2.0.0), data.table (>= 1.14.0), lgr (>= 0.4.2), methods, mlr3misc, R6 (>= 2.5.0), terra (>= 1.5-12), utils

Suggests bench, e1071, future, future.callr, knitr, mlr3learners (>= 0.4.5), mlr3tuning, paradox, ranger, raster, rgdal, rmarkdown, rpart, sf, stars (>= 0.5-5), testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Author Marc Becker [aut, cre] (<https://orcid.org/0000-0002-8115-0400>), Patrick Schratz [aut] (<https://orcid.org/0000-0003-0748-6624>)

Maintainer Marc Becker <marcbecker@posteo.de>

Repository CRAN

Date/Publication 2022-03-06 18:50:02 UTC

R topics documented:

`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 (2022). *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 DataBackendRaster*

Description

Wraps a DataBackend around spatial objects. Currently this is only a synonym for `DataBackendRaster$new()` and does not support coercing from other backends.

Wraps a DataBackend around spatial objects. Currently this is only a synonym for `DataBackendVector$new()` and does not support coercing from other backends.

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 'Raster'
as_data_backend(data, primary_key = NULL, ...)

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

Arguments

`data`	(Spatial object) Supported objects: • raster::brick • stars::st_as_stars • terra::SpatRaster
`primary_key`	(`character(1)` | `integer()`) Name of the primary key column, or integer vector of row ids.
`...`	(`any`) Not used.

Value

DataBackend.

DataBackend.

`DataBackendRaster` *DataBackend for raster objects*

Description

mlr3::DataBackend for raster objects:

- raster::brick
- stars::st_as_stars
- terra::SpatRaster

The DataBackend can be constructed using the spatial objects listed above. Internally terra is used for processing operations.

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`

Public fields

`compact_seq logical(1)`
: If `TRUE`, row ids are a natural sequence from 1 to `nrow(data)` (determined internally). In this case, row lookup uses faster positional indices instead of equi joins.

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()`

Method `new()`**:** Creates a backend for a raster objects.

Usage:

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

Arguments:

`data` (Spatial object)
: Supported objects:
  - raster::brick
  - stars::st_as_stars
  - 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()`).

`DataBackendVector` *DataBackend for vector objects*

Description

A mlr3::DataBackend for vector objects (only package **sf** is supported).

Super class

`mlr3::DataBackend` -> `DataBackendVector`

Public fields

`compact_seq` `logical(1)`
: If `TRUE`, row ids are a natural sequence from 1 to `nrow(data)` (determined internally). In this case, row lookup uses faster positional indices instead of equi joins.

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, including the primary key column.

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

`ncol (integer(1))`
Number of columns (variables), including the primary key column.

`geometry (integer(1))`
Returns the sf geometry.

Methods

Public methods:

- `DataBackendVector$new()`
- `DataBackendVector$data()`
- `DataBackendVector$head()`
- `DataBackendVector$distinct()`
- `DataBackendVector$missings()`

Method `new()`**:** Creates a backend for spatial vector objects.

Usage:

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

Arguments:

`data (sf)`
A raster object.

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

Method `data()`**:** Returns a slice of the data in the specified format. Currently, the only supported formats are `"data.table"` and `"Matrix"`. The rows must be addressed as vector of primary key values, columns must be referred to via column 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:

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

Arguments:

`rows integer()`
Row indices.

`cols character()`
Column names.

`data_format (character(1))`
Desired data format, e.g. `"data.table"` or `"Matrix"`.

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

Usage:

```
DataBackendVector$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:

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

Arguments:

`rows` `integer()`
Row indices.

`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:

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

Arguments:

`rows` `integer()`
Row indices.

`cols` `character()`
Column names.

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

`demo_stack_rasterbrick`

Generate a demo 'raster::brick'

Description

Generates a square demo raster::brick object with options to set the size (on disk) and number of layers.

Usage

```
demo_stack_rasterbrick(size = 50, layers = 5)
```

Arguments

`size`	[integer(1)] Size of raster stack in megabyte (disk space).
`layers`	[integer(1)] Number of layers.

Examples

```
if (mlr3misc::require_namespaces("raster", quietly = TRUE)) {
  demo_stack_rasterbrick(size = 5, layers = 2)
}
```

`demo_stack_spatraster` *Generate a demo 'terra::SpatRaster'*

Description

Generates a square demo terra::SpatRaster object with options to set the size (on disk) and number of layers.

Usage

```
demo_stack_spatraster(size = 50, layers = 5)
```

Arguments

`size`	[integer(1)] Size of raster stack in megabyte (disk space).
`layers`	[integer(1)] Number of layers.

Examples

```
demo_stack_spatraster(size = 5, layers = 2)
```

`predict_spatial` *Predict on spatial objects with mlr3 learners*

Description

This function allows to directly predict mlr3 learners on various spatial objects (see section "Supported Spatial Classes"). It returns an mlr3::Prediction object and (optionally) the same object that was used for the prediction.

Usage

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

Arguments

`task`	(Task).
`learner`	(Learner).
`chunksize`	[integer] 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] Output class of the resulting object. Accepted values are `"raster"`, `"stars"` and `"terra"` if the input is a DataBackendRaster. 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 a DataBackendVector, the output class will always be sf::sf.
`filename`	[character] Path where the spatial object should be written to.

Details

When parallelizing the prediction via future, plan `"multisession"` will not work due to external pointers within the spatial object. If the execution platform is UNIX-based, `plan("multicore")` is recommended. For Windows users, `plan(future.callr::callr)` might be an alternative.

Value

Spatial object of class given in argument `format`.

Examples

```
stack = demo_stack_spatraster(size = 1)
value = data.table::data.table(ID = c(0, 1), y = c("negative", "positive"))
terra::setCats(stack, layer = "y", value = value)

# create backend
backend = as_data_backend(stack)
task = as_task_classif(backend, target = "y", positive = "positive")
# train
learner = lrn("classif.featureless")
learner$train(task, row_ids = sample(1:task$nrow, 50))
ras = predict_spatial(task, learner)
ras
```

Index