Package ‘modelenv’

March 9, 2023

Title  Provide Tools to Register Models for Use in 'tidymodels'
Version  0.1.1
Description  An developer focused, low dependency package in 'tidymodels' that provides functions to register how models are to be used. Functions to register models are complimented with accessor functions to retrieve registered model information to aid in model fitting and error handling.
License  MIT + file LICENSE
Encoding  UTF-8
RoxygenNote  7.2.3
Imports  glue, rlang, tibble, vctrs
Suggests  covr, testthat (>= 3.0.0)
Config/Needs/website  tidyverse/tidytemplate
Config/testthat/edition  3
URL  https://github.com/tidymodels/modelenv,
     http://modelenv.tidymodels.org/
BugReports  https://github.com/tidymodels/modelenv/issues
NeedsCompilation  no
Author  Emil Hvitfeldt [aut, cre] (https://orcid.org/0000-0002-0679-1945), Posit Software, PBC [cph, fnd]
Maintainer  Emil Hvitfeldt <emil.hvitfeldt@posit.co>
Repository  CRAN
Date/Publication  2023-03-08 23:10:02 UTC

R topics documented:

  check_spec_mode_engine_val ........................................... 2
  get_model_env .......................................................... 3
  is_unsupervised_fit .................................................... 4
  is_unsupervised_spec .................................................. 4
check_spec_mode_engine_val

Error handling for unknown mode

Description

Checks that a given model, mode, engine combination have been registered.

Usage

check_spec_mode_engine_val(model, mode, eng)

Arguments

model Character of specific model
mode Character of specific mode
eng Character of specific engine

Value

An error

Examples

library(rlang)
tmp <- catch_cnd(check_spec_mode_engine_val("turtle", "partition", "vegan"))
**get_model_env**

**Working with the modelenv model environment**

**Description**

These functions read and write to the environment where the package stores information about model specifications.

**Usage**

```r
get_model_env()
get_from_env(items)
set_env_val(name, value)
```

**Arguments**

- `items` A character string of objects in the model environment.
- `name` A single character value for a new symbol in the model environment.
- `value` A single value for a new value in the model environment.

**Value**

The modelenv environment

**Examples**

```r
# Access the model data:
current_code <- get_model_env()
ls(envir = current_code)
get_from_env("models")
get_from_env("modes")
get_from_env("example")
set_env_val("example", 4)
get_from_env("example")
```
**is_unsupervised_fit**  
*Determine object is has class unsupervised_fit*

**Description**
Determine object is has class unsupervised_fit

**Usage**
```r
is_unsupervised_fit(x)
```

**Arguments**
- `x` an model fit object

**Value**
A single logical indicating input has class "unsupervised_fit"

**See Also**
- `new_unsupervised_fit()`

**Examples**
```r
model_fit <- list(letters)
is_unsupervised_fit(model_fit)
model_fit <- new_unsupervised_fit(model_fit)
is_unsupervised_fit(model_fit)
```

---

**is_unsupervised_spec**  
*Determine object is has class unsupervised_spec*

**Description**
Determine object is has class unsupervised_spec

**Usage**
```r
is_unsupervised_spec(x)
```

**Arguments**
- `x` an model specification
Value
A single logical indicating input has class "unsupervised_spec"

See Also
new_unsupervised_spec()

Examples
model_spec <- list(letters)
is_unsupervised_spec(model_spec)
model_spec <- new_unsupervised_spec(model_spec)
is_unsupervised_spec(model_spec)

Description
Give object unsupervised fit class

Usage
new_unsupervised_fit(x)

Arguments
x an model fit object

Value
x with added class "unsupervised_fit"

See Also
is_unsupervised_fit()

Examples
model_fit <- list(letters)
model_fit <- new_unsupervised_fit(model_fit)
model_fit
new_unsupervised_spec  Give object unsupervised specification class

Description
Give object unsupervised specification class

Usage
new_unsupervised_spec(x)

Arguments
x  an model specification

Value
x with added class "unsupervised_spec"

See Also
is_unsupervised_spec()

Examples
model_spec <- list(letters)
model_spec <- new_unsupervised_spec(model_spec)
model_spec

set_dependency  Register Dependency for Model

Description
This function is used to register a mode for a model, engine, and mode combination.

Usage
set_dependency(model, mode, eng, pkg)
get_dependency(model)
### set_encoding

Register Encoding Options for Model

#### Description

This function is used to register encoding information for a model, engine, and mode combination.

#### Usage

```r
set_encoding(model, mode, eng, options)

get_encoding(model)
```
Arguments

model  A single character string for the model type (e.g. "k_means", etc).
mode   A single character string for the model mode (e.g. "partition").
eng    A single character string for the model engine.
options A list of options for engine-specific preprocessing encodings. See Details below.

Details

The list passed to options needs the following values:

- **predictor_indicators** describes whether and how to create indicator/dummy variables from factor predictors. There are three options: "none" (do not expand factor predictors), "traditional" (apply the standard `model.matrix()` encodings), and "one_hot" (create the complete set including the baseline level for all factors).

- **compute_intercept** controls whether `model.matrix()` should include the intercept in its formula. This affects more than the inclusion of an intercept column. With an intercept, `model.matrix()` computes dummy variables for all but one factor level. Without an intercept, `model.matrix()` computes a full set of indicators for the first factor variable, but an incomplete set for the remainder.

- **remove_intercept** removes the intercept column after `model.matrix()` is finished. This can be useful if the model function (e.g. `lm()`) automatically generates an intercept.

- **allow_sparse_x** specifies whether the model can accommodate a sparse representation for predictors during fitting and tuning.

Value

A tibble

Examples

```r
set_new_model("shallow_learning_model")
set_model_mode("shallow_learning_model", "partition")
set_model_engine("shallow_learning_model", "partition", "stats")

set_encoding(
  model = "shallow_learning_model",
  mode = "partition",
  eng = "stats",
  options = list(
    predictor_indicators = "traditional",
    compute_intercept = TRUE,
    remove_intercept = TRUE,
    allow_sparse_x = FALSE
  )
)

get_encoding("shallow_learning_model")
get_encoding("shallow_learning_model")$value
```
Description

This function is used to register a fit method for a model, engine, and mode combination.

Usage

set_fit(model, mode, eng, value)

get_fit(model)

Arguments

- model: A single character string for the model type (e.g. "k_means", etc).
- mode: A single character string for the model mode (e.g. "partition").
- eng: A single character string for the model engine.
- value: A list of values, described in the Details.

Details

The list passed to value needs the following values:

- **interface**: is a single character value that could be "formula", "data.frame", or "matrix". This defines the type of interface used by the underlying fit function (stats::lm, in this case). This helps the translation of the data to be in an appropriate format for the that function.

- **protect**: is an optional list of function arguments that should not be changeable by the user. In this case, we probably don’t want users to pass data values to these arguments (until the fit() function is called).

- **func**: is the package and name of the function that will be called. If you are using a locally defined function, only fun is required.

- **defaults**: is an optional list of arguments to the fit function that the user can change, but whose defaults can be set here. This isn’t needed in this case, but is described later in this document.

Value

A tibble

Examples

```r
set_new_model("shallow_learning_model")
set_model_mode("shallow_learning_model", "partition")
set_model_engine("shallow_learning_model", "partition", "stats")
set_fit(
```
set_model_arg

Description
This function is used to register argument information for a model and engine combination.

Usage
set_model_arg(model, eng, exposed, original, func, has_submodel)

get_model_arg(model, eng)

Arguments
model  A single character string for the model type (e.g. "k_means", etc).
eng    A single character string for the model engine.
exposed A single character string for the "harmonized" argument name that the modeling
         function exposes.
original A single character string for the argument name that underlying model function
         uses.
func    A named character vector that describes how to call a function. func should
         have elements pkg and fun. The former is optional but is recommended and the
         latter is required. For example, c(pkg = "stats", fun = "lm") would be used
         to invoke the usual linear regression function. In some cases, it is helpful to use
         c(fun = "predict") when using a package's predict method.
has_submodel A single logical for whether the argument can make predictions on multiple
               submodels at once.

Details
This function needs to be called once for each argument that you are exposing.
set_model_engine

Value
A tibble

Examples

```r
set_new_model("shallow_learning_model")
set_model_mode("shallow_learning_model", "partition")
set_model_engine("shallow_learning_model", "partition", "stats")

set_model_arg(
  model = "shallow_learning_model",
  eng = "stats",
  exposed = "method",
  original = "method",
  func = list(pkg = "stats", fun = "lm"),
  has_submodel = FALSE
)

get_model_arg("shallow_learning_model", "stats")
get_model_arg("shallow_learning_model", "stats")$func
```

---

**set_model_engine**  
*Register Engine for Model*

**Description**

This function is used to register a mode for a model and mode combination.

**Usage**

```r
set_model_engine(model, mode, eng)
```

**Arguments**

- `model` A single character string for the model type (e.g. "k_means", etc).
- `mode` A single character string for the model mode (e.g. "partition").
- `eng` A single character string for the model engine.

**Details**

This function will error if called multiple times with the same arguments. As you should only have one unique model, mode, eng combination.

**Value**

NULL invisibly
set_model_mode

Examples

set_new_model("shallow_learning_model")
set_model_mode("shallow_learning_model", "partition")

get_from_env("shallow_learning_model")
set_model_engine("shallow_learning_model", "partition", "stats")
get_from_env("shallow_learning_model")

---

set_model_mode | Register Mode for Model
---

Description

This function is used to register a mode for a model.

Usage

set_model_mode(model, mode)

Arguments

- **model**: A single character string for the model type (e.g. "k_means", etc).
- **mode**: A single character string for the model mode (e.g. "partition").

Details

This function can be called multiple times without error. This becomes valuable when multiple packages add the same mode to a model. Having both packages use set_model_mode() avoids having one package depend on the other.

Value

NULL invisibly

Examples

set_new_model("shallow_learning_model")
get_from_env("shallow_learning_model_modes")
set_model_mode("shallow_learning_model", "partition")
get_from_env("shallow_learning_model_modes")
**set_new_model**

Register New Model

Description

This function is used to register new types of models.

Usage

```r
set_new_model(model)
```

Arguments

- **model**: A single character string for the model type (e.g. "k_means", etc).

Details

This function is available for users to add their own models or engines (in a package or otherwise) so that they can be accessed using packages that use modelenv.

Value

NULL invisibly

Examples

```r
set_new_model("shallow_learning_model")
```

---

**set_pred**

Register Prediction Method for Model

Description

This function is used to register prediction method information for a model, mode, and engine combination.

Usage

```r
set_pred(model, mode, eng, type, value)
```

```r
get_pred_type(model, type)
```
Arguments

- `model`  A single character string for the model type (e.g. "k_means", etc).
- `mode`  A single character string for the model mode (e.g. "partition").
- `eng`  A single character string for the model engine.
- `type`  A single character value for the type of prediction. Possible values are: `cluster` and `raw`.
- `value`  A list of values, described in the Details.

Details

The list passed to `value` needs the following values:

- `pre` and `post` are optional functions that can preprocess the data being fed to the prediction code and to postprocess the raw output of the predictions. These won't be needed for this example, but a section below has examples of how these can be used when the model code is not easy to use. If the data being predicted has a simple type requirement, you can avoid using a `pre` function with the `args` below.

- `func` is the prediction function (in the same format as above). In many cases, packages have a predict method for their model's class but this is typically not exported. In this case (and the example below), it is simple enough to make a generic call to `predict()` with no associated package.

- `args` is a list of arguments to pass to the prediction function. These will most likely be wrapped in `rlang::expr()` so that they are not evaluated when defining the method. For mda, the code would be `predict(object, newdata, type = "class")`. What is actually given to the function is the model fit object, which includes a sub-object called `fit()` that houses the mda model object. If the data need to be a matrix or data frame, you could also use `newdata = quote(as.data.frame(newdata))` or similar.

Value

A tibble

Examples

```r
set_new_model("shallow_learning_model")
set_model_mode("shallow_learning_model", "partition")
set_model_engine("shallow_learning_model", "partition", "stats")

set_pred(
  model = "shallow_learning_model",
  eng = "stats",
  mode = "partition",
  type = "cluster",
  value = list(
    pre = NULL,
    post = NULL,
    func = c(fun = "predict"),
)
```
stop_incompatible_mode

Error handling for incompatible modes

Description

Error handling for incompatible modes

Usage

stop_incompatible_mode(spec_modes, eng = NULL, model = NULL)

Arguments

- spec_modes: Character vector of modes
- eng: Character of specific engine
- model: Character of specific model

Value

An error

Examples

library(rlang)
tmp <- catch_cnd(stop_incompatible_mode("partition"))
Index

check_spec_mode_engine_val, 2
get_dependency (set_dependency), 6
get_encoding (set_encoding), 7
get_fit (set_fit), 9
get_from_env (get_model_env), 3
get_model_arg (set_model_arg), 10
get_model_env, 3
get_pred_type (set_pred), 13
is_unsupervised_fit, 4
is_unsupervised_fit(), 5
is_unsupervised_spec, 4
is_unsupervised_spec(), 6
new_unsupervised_fit, 5
new_unsupervised_fit(), 4
new_unsupervised_spec, 6
new_unsupervised_spec(), 5
set_dependency, 6
set_encoding, 7
set_env_val (get_model_env), 3
set_fit, 9
set_model_arg, 10
set_model_engine, 11
set_model_mode, 12
set_new_model, 13
set_pred, 13
stop_incompatible_mode, 15