

Package ‘struct’

March 30, 2021

Type Package

Title Statistics in R Using Class-based Templates

Version 1.2.0

Description Defines and includes a set of class-based templates for developing and implementing data processing and analysis workflows, with a strong emphasis on statistics and machine learning. The templates can be used and where needed extended to 'wrap' tools and methods from other packages into a common standardised structure to allow for effective and fast integration. Model objects can be combined into sequences, and sequences nested in iterators using overloaded operators to simplify and improve readability of the code. STATistics Ontology (STATO) has been integrated and implemented to provide standardised definitions for methods, inputs and outputs wrapped using the class-based templates.

License GPL-3

Encoding UTF-8

LazyData true

Collate 'generics.R' 'struct_class.R' 'parameter_class.R'
'chart_class.R' 'stato_class.R' 'DatasetExperiment_class.R'
'entity_class.R' 'entity_stato_class.R' 'enum_class.R'
'enum_stato_class.R' 'output_class.R' 'model_class.R'
'example_objects.R' 'model_list_class.R' 'metric_class.R'
'iterator_class.R' 'optimiser_class.R' 'preprocess_class.R'
'resampler_class.R' 'struct.R' 'struct_templates.R'

RoxygenNote 7.1.1

Depends R (>= 4.0)

Suggests testthat, rstudioapi, rmarkdown, covr, BiocStyle, openxlsx,
ggplot2, magick

VignetteBuilder knitr

Imports methods, ontologyIndex, datasets, graphics, stats, utils,
knitr, SummarizedExperiment, S4Vectors

biocViews WorkflowStep

git_url <https://git.bioconductor.org/packages/struct>

git_branch RELEASE_3_12

git_last_commit 47c7614

git_last_commit_date 2020-10-27

Date/Publication 2021-03-29

Author Gavin Rhys Lloyd [aut, cre],
Ralf Johannes Maria Weber [aut]

Maintainer Gavin Rhys Lloyd <g.r.lloyd@bham.ac.uk>

R topics documented:

as.code	3
as.DatasetExperiment	4
as.DatasetExperiment,SummarizedExperiment-method	4
as.SummarizedExperiment	5
as.SummarizedExperiment,DatasetExperiment-method	5
as_data_frame	6
calculate	6
chart	7
chart_names	8
chart_plot	9
citations	10
DatasetExperiment	10
entity_stato	11
enum	12
enum_stato	13
example_chart	14
example_iterator-class	15
example_model	16
export_xlsx	17
iris_DatasetExperiment	17
is_output	18
is_param	18
libraries	19
max_length	20
model	21
models	23
model_apply	24
model_predict	25
model_reverse	25
model_seq	26
model_train	28
new_struct	29
optimiser	29
output_ids	30
output_list	30
output_name	31
output_obj	32
output_value	33
param_ids	34
param_list	34
param_name	35
param_obj	36
param_value	37
predicted	38

predicted_name	38
preprocess	39
resampler	39
result	40
result_name	41
run	41
seq_in	43
set_obj_method	44
set_obj_show	45
set_struct_obj	46
stato_id	47
struct	48
struct_class	49
struct_class-class	49
struct_template	50
test_metric-class	51
\$.struct_class-method	51
\$<-,struct_class-method	52

Index 53

as.code	<i>Convert to code</i>
---------	------------------------

Description

Prints a block of code that can be used to replicate the input object.

Usage

```
as.code(M, start = "M = ", mode = "compact")

## S4 method for signature 'model'
as.code(M, start = "M = ", mode = "compact")

## S4 method for signature 'model_seq'
as.code(M, start = "M = ", mode = "compact")

## S4 method for signature 'iterator'
as.code(M, start = "M = ", mode = "compact")
```

Arguments

M	a struct model, model_seq or iterator object
start	text prepended to the code. Default is "M = "
mode	"compact" will use the least amount of lines, "expanded" will put each object and input on a new line. "neat" will produce an output somewhere between "compact" and "extended".

Value

A string of code to reproduce the input object.
 a string of code to reproduce the model
 a string of code to reproduce the model sequence
 a string of code to reproduce the iterator

Examples

```
M = example_model(value_1 = 10)
as.code(M)
M = example_model()
as.code(M)
M = example_model()
as.code(M)
M = example_model()
as.code(M)
```

as.DatasetExperiment *Convert a SummarizedExperiment to DatasetExperiment*

Description

Converts a SummarizedExperiment to DatasetExperiment. The assay data is transposed, and colData and rowData switched to match. struct specific slots such as "name" and "description" are extracted from the metaData.

Usage

```
as.DatasetExperiment(obj)
```

Arguments

obj a SummarizedExperiment object

Value

a DatasetExperiment object

as.DatasetExperiment, SummarizedExperiment-method
Convert a SummarizedExperiment to DatasetExperiment

Description

The assay data is transposed, and colData and rowData switched to match. struct specific slots such as "name" and "description" are extracted from the metaData if available. NB Any additional metadata will be lost during this conversion.

Usage

```
## S4 method for signature 'SummarizedExperiment'  
as.DatasetExperiment(obj)
```

Arguments

obj a SummarizedExperiment object

Value

a DatasetExperiment object

as.SummarizedExperiment

Convert a DatasetExperiment to a SummarizedExperiment

Description

Converts a DatasetExperiment to SummarizedExperiment. The assay data is transposed, and col-Data and rowData switched to match. struct specific slots such as "name" and "description" are stored in the metaData.

Usage

```
as.SummarizedExperiment(obj)
```

Arguments

obj a DatasetExperiment object

Value

a SummarizedExperiment object

as.SummarizedExperiment, DatasetExperiment-method

Convert a DatasetExperiment to SummarizedExperiment

Description

Converts a DatasetExperiment to SummarizedExperiment. The assay data is transposed, and col-Data and rowData switched to match. struct specific slots such as "name" and "description" are stored in the metaData.

Usage

```
## S4 method for signature 'DatasetExperiment'  
as.SummarizedExperiment(obj)
```

Arguments

obj a DatasetExperiment object

Value

a SummarizedExperiment object

as_data_frame	<i>convert to data.frame</i>
---------------	------------------------------

Description

Most often used with univariate statistics to gather all the different outputs in a consistent format.

Usage

```
as_data_frame(M, ...)
```

Arguments

M a struct object
 ... other inputs passed through this function

Value

a data.frame containing outputs from an object

calculate	<i>Calculate metric</i>
-----------	-------------------------

Description

A class for metrics to assess performance of e.g. models, iterators. Not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

Usage

```
calculate(obj, ...)

value(obj)

value(obj) <- value

max_length(obj) <- value

metric(...)

## S4 method for signature 'metric'
calculate(obj, Y, Yhat)
```

```
## S4 method for signature 'metric'  
value(obj)  
  
## S4 replacement method for signature 'metric'  
value(obj) <- value
```

Arguments

obj	a metric object
...	named slots and their values.
value	value
Y	the true class labels
Yhat	the predicted class labels

Value

value the calculated value of a metric
a metric object

Examples

```
MET = metric()  
calculate(MET)  
MET = metric()  
M = metric()  
calculate(M,Y,Yhat)  
MET = metric()  
value(MET)  
MET = metric()  
value(MET) = 10
```

chart

Constructor for struct chart objects

Description

A base class in the **struct** package. Should not be called directly.

Usage

```
chart(...)
```

Arguments

... named slots and their values that get passed to struct_class

Details

The `chart` class provides a template for figures, charts and plots associated with other objects. For example, a `DatasetExperiment` object could have a histogram plotted for a specified column.

Charts can have parameters but not outputs (other than the figure itself), as chart objects are not intended to be used for calculations. The `chart_plot` method can be used to display a chart for an object, and `chart_names` can be used to list all chart objects associated with an object.

Classes that inherit the `stato` class have STATO integration enabled, allowing `stato_id` to be set and formal names and descriptions pulled from the STATO ontology database.

Value

a chart object

a `struct_class` object

Examples

```
C = example_chart()
```

<code>chart_names</code>	<i>chart names</i>
--------------------------	--------------------

Description

Returns a list of valid charts for a struct object

Usage

```
chart_names(obj, ret = "char")
```

```
## S4 method for signature 'struct_class'
chart_names(obj, ret = "char")
```

Arguments

`obj` An object derived from the `struct_class` object

`ret` A string indicating whether a list of objects ('obj') or a list of chart names ('char') is returned. 'char' is default.

Details

The `chart_names` method searches chart objects that specify the input object type as an input.

Value

list of chart names, or a list of chart objects

Methods (by class)

- `struct_class`:

Examples

```
M = example_model()
chart_names(M) # 'example_chart'
chart_names(M, 'char') # as above
chart_names(M, 'obj') # returns a list of chart objects
```

chart_plot

chart_plot

Description

Plots a chart object

Usage

```
chart_plot(obj, dobj, ...)
```

S4 method for signature 'chart,ANY'

```
chart_plot(obj, dobj)
```

Arguments

obj	A chart object
dobj	An object derived from struct_class
...	optional inputs

Details

The optional optional inputs depend on the input object/chart, but might include an additional dataset object or a second model object, for example.

Value

a plot object

Methods (by class)

- obj = chart, dobj = ANY:

Examples

```
C = example_chart()
chart_plot(C, iris_DatasetExperiment())
```

citations	<i>Citations for an object</i>
-----------	--------------------------------

Description

All struct objects have a "citations" slot, which is a character array of references relevant to the object. The `citations` method gathers citations from an object and all struct objects that it inherits to generate a complete list.

Usage

```
citations(obj)

## S4 method for signature 'struct_class'
citations(obj)
```

Arguments

`obj` a struct object

Value

a character array of citations

Examples

```
D = iris_DatasetExperiment()
D$citations # the list specifically defined for this object
citations(D) # the list for this object and all inherited ones
```

DatasetExperiment	<i>DatasetExperiment class</i>
-------------------	--------------------------------

Description

An object for holding raw data and associated meta data

Usage

```
DatasetExperiment(
  data = data.frame(),
  sample_meta = data.frame(),
  variable_meta = data.frame(),
  ...
)

## S4 method for signature 'DatasetExperiment'
x$name

## S4 replacement method for signature 'DatasetExperiment'
x$name <- value
```

Arguments

data	A data frame with samples in rows and features in columns
sample_meta	A data frame with samples in rows and meta data in columns
variable_meta	A data frame with features in rows and meta data in columns
...	named slot values to pass through to struct_class
x	A DatasetExperiment object
name	DatasetExperiment slot to get/set
value	the value to assign to the named slot

Details

The DatasetExperiment object is an extension of the SummarizedExperiment object from the SummarizedExperiment package (found on Bioconductor). It incorporates the basic functionality of struct objects, containing fields such as Description, Name and Type with features of SummarizedExperiment such as subsetting.

There are some important differences between DatasetExperiment and SummarizedExperiment:

- In DatasetExperiment data is stored as Samples (rows) x Features (columns)
- DatasetExperiment currently only supports a single assay
- length(DatasetExperiment) returns the number of samples

Value

DatasetExperiment

Slots

name Name of the dataset
description Brief description of the dataset
type The type of dataset e.g. single_block

entity_stato *entity_stato class*

Description

A base class in the **struct** package. Should not be called directly.

Usage

```
entity_stato(
  name,
  description = character(0),
  type = "character",
  value = NULL,
  max_length = Inf,
  stato_id
)
```

Arguments

name	the name of the object
description	a description of the object
type	the type of the struct object
value	The value of the parameter/outputs
max_length	Maximum length of value vector (default 1)
stato_id	The STATO ID for the entity

Details

Extends the `entity` class to include `stato` functionality.

Value

an `entity_stato` object

See Also

Refer to [entity](#) and [stato](#) for further info.

Examples

```
E = entity_stato(
  name = 'example',
  description = 'this is an example',
  type = 'numeric',
  value = 1,
  stato_id='XYZ000001'
)
```

enum

Enum objects

Description

A base class in the `struct` package. Not normally called directly.

Usage

```
enum(
  name,
  description = character(0),
  type = "character",
  value = character(0),
  max_length = 1,
  allowed
)

## S4 replacement method for signature 'enum'
value(obj) <- value
```

Arguments

name	the name of the object
description	a description of the object
type	the type of the struct object
value	value of the enum
max_length	Maximum length of value vector (default 1)
allowed	A list of allowed values
obj	an enum object

Details

An enum object is a special type of entity object that ensures the value must be one from a list of allowed values.

Enum objects are usually defined in the prototype of another object, but can be extracted using `param_obj` and `output_obj`.

Value

an enum object

Examples

```
# Create a new enum object
E = enum(
  name = 'example',
  description = 'this is an example',
  type = 'character',
  value = 'hello',
  allowed = c('hello', 'world')
)

# Get/set the value of the entity object
value(E)
value(E) = 'world'
```

enum_stato

enum_stato class

Description

A base class in the **struct** package. Should not be called directly.

Usage

```
enum_stato(
  name,
  description = character(0),
  type = "character",
  value = character(0),
  max_length = 1,
```

```

    allowed,
    stato_id
)

```

Arguments

name	the name of the object
description	a description of the object
type	the type of the struct object
value	The value of the parameter/outputs
max_length	Maximum length of value vector (default 1)
allowed	A list of allowed values
stato_id	The STATO ID for the entity

Details

Extends the enum class to include stato functionality.

Value

an enum_stato object

See Also

Refer to [enum](#) and [stato](#) for further info.

Examples

```

E = enum_stato(
  name='example',
  allowed=list('choice_1','choice_2'),
  value='choice_1',
  type='character',
  stato_id='XYZ000001'
)

```

example_chart

example chart object

Description

an example of a chart object for documentation purposes

Usage

```
example_chart(...)
```

```
## S4 method for signature 'example_chart,example_model'
chart_plot(obj, dobj)
```

Arguments

... named slots and their values.
obj a chart object
dobj a example_model object

Value

a chart object

Examples

```
C = example_chart()
chart_plot(C,example_model())
```

example_iterator-class

Example iterator

Description

An example iterator for testing
runs the example iterator, which just returns a value of 3.142

Usage

```
## S4 method for signature 'example_iterator,DatasetExperiment,metric'
run(I, D, MET)
```

Arguments

I example_iterator object
D dataset object
MET metric object

Value

test iterator object
dataset object

Examples

```
I = example_iterator()

I = example_iterator()
D = iris_DatasetExperiment()
MET = metric()
I = run(I,D,MET)
```

 example_model

Example model

Description

An example model for testing. Training this model adds value_1 to a data set, and prediction using this model adds value_2.

trains the example model, which adds value_1 to the raw data of a dataset

predicts using the example model, which adds value_2 to the raw data of a dataset

Usage

```
example_model(value_0 = 0, value_1 = 10, value_2 = 20, ...)
```

```
## S4 method for signature 'example_model,DatasetExperiment'
model_train(M, D)
```

```
## S4 method for signature 'example_model,DatasetExperiment'
model_predict(M, D)
```

Arguments

value_0	a numeric value
value_1	a numeric value
value_2	a numeric value
...	named slots and their values.
M	A struct model object
D	A DatasetExperiment object

Value

modified example_model object

dataset object

dataset object

Examples

```
M = example_model()
M = example_model(value_1 = 10, value_2 = 20)
D = iris_DatasetExperiment()
M = example_model(value_1 = 10, value_2 = 20)
M = model_train(M,D)
D = iris_DatasetExperiment()
M = example_model(value_1 = 10, value_2 = 20)
M = model_predict(M,D)
```

export_xlsx	<i>write a dataset object to file</i>
-------------	---------------------------------------

Description

Exports a dataset object to an excel file with sheets for data, sample_meta and variable_meta

Usage

```
export_xlsx(object, outfile, transpose = TRUE)
```

```
## S4 method for signature 'DatasetExperiment'
export_xlsx(object, outfile, transpose = TRUE)
```

Arguments

object	a dataset object
outfile	the filename (including path) to write the data to
transpose	TRUE (default) or FALSE to transpose the output data

Value

an excel file with sheets for data and meta data

Examples

```
## Not run:
D = iris_DatasetExperiment() # example dataset
export_xlsx(D, 'iris_DatasetExperiment.xlsx')

## End(Not run)
```

iris_DatasetExperiment	<i>Fisher's Iris data</i>
------------------------	---------------------------

Description

Fisher's Iris data as a DatasetExperiment object

Usage

```
iris_DatasetExperiment()
```

Value

DatasetExperiment object

Examples

```
D = iris_DatasetExperiment()
```

is_output	<i>Verify output</i>
-----------	----------------------

Description

Verify that the name of a output is valid for an object

Usage

```
is_output(obj, name)

## S4 method for signature 'struct_class'
is_output(obj, name)
```

Arguments

obj	A model or iterator object derived from the <i>*struct*</i> class
name	Name of output

Value

TRUE if output name is valid, FALSE if not

Methods (by class)

- struct_class:

Examples

```
M = example_model()
is_output(M, 'result_1') # TRUE
is_output(M, 'result_0') # FALSE
```

is_param	<i>Verify parameter</i>
----------	-------------------------

Description

Verify that the input name is a valid input parameter for an object

Usage

```
is_param(obj, name)

## S4 method for signature 'struct_class'
is_param(obj, name)
```

Arguments

obj	An object derived from struct_class
name	Name of parameter

Value

TRUE if parameter name is valid, FALSE if not

Methods (by class)

- struct_class:

Examples

```
M = example_model()
is_param(M, 'value_1') # TRUE
is_param(M, 'alpha')  # FALSE
```

libraries

Libraries for an object

Description

All struct objects have a "libraries" slot, which is a character array of libraries required to use the object. The `libraries` method gathers libraries from an object and all struct objects that it inherits to generate a complete list.

Usage

```
libraries(obj)

## S4 method for signature 'struct_class'
libraries(obj)
```

Arguments

`obj` a struct object

Value

a character array of R packages needed by the object

Examples

```
M = example_model()
libraries(M)
```

max_length	<i>get the max value vector length for an entity</i>
------------	--

Description

A base class in the **struct** package. Not normally called directly. An entity object is used to store information about a parameter or output_ The standard 'name','description' and 'type' slots are included, along with 'value' for storing the value of the parameter and 'max_length' for restricting the length of 'value' if needed.

Usage

```
max_length(obj)

entity(
  name,
  description = character(0),
  type = "character",
  value = NULL,
  max_length = Inf
)

## S4 method for signature 'entity'
value(obj)

## S4 replacement method for signature 'entity'
value(obj) <- value

## S4 method for signature 'entity'
max_length(obj)

## S4 replacement method for signature 'entity'
max_length(obj) <- value
```

Arguments

obj	An entity object
name	the name of the object
description	a description of the object
type	the type of the struct object
value	The value of the parameter/outputs
max_length	Maximum length of value vector (default 1)

Details

Entity objects are usually defined in the prototype of another object, but can be extracted using param_obj and output_obj.

Value

max value vector length for an entity

An entity object

Examples

```
# Create a new entity object
E = entity(
  name = 'example',
  description = 'this is an example',
  type = 'numeric',
  value = 1
)

# Get/set the value of the entity object
value(E)
value(E) = 10
```

 model

model class

Description

A class for models that can be trained/applied to datasets e.g. PCA, PLS etc. Also used for preprocessing steps that require application to test sets. not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

Usage

```
model(
  predicted = character(0),
  seq_in = "data",
  seq_fcn = function(x) { return(x) },
  ...
)

## S4 method for signature 'model,DataSetExperiment'
model_train(M, D)

## S4 method for signature 'model,DataSetExperiment'
model_predict(M, D)

## S4 method for signature 'model,DataSetExperiment'
model_apply(M, D)

## S4 method for signature 'model,DataSetExperiment'
model_reverse(M, D)

## S4 method for signature 'model'
predicted(M)
```

```

## S4 method for signature 'model'
seq_in(M)

## S4 replacement method for signature 'model,character'
seq_in(M) <- value

## S4 method for signature 'model'
predicted_name(M)

## S4 replacement method for signature 'model,character'
predicted_name(M) <- value

```

Arguments

predicted	The name of an output slot to return when using predicted() (see details)
seq_in	the name of an output slot to connect with the "predicted" output of another model (see details)
seq_fcn	a function to apply to seq_in before inputting into the next model. Typically used to extract a single column, or convert from factor to char etc.
...	named slots and their values.
M	A struct model object
D	A DatasetExperiment object
value	The value to assign

Value

trained model object
 model object with test set results
 trained model object
 dataset dataset object with the reverse model applied
 the predicted output, as specified by predicted_name
 the id of the input parameter to be replaced by the predicted output of the previous model in a model sequence. Reserved keyword 'data' means that the input data used by model_train, model_apply etc is used. seq_in = 'data' is the default setting.
 the modified model object
 the id of the output returned by predicted()
 the modified model object

predicted slot

The "predicted" slot is a slots for use by users to control the flow of model sequences. The predicted() function is used to return a default output and from a model. Typically it is a DatasetExperiment object that is passed directly into the next model in a sequence as the data for that model.

seq_in slot

In a sequence of models (see `model_seq`) the "predicted" slot is connected to the `DatasetExperiment` input of the next model. `seq_in` can be used to control flow and connect the "predicted" output to the input parameter of the next model. Default is the keyword 'data', and can otherwise be replaced by any input slot from the model. The slot `seq_fcn` can be used to apply a transformation to the output before it is used as an input. This allows you to e.g. convert between types, extract a single column from a `data.frame` etc.

Examples

```
M = model()
D = DatasetExperiment()
M = model()
M = model_train(M,D)
D = DatasetExperiment()
M = model()
M = model_train(M,D)
M = model_predict(M,D)
D = DatasetExperiment()
M = model()
M = model_apply(M,D)
D = DatasetExperiment()
M = model()
M = model_train(M,D)
M = model_predict(M,D)
M = model_reverse(M,D)
D = DatasetExperiment()
M = example_model()
M = model_train(M,D)
M = model_predict(M,D)
p = predicted(M)
D = DatasetExperiment()
M = example_model()
seq_in(M) = 'data'
M = example_model()
seq_in(M) = 'value_1'
M = example_model()
predicted_name(M)
M = example_model()
predicted_name(M) = 'result_2'
```

models

Get/set models of a model_seq

Description

Returns the list of models in a `model_seq` object

Usage

```
models(ML)
```

```
models(ML) <- value
```

Arguments

ML a model_seq object
value a list containing only model objects

Value

models(ML) returns a list of models in the model sequence
models(ML)<- sets the list of models in the model sequence

Examples

```
# Create a model sequence  
ML = model_seq()  
models(ML) = list(example_model(), example_model())  
models(ML)
```

model_apply *Apply a model*

Description

Applies a method to the input dataset

Usage

```
model_apply(M, D)
```

Arguments

M a 'method' object
D another object used by the first

Value

Returns a modified method object

Examples

```
M = example_model()  
M = model_apply(M, iris_DatasetExperiment())
```

model_predict	<i>Model prediction</i>
---------------	-------------------------

Description

Apply a model using the input dataset_ Assumes the model is trained first.

Usage

```
model_predict(M, D)
```

Arguments

M	a model object
D	a dataset object

Value

Returns a modified model object

Examples

```
M = example_model()
M = model_predict(M, iris_DatasetExperiment())
```

model_reverse	<i>Reverse preprocessing</i>
---------------	------------------------------

Description

Reverse the effect of a preprocessing step on a dataset_

Usage

```
model_reverse(M, D)
```

Arguments

M	a model object
D	a dataset object

Value

Returns a modified dataset object

Examples

```
M = example_model()
D = model_reverse(M, iris_DatasetExperiment())
```

 model_seq

model_seq class

Description

A class for (ordered) lists of models

Usage

```

model_seq(...)

## S4 method for signature 'model_seq,DataSetExperiment'
model_train(M, D)

## S4 method for signature 'model_seq,DataSetExperiment'
model_predict(M, D)

## S4 method for signature 'model_seq,ANY,ANY,ANY'
x[i]

## S4 replacement method for signature 'model_seq,ANY,ANY,ANY'
x[i] <- value

## S4 method for signature 'model_seq'
models(ML)

## S4 replacement method for signature 'model_seq,list'
models(ML) <- value

## S4 method for signature 'model_seq'
length(x)

## S4 method for signature 'model,model_seq'
e1 + e2

## S4 method for signature 'model_seq,model'
e1 + e2

## S4 method for signature 'model,model'
e1 + e2

## S4 method for signature 'model_seq'
predicted(M)

## S4 method for signature 'model_seq,DataSetExperiment'
model_apply(M, D)

```

Arguments

...	named slots and their values.
M	a model object

D	a dataset object
x	a model_seq object
i	index
value	value
ML	a model_seq object
e1	a model or model_seq object
e2	a model or model_seq object

Value

model sequence
 model sequence
 model at the given index in the sequence
 model sequence with the model at index i replaced
 a list of models in the sequence
 a model sequence containing the input models
 the number of models in the sequence
 a model sequence with the additional model appended to the front of the sequence
 a model sequence with the additional model appended to the end of the sequence
 a model sequence
 the predicted output of the last model in the sequence

Examples

```

MS = model_seq()
MS = model() + model()
MS = example_model() + example_model()
MS = model_train(MS, DatasetExperiment())
D = DatasetExperiment()
MS = example_model() + example_model()
MS = model_train(MS, D)
MS = model_predict(MS, D)
MS = model() + model()
MS[2]

MS = model() + model()
MS[3] = model()

MS = model() + model()
L = models(MS)

MS = model_seq()
L = list(model(), model())
models(MS) = L

MS = model() + model()
length(MS) # 2

MS = model() + model()
M = model()

```

```
MS = M + MS

MS = model() + model()
M = model()
MS = MS + M

MS = model() + model()

D = DatasetExperiment()
M = example_model()
M = model_train(M,D)
M = model_predict(M,D)
p = predicted(M)
D = DatasetExperiment()
MS = example_model() + example_model()
MS = model_apply(MS,D)
```

model_train	<i>Train a model</i>
-------------	----------------------

Description

Trains a model using the input dataset

Usage

```
model_train(M, D)
```

Arguments

M	a model object
D	a dataset object

Value

Returns a modified model object

Examples

```
M = example_model()
M = model_train(M,iris_DatasetExperiment())
```

new_struct	<i>Generate a struct object from a Class</i>
------------	---

Description

This function creates a newly allocated object from the class identified by the first argument. It works almost identically to `new` but is specific to objects from the **struct** package and ensures that entity slots have their values assigned correctly. This function is usually called by class constructors and not used directly.

Usage

```
new_struct(class, ...)
```

Arguments

<code>class</code>	The class of struct object to create
<code>...</code>	named slots and values to assign

Value

An object derived from `struct_class`

Examples

```
S = new_struct('struct_class')
```

optimiser	<i>optimiser class</i>
-----------	------------------------

Description

A special class of iterator for selecting optimal parameter values not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

Usage

```
optimiser(...)
```

Arguments

<code>...</code>	named slots and their values.
------------------	-------------------------------

Value

an optimiser object

Examples

```
OPT = optimiser()
```

output_ids	<i>Output identifiers</i>
------------	---------------------------

Description

return a list of valid output ids for an object

Usage

```
output_ids(obj)

## S4 method for signature 'struct_class'
output_ids(obj)
```

Arguments

obj A model or iterator object derived from the **struct** class

Value

list of output ids

Methods (by class)

- struct_class:

Examples

```
M = example_model()
output_ids(M)
```

output_list	<i>output list</i>
-------------	--------------------

Description

get/set a named list of outputs and their current value for an object

Usage

```
output_list(obj)

output_list(obj) <- value

## S4 method for signature 'struct_class'
output_list(obj)

## S4 replacement method for signature 'struct_class,list'
output_list(obj) <- value
```

Arguments

obj An object derived from struct_class
 value A named list of outputs and corresponding values

Value

A named list of outputs and corresponding values
 struct object

Methods (by class)

- struct_class:
- obj = struct_class, value = list:

Examples

```
M = example_model()
L = output_list(M)
M = example_model()
output_list(M) = list('result_1' = DatasetExperiment(), 'result_2' = DatasetExperiment())
```

output_name	<i>output name</i>
-------------	--------------------

Description

return a the name for a output, if available

Usage

```
output_name(obj, name)
```

```
## S4 method for signature 'struct_class,character'
output_name(obj, name)
```

Arguments

obj A model or iterator object derived from the *struct* class
 name Name of output

Value

name of output

Methods (by class)

- obj = struct_class, name = character:

Examples

```
M = example_model()
output_name(M, 'result_1')
```

output_obj	<i>Output objects</i>
------------	-----------------------

Description

Gets or sets the object of an output e.g. to an entity() object.

Usage

```
output_obj(obj, name)

output_obj(obj, name) <- value

## S4 method for signature 'struct_class,character'
output_obj(obj, name)

## S4 replacement method for signature 'struct_class,character'
output_obj(obj, name) <- value
```

Arguments

obj	A model or iterator object derived from the *struct* class
name	Name of output
value	A valid value for the output being set

Value

output_obj(M, name) returns the named output as an object
 output_obj(M, name)<- sets the named output of an object
 the modified object

Methods (by class)

- obj = struct_class, name = character:
- obj = struct_class, name = character:

Examples

```
# get the output as an object
M = example_model()
obj = output_obj(M, 'result_1')

# set a output as an object
output_obj(M, 'result_1') = entity(value = 15, type = 'numeric', name = 'result_1')
```

output_value	<i>output values</i>
--------------	----------------------

Description

get/set the values for an output_

Usage

```
output_value(obj, name)
```

```
output_value(obj, name) <- value
```

```
## S4 method for signature 'struct_class,character'  
output_value(obj, name)
```

```
## S4 replacement method for signature 'struct_class,character'  
output_value(obj, name) <- value
```

Arguments

obj	A model or iterator object derived from the <i>*struct*</i> class
name	Name of output
value	A valid value for the output being set

Value

Value of output
struct object

Methods (by class)

- obj = struct_class, name = character:
- obj = struct_class, name = character:

Examples

```
M = example_model()  
output_value(M, 'result_1')  
M = example_model()  
output_value(M, 'result_1') = DatasetExperiment()
```

param_ids	<i>Parameter identifiers</i>
-----------	------------------------------

Description

return a list of valid parameter ids for an object

Usage

```
param_ids(obj)

## S4 method for signature 'struct_class'
param_ids(obj)
```

Arguments

obj An object derived from struct_class

Value

list of parameter ids

Methods (by class)

- struct_class:

Examples

```
M = example_model()
param_ids(M)
```

param_list	<i>Parameter list</i>
------------	-----------------------

Description

get/set a named list of parameters and their current value for an object

Usage

```
param_list(obj)

param_list(obj) <- value

## S4 method for signature 'struct_class'
param_list(obj)

## S4 replacement method for signature 'struct_class,list'
param_list(obj) <- value
```

Arguments

obj	An object derived from struct_class
value	A named list of parameters and corresponding values

Value

A named list of parameters names and corresponding values

Methods (by class)

- struct_class:
- obj = struct_class, value = list:

Examples

```
M = example_model()
L = param_list(M)
```

```
M = example_model()
param_list(M) = list('value_1' = 15, 'value_2' = 20)
```

param_name

Parameter name

Description

Returns the name for a parameter, if available

Usage

```
param_name(obj, name)
```

```
## S4 method for signature 'struct_class,character'
param_name(obj, name)
```

Arguments

obj	An object derived from struct_class
name	Name of parameter

Value

name of parameter

Methods (by class)

- obj = struct_class, name = character:

Examples

```
M = example_model()
param_name(M, 'value_1')
```

param_obj	<i>Parameter objects</i>
-----------	--------------------------

Description

Gets or sets the object of a parameter e.g. to an entity() object.

Usage

```
param_obj(obj, name)

param_obj(obj, name) <- value

## S4 replacement method for signature 'struct_class,character'
param_obj(obj, name) <- value

## S4 method for signature 'struct_class,character'
param_obj(obj, name)
```

Arguments

obj	An object derived from struct_class
name	Name of parameter
value	A valid value for the parameter being set

Value

param_obj(M, name) Returns the named parameter as an object
param_obj(M, name) <- Sets the named parameter of an object

Methods (by class)

- obj = struct_class, name = character:
- obj = struct_class, name = character:

Examples

```
# get the parameter as an object
M = example_model()
obj = param_obj(M, 'value_0')

# set a parameter as an object
param_obj(M, 'value_0') = entity(value = 15, type = 'numeric', name='value_0')
```

param_value	<i>Parameter values</i>
-------------	-------------------------

Description

get/set the values for a parameter.

Usage

```
param_value(obj, name)
```

```
param_value(obj, name) <- value
```

```
## S4 method for signature 'struct_class,character'  
param_value(obj, name)
```

```
## S4 replacement method for signature 'struct_class,character'  
param_value(obj, name) <- value
```

Arguments

obj	A model or iterator object derived from structclass
name	Name of parameter
value	A valid value for the parameter being set

Value

Value of parameter

Methods (by class)

- obj = struct_class, name = character:
- obj = struct_class, name = character:

Examples

```
M = example_model()  
param_value(M, 'value_1')
```

```
M = example_model()  
param_value(M, 'value_1') = 0.95
```

predicted	<i>Prediction output</i>
-----------	--------------------------

Description

returns the prediction output for a model_ This is supplied as input to the next model when used in a model_seq

Usage

```
predicted(M)
```

Arguments

M a model object

Value

The value returned varies depending on the output_

Examples

```
M = example_model()
M = model_train(M, iris_DatasetExperiment())
M = model_predict(M, iris_DatasetExperiment())
predicted(M)
```

predicted_name	<i>Predicted output name</i>
----------------	------------------------------

Description

get/set the prediction output for a model_ This determines which outputs from this model are supplied as inputs to the next model when used in a model_seq

Usage

```
predicted_name(M)

predicted_name(M) <- value
```

Arguments

M a model object
value name of an output for this model

Value

predicted_name returns the name of the predicted output
predicted_name<- sets the name of the predicted output

Examples

```
M = example_model()
predicted_name(M)
predicted_name(M) = 'result_2'
```

```
preprocess          preprocessing class
```

Description

A class used for preprocessing steps that require application to test sets. not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

Usage

```
preprocess(...)
```

S4 method for signature 'preprocess,DatasetExperiment'

```
model_reverse(M, D)
```

Arguments

```
...          named slots and their values.
M           a model object
D           a dataset object
```

Value

dataset object

Examples

```
M = preprocess()
D = DatasetExperiment()
M = model()
D2 = model_reverse(M,D)
```

```
resampler          resampler class
```

Description

A class for resampling methods such as cross-validation. not intended to be called directly.

Usage

```
resampler(...)
```

Arguments

... named slots and their values.

Value

a resampler object

Examples

```
R = resampler()
```

result	<i>Iterator result</i>
--------	------------------------

Description

Returns the results of an iterator. This is used to control model flow in a similar way to predict for model and model_seq objects.

Usage

```
result(M)
```

Arguments

M an iterator object

Value

the returned output varies with the algorithm implemented

Examples

```
D = iris_DatasetExperiment() # get some data
MET = metric() # use a metric
I = example_iterator() # initialise iterator
models(I) = example_model() # set the model
I = run(I,D,MET) # run
result(I)
```

result_name	<i>get/set output name as prediction output for a model</i>
-------------	---

Description

get/set the prediction output for a model_ This determines which outputs from this model are supplied as inputs to the next model when used in a model_seq

Usage

```
result_name(M)

result_name(I) <- value
```

Arguments

M	an iterator object
I	an iterator object
value	name of an output for iterator M

Value

result_name(M) returns the name of the output for this iterator (equivalent to predicted for model objects)

result_name(I)<- sets the default output for an iterator

Examples

```
I = example_iterator() # initialise iterator
result_name(I)
result_name(I) = 'result_1'
```

run	<i>Run iterator</i>
-----	---------------------

Description

Runs an iterator, applying the chosen model multiple times.

Evaluates an iterator by e.g. averaging over all iterations. May be deprecated in a future release as evaluate is applied by run anyway.

A class for iterative approaches that involve the training/prediction of a model multiple times. Not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

Usage

```

run(I, D, MET)

evaluate(I, MET)

iterator(...)

## S4 method for signature 'iterator,DatasetExperiment,metric'
run(I, D, MET = NULL)

## S4 method for signature 'iterator,metric'
evaluate(I, MET)

## S4 method for signature 'iterator'
models(ML)

## S4 replacement method for signature 'iterator,model_OR_iterator'
models(ML) <- value

## S4 replacement method for signature 'iterator,character'
result_name(I) <- value

## S4 method for signature 'iterator'
result(M)

## S4 method for signature 'iterator'
result_name(M)

## S4 method for signature 'iterator,model_OR_iterator'
e1 * e2

## S4 method for signature 'iterator,ANY,ANY,ANY'
x[i]

## S4 replacement method for signature 'iterator,ANY,ANY,ANY'
x[i] <- value

```

Arguments

I	an iterator object
D	a dataset object
MET	a metric object
...	named slots and their values.
ML	a model sequence object
value	value
M	a model object
e1	an iterator object
e2	an iterator or a model object
x	a sequence object
i	index into sequence

Details

Running an iterator will apply the iterator a number of times to a dataset_ For example, in cross-validation the same model is applied multiple times to the same data, splitting it into training and test sets. The input metric object can be calculated and collected for each iteration as an output_

Value

Modified iterator object
 Modified iterator object
 the modified model object
 model at the given index in the sequence
 iterator with the model at index i replaced

Examples

```
D = iris_DatasetExperiment() # get some data
MET = metric() # use a metric
I = example_iterator() # initialise iterator
models(I) = example_model() # set the model
I = run(I,D,MET) # run
D = iris_DatasetExperiment() # get some data
MET = metric() # use a metric
I = example_iterator() # initialise iterator
models(I) = example_model() # set the model
I = run(I,D,MET) # run
I = evaluate(I,MET) # evaluate
I = iterator()
I = iterator() * model()
D = DatasetExperiment()
MET = metric()
I = iterator() * model()
I = run(I,D,MET)

I = iterator()
result_name(I) = 'example'
MS = model() + model()
I = iterator() * MS
I[2] # returns the second model() object

MS = model() + model()
I = iterator() * MS
I[2] = model() # sets the second model to model()
```

seq_in

*Sequence input***Description**

get/set the input parameter replaced by the output of the previous model in a model sequence. Default is "data" which passes the output as the data input for methods such as model_train and model_apply.

Usage

```
seq_in(M)

seq_in(M) <- value
```

Arguments

M	a model object
value	name of an output for this model

Value

seq_in returns the name of the input parameter replaced when used in a model sequence
 seq_in<- sets the name of the input parameter replaced when used in a model sequence

Examples

```
M = example_model()
seq_in(M)
seq_in(M) = 'value_1'
```

set_obj_method	<i>update method for a struct object</i>
----------------	--

Description

a helper function to update methods for a struct object

Usage

```
set_obj_method(
  class_name,
  method_name,
  definition,
  where = toplev(parent.frame()),
  signature = c(class_name, "DatasetExperiment")
)
```

Arguments

class_name	the name of the to update the method for
method_name	the name of the method to update. Must be an existing method for the object.
definition	the function to replace the method with. This function will be used when the method is called on the object.
where	the environment to create the object in. default where = toplev(parent.frame())
signature	a list of classes that this object requires as inputs. Default is c(class_name,'DatasetExperiment')

Value

a method is created in the specified environment

Examples

```

set_struct_obj(
  class_name = 'add_two_inputs',
  struct_obj = 'model',
  stato = FALSE,
  params = c(input_1 = 'numeric', input_2 = 'numeric'),
  outputs = c(result = 'numeric'),
  prototype = list(
    input_1 = 0,
    input_2 = 0,
    name = 'Add two inputs',
    description = 'example class that adds two values together')
)

```

set_obj_show

a helper function to update the show method for a struct object

Description

a helper function to update the show method for a struct object

Usage

```
set_obj_show(class_name, extra_string, where = toparent(parent.frame()))
```

Arguments

class_name	the name of the to update the method for
extra_string	a function that returns an extra string using the input object as an input e.g. function(object)return = 'extra_string'
where	the environment to create the object in. default where = toparent(parent.frame())

Value

a method is created in the specified environment

Examples

```

# create an example object first
set_struct_obj(
  class_name = 'add_two_inputs',
  struct_obj = 'model',
  stato = FALSE,
  params = c(input_1 = 'numeric', input_2 = 'numeric'),
  outputs = c(result = 'numeric'),
  prototype = list(
    input_1 = 0,
    input_2 = 0,
    name = 'Add two inputs',
    description = 'example class that adds two values together')
)

# now update the method

```

```

set_obj_show(
  class_name = 'add_two_inputs',
  extra_string = function(object) {return('The extra text')}}
)

```

set_struct_obj *define a new struct object*

Description

a helper function to create new struct objects

Usage

```

set_struct_obj(
  class_name,
  struct_obj,
  stato = TRUE,
  params = character(0),
  outputs = character(0),
  private = character(0),
  prototype = list()
)

```

Arguments

class_name	the name of the new class to create
struct_obj	the struct obj to inherit e.g. 'model', 'metric' etc
stato	TRUE (default) or FALSE to inherit the stato class
params	a named character vector of input parameters where each element specifies the type of value that will be in the slot e.g. c(example = 'character')
outputs	a named character vector of outputs where each element specifies the type of value that will be in the slot e.g. c(example = 'character')
private	a named character vector of private slots where each element specifies the type of value that will be in the slot e.g. c(example = 'character'). These are intended for internal use by the object and generally not available to the user.
prototype	a named list with initial values for slots.

Value

a new class definition. to create a new object from this class use `X = new_class_name()`

stato_id	<i>get the stato_id for an object</i>
----------	---------------------------------------

Description

A base class in the **struct** package. Provides several fundamental methods and should not be called directly.

Usage

```
stato_id(obj)

stato_name(obj)

stato_definition(obj)

stato_summary(obj)

stato(stato_id)

## S4 method for signature 'stato'
stato_id(obj)

## S4 method for signature 'stato'
stato_name(obj)

## S4 method for signature 'stato'
stato_definition(obj)

## S4 method for signature 'stato'
stato_summary(obj)
```

Arguments

obj	An object derived from the stato object
stato_id	A STATO ID e.g. OBI:0000001

Details

STATO is the statistical methods ontology. It contains concepts and properties related to statistical methods, probability distributions and other concepts related to statistical analysis, including relationships to study designs and plots (see <http://stato-ontology.org/>).

This class provides access to a version of the STATO ontology database that can be searched by ontology id to provide formal names and definitions for methods, models, iterators, metrics and charts.

This class makes use of the ontologyIndex package to search a copy of the STATO database included in this package.

Value

id the stato id
 name the stato name
 def the stato description
 Value returned depends on the method used.

Examples

```
M = example_model()
stato_id(M)
stato_name(M)
stato_definition(M)
stato_summary(M)
# an example stato object
M = example_model()

# the stato id assigned to object M
stato_id(M) # OBI:0000011

# the name associated with that id
stato_name(M)

# the STATO definition for that id
stato_definition(M)

# a summary of the STATO database entry for the id, and any parameters or
# outputs that also have stato ids.
stato_summary(M)
```

 struct

StRUCT: Statistics in R Using Class Templates

Description

This package defines classes (templates) for developing statistical workflows. These classes can be extended using other packages, making it easier to combine methods from different packages into a robust workflow. Integration with STATO: the statistical methods ontology (<https://www.ebi.ac.uk/ols/ontologies/stato>) provides standardised definitions for many statistical methods.

Classes

The classes include:

- **DatasetExperiment**: An extension of the SummarizedExperiment object by Bioconductor
- **model**: A template for training and applying statistics
- **iterator**: A template for resampling, optimisation and validation of statistical models
- **chart**: A template for generating graphical outputs for models and iterators

struct_class	<i>Constructor for struct_class objects</i>
--------------	---

Description

Creates a new `struct_class` object and populates the slots. Not intended for direct use.

Usage

```
struct_class(
  name = character(),
  description = character(),
  type = character(),
  citations = list()
)
```

Arguments

name	the name of the object
description	a description of the object
type	the type of the struct object
citations	a list of citations for the object in "bibentry" format

Value

a `struct_class` object

struct_class-class	<i>struct_class object definition</i>
--------------------	---------------------------------------

Description

Defines the struct class base template. This class is inherited by other objects and not intended for direct use. It defines slots and methods common to all **struct** objects.

Value

Returns a **struct** object

Public slots

Public slots can be accessed using shorthand \$ notation and are intended for users building workflows.

name	character()	A short descriptive name of the struct object
description	character()	A longer description of the struct object and what it does
type	character()	A keyword that describes the type of struct object
libraries	character()	A (read only) list of R packages used by this struct object
citations	list of bibentry	A (read only) list of citations relevant to this struct object, in Bibtex format.

Private slots

Private slots are not readily accessible to users and are intended for developers creating their own struct objects. Any slot not listed within `‘.params‘` or `‘.outputs‘` is considered a private slot.

`.params` character() A list of additional slot names that can be get/set by the user for a specific struct object. These are used as input parameters for different methods.

`.outputs` character() a list of additional slot names that can be get by the user. These are used to store the results of a method.

Examples

```
S = struct_class(name = 'Example',description = 'An example object')
```

struct_template	<i>StRUCT templates</i>
-----------------	-------------------------

Description

Create a struct template

Usage

```
struct_template(  
  template = "model",  
  output,  
  in_editor = TRUE,  
  overwrite = FALSE  
)
```

Arguments

template	the type of object you want a template for e.g. 'model'
output	the name/path of the output file
in_editor	TRUE/FALSE to open the created file in the default editor
overwrite	= TRUE/FALSE to overwrite file if exists already

Value

A template is created at the output location specified

Examples

```
## Not run:  
struct_template('model', 'example.R', FALSE)  
  
## End(Not run)
```

test_metric-class	<i>Example metric</i>
-------------------	-----------------------

Description

An example metric for testing
calculates a metric, which just returns a value of 3.142

Usage

```
## S4 method for signature 'test_metric'
calculate(obj)
```

Arguments

obj	metric object
-----	---------------

Value

test metric object
dataset object

Examples

```
MET = test_metric()

MET = test_metric()
MET = calculate(MET)
```

`$, struct_class-method` *Get/set parameter or output values*

Description

Dollar syntax can be used to as a shortcut for getting/setting input parameter and output values for struct objects.

Usage

```
## S4 method for signature 'struct_class'
x$name
```

Arguments

x	An object derived from struct_class
name	The name of the slot to access

Value

Parameter/output value

Examples

```
M = example_model()
M$value_1 = 10
M$value_1 # 10
```

`$<- ,struct_class-method`

Get/set parameter or output values

Description

Dollar syntax can be used to as a shortcut for getting/setting input parameter and output values for struct objects.

Usage

```
## S4 replacement method for signature 'struct_class'
x$name <- value
```

Arguments

<code>x</code>	An object derived from <code>struct_class</code>
<code>name</code>	The name of the slot to access
<code>value</code>	The value to assign

Value

Parameter/output value

Examples

```
M = example_model()
M$value_1 = 10
M$value_1 # 10
```

Index

- `*`, iterator, model_OR_iterator-method (run), 41
- `+`, model, model-method (model_seq), 26
- `+`, model, model_seq-method (model_seq), 26
- `+`, model_seq, model-method (model_seq), 26
- `.struct_class` (struct_class-class), 49
- `[`, iterator, ANY, ANY, ANY-method (run), 41
- `[`, model_seq, ANY, ANY, ANY-method (model_seq), 26
- `[<-`, iterator, ANY, ANY, ANY-method (run), 41
- `[<-`, model_seq, ANY, ANY, ANY-method (model_seq), 26
- `$`, DatasetExperiment-method (DatasetExperiment), 10
- `$`, struct_class-method, 51
- `$<-`, struct_class-method, 52
- `$<-`, DatasetExperiment-method (DatasetExperiment), 10

- `as.code`, 3
- `as.code`, iterator-method (as.code), 3
- `as.code`, model-method (as.code), 3
- `as.code`, model_seq-method (as.code), 3
- `as.DatasetExperiment`, 4
- `as.DatasetExperiment`, SummarizedExperiment-method, 4
- `as.SummarizedExperiment`, 5
- `as.SummarizedExperiment`, DatasetExperiment-method, 5
- `as_data_frame`, 6

- `calculate`, 6
- `calculate`, metric-method (calculate), 6
- `calculate`, test_metric-method (test_metric-class), 51
- `chart`, 7
- `chart_names`, 8
- `chart_names`, struct_class-method (chart_names), 8
- `chart_plot`, 9
- `chart_plot`, chart, ANY-method (chart_plot), 9

- `chart_plot`, example_chart, example_model-method (example_chart), 14
- `citations`, 10
- `citations`, struct_class-method (citations), 10

- DatasetExperiment, 10

- `entity`, 12
- `entity` (max_length), 20
- `entity_stato`, 11
- `enum`, 12, 14
- `enum_stato`, 13
- `evaluate` (run), 41
- `evaluate`, iterator, metric-method (run), 41
- `example_chart`, 14
- `example_iterator` (example_iterator-class), 15
- `example_iterator-class`, 15
- `example_model`, 16
- `export_xlsx`, 17
- `export_xlsx`, DatasetExperiment-method (export_xlsx), 17

- `iris_DatasetExperiment`, 17
- `is_output`, 18
- `is_output`, struct_class-method (is_output), 18
- `is_param`, 18
- `is_param`, struct_class-method (is_param), 18
- `iterator` (run), 41

- `length`, model_seq-method (model_seq), 26
- `libraries`, 19
- `libraries`, struct_class-method (libraries), 19

- `max_length`, 20
- `max_length`, entity-method (max_length), 20
- `max_length<-` (calculate), 6
- `max_length<-`, entity-method (max_length), 20

- metric (calculate), 6
- model, 21
- model_apply, 24
- model_apply, model, DatasetExperiment-method (model), 21
- model_apply, model_seq, DatasetExperiment-method (model_seq), 26
- model_predict, 25
- model_predict, example_model, DatasetExperiment-method (example_model), 16
- model_predict, model, DatasetExperiment-method (model), 21
- model_predict, model_seq, DatasetExperiment-method (model_seq), 26
- model_reverse, 25
- model_reverse, model, DatasetExperiment-method (model), 21
- model_reverse, preprocess, DatasetExperiment-method (preprocess), 39
- model_seq, 26
- model_train, 28
- model_train, example_model, DatasetExperiment-method (example_model), 16
- model_train, model, DatasetExperiment-method (model), 21
- model_train, model_seq, DatasetExperiment-method (model_seq), 26
- models, 23
- models, iterator-method (run), 41
- models, model_seq-method (model_seq), 26
- models<- (models), 23
- models<-, iterator, model_OR_iterator-method (run), 41
- models<-, model_seq, list-method (model_seq), 26
- new_struct, 29
- optimiser, 29
- output_ids, 30
- output_ids, struct_class-method (output_ids), 30
- output_list, 30
- output_list, struct_class-method (output_list), 30
- output_list<- (output_list), 30
- output_list<-, struct_class, list-method (output_list), 30
- output_name, 31
- output_name, struct_class, character-method (output_name), 31
- output_obj, 32
- output_obj, struct_class, character-method (output_obj), 32
- output_obj<- (output_obj), 32
- output_obj<-, struct_class, character-method (output_obj), 32
- output_value, 33
- output_value, struct_class, character-method (output_value), 33
- output_value<- (output_value), 33
- output_value<-, struct_class, character-method (output_value), 33
- param_ids, 34
- param_ids, struct_class-method (param_ids), 34
- param_list, 34
- param_list, struct_class-method (param_list), 34
- param_list<- (param_list), 34
- param_list<-, struct_class, list-method (param_list), 34
- param_name, 35
- param_name, struct_class, character-method (param_name), 35
- param_obj, 36
- param_obj, struct_class, character-method (param_obj), 36
- param_obj<- (param_obj), 36
- param_obj<-, struct_class, character-method (param_obj), 36
- param_value, 37
- param_value, struct_class, character-method (param_value), 37
- param_value<- (param_value), 37
- param_value<-, struct_class, character-method (param_value), 37
- predicted, 38
- predicted, model-method (model), 21
- predicted, model_seq-method (model_seq), 26
- predicted_name, 38
- predicted_name, model-method (model), 21
- predicted_name<- (predicted_name), 38
- predicted_name<-, model, character-method (model), 21
- preprocess, 39
- resampler, 39
- result, 40
- result, iterator-method (run), 41
- result_name, 41
- result_name, iterator-method (run), 41
- result_name<- (result_name), 41

result_name<- , iterator, character-method
(run), 41

run, 41

run, example_iterator, DatasetExperiment, metric-method
(example_iterator-class), 15

run, iterator, DatasetExperiment, metric-method
(run), 41

seq_in, 43

seq_in, model-method (model), 21

seq_in<- (seq_in), 43

seq_in<- , model, character-method
(model), 21

set_obj_method, 44

set_obj_show, 45

set_struct_obj, 46

stato, 12, 14

stato (stato_id), 47

stato_definition (stato_id), 47

stato_definition, stato-method
(stato_id), 47

stato_id, 47

stato_id, stato-method (stato_id), 47

stato_name (stato_id), 47

stato_name, stato-method (stato_id), 47

stato_summary (stato_id), 47

stato_summary, stato-method (stato_id),
47

struct, 48

struct_class, 49, 49

struct_class-class, 49

struct_template, 50

test_metric (test_metric-class), 51

test_metric-class, 51

value (calculate), 6

value, entity-method (max_length), 20

value, metric-method (calculate), 6

value<- (calculate), 6

value<- , entity-method (max_length), 20

value<- , enum-method (enum), 12

value<- , metric-method (calculate), 6