Package: fuseMLR (via r-universe)

December 18, 2024

Type Package

Title Fusing Machine Learning in R

Version 0.0.1

Maintainer Cesaire J. K. Fouodo <cesaire.kuetefouodo@uni-luebeck.de>

Description Recent technological advances have enable the simultaneous collection of multi-omics data i.e., different types or modalities of molecular data, presenting challenges for integrative prediction modeling due to the heterogeneous, high-dimensional nature and possible missing modalities of some individuals. We introduce this package for late integrative prediction modeling, enabling modality-specific variable selection and prediction modeling, followed by the aggregation of the modality-specific predictions to train a final meta-model. This package facilitates conducting late integration predictive modeling in a systematic, structured, and reproducible way.

License GPL-3

Encoding UTF-8

Imports R6, stats, digest

Suggests testthat (>= 3.0.0), UpSetR (>= 1.4.0), caret, ranger, glmnet, Boruta, knitr, rmarkdown, pROC, checkmate

Config/testthat/edition 3

Depends R (>= 3.6.0)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Collate 'Data.R' 'HashTable.R' 'Lrner.R' 'Model.R' 'PredictData.R' 'PredictLayer.R' 'PredictMetaLayer.R' 'Predicting.R' 'Target.R' 'TestData.R' 'TestLayer.R' 'TestMetaLayer.R' 'Testing.R' 'TrainData.R' 'TrainLayer.R' 'TrainMetaLayer.R' 'Training.R' 'VarSel.R' 'bestLayerLearner.R' 'cobra.R' 'createCobraPred.R' 'createDif.R' 'createLoss.R' 'createWeights.R' 'multi_omics.R' 'predict.bestLayerLearner.R' 'predict.cobra.R' 'weightedMeanLearner.R' 'predict.weightedMeanLearner.R' 'testingFunctions.R' 'trainingFunctions.R'

VignetteBuilder knitr, rmarkdown

BugReports `https://github.com/imbs-hl/fuseMLR/issues`

Repository https://imbs-hl.r-universe.dev

RemoteUrl https://github.com/imbs-hl/fusemlr

RemoteRef HEAD

RemoteSha 9869b24595785920ab0fae525e08928bbf2722b7

Contents

`bestLayerLearner` *The best layer-specific model is used as meta model.*

Description

The meta learner is the best layer-specific learner. This function is intended to be (internally) used as meta-learner in fuseMLR.

Usage

```
bestLayerLearner(x, y, perf = NULL)
```

Arguments

`x`	`data.frame` `data.frame` of predictors.
`y`	`vector` True target observations. Either binary or two level factor variable.
`perf`	`function` Function to compute layer-specific performance of learners. If NULL, the Brier Score (classification) or a mean squared error (regression) is used by default as performance measure. Otherwise, the performance function must accept two parameters: `observed` (observed values) and `predicted` (predicted values).

Value

A model object of class `weightedMeanLeaner`.

Examples

```
set.seed(20240624L)
x = data.frame(x1 = runif(n = 50L, min = 0, max = 1))
y = sample(x = 0L:1L, size = 50L, replace = TRUE)
my_best_model = bestLayerLearner(x = x, y = y)
```

cobra *Cobra Meta Learner*

Description

The function cobra implements the COBRA (COmBined Regression Alternative), an aggregation method for combining predictions from multiple individual learners. This method aims to tune key parameters for achieving optimal predictions by averaging the target values of similar candidates in the training dataset's predictions. Only the training points that are sufficiently similar to the test point (based on the proximity threshold epsilon) are used for prediction. If no suitable training points are found, the function returns NA.

Usage

```
cobra(x, y, tune = "epsilon", k_folds = NULL, eps = NULL)
```

Arguments

x	data.frame A training data, where rows are observations and columns are predictions from individual learners. Use NA for missing predictions.
y	vector A vector containing the training targets. This can be a binary or two-level factor variable.
tune	character A character value specifying the tuning mode: • "alpha_epsilon": Tunes both alpha (number of learners) and epsilon (proximity threshold) via cross-validation. • "epsilon": Tunes epsilon only via cross-validation. • "user": No tuning; the user provides an optimal epsilon. #' The default value is epsilon.
k_folds	integer Number of folds for cross-validation when tune = "alpha_epsilon" or "epsilon". Default is 10.
eps	numeric A numeric value for the proximity threshold, used only when tune = "user". Defaults to 0.1.

Value

An object of class cobra containing the training data, target values, and chosen parameters.

References

Biau, G., Fischer, A., Guedj, B., & Malley, J. D. (2014). COBRA: A combined regression strategy. The Journal of Multivariate Analysis 46:18-28

Examples

```
# Example usage
set.seed(123)
x_train <- data.frame(a = runif(10L), b = runif(10L))
y_train <- sample(0L:1L, size = 10L, replace = TRUE)

# Train the model with epsilon optimization
cobra_model <- cobra(x = x_train, y = y_train, tune = "epsilon", k_folds = 2)

# Make predictions on new data
set.seed(156)
x_new <- data.frame(a = runif(5L), b = runif(5L))
prediction <- predict(object = cobra_model, data = x_new)
```

`createCobraPred` *Create COBRA Predictions*

Description

The `createCobraPred` function calculates predictions by averaging the target values of all the nearest candidates in the training dataset. Only the training points that are within the specified proximity (`eps`) to the test point are used to determine the prediction. If no suitable training points are found, the function returns `NA` as the prediction.

Usage

```
createCobraPred(
  train,
  test,
  n_train,
  n_test,
  nlearners,
  eps,
  alpha,
  train_target
)
```

Arguments

`train`	A `matrix` representing the training data. Rows represent observations, and columns contain predictions from individual learners for these observations. In cases where a prediction is unavailable for a specific observation, `NA` is used.
`test`	A `matrix` representing the test data. Rows represent observations, and columns contain predictions from individual learners for these observations. In cases where a prediction is unavailable for a specific observation, `NA` is used.
`n_train`	An `integer` specifying the number of training observations.

`n_test`	An `integer` specifying the number of test observations.
`nlearners`	An `integer` representing the number of learners.
`eps`	A `numeric` value representing the threshold for proximity between two predictions.
`alpha`	A value that determines the optimal number of learners in the neighborhood (only for alpha optimization).
`train_target`	A `vector` containing the target values for the training dataset

`createDif` *Create Difference*

Description

The `createDif` function computes the difference between the maximum and minimum predictions in a dataset.

Usage

```
createDif(x)
```

Arguments

`x`	Predictions vector

`createLoss` *Create Loss*

Description

Create Loss

Usage

```
createLoss(pred, target)
```

Arguments

`pred`	A `vector` of predictions.
`target`	A `vector` of target values.

`createTesting` *createTesting*

Description

Creates a Testing object.

Usage

```
createTesting(id, ind_col, verbose = TRUE)
```

Arguments

`id`	`character` Testing id.
`ind_col`	`character` Name of column of individuals IDs in testing `data.frame`.
`verbose`	`boolean` Warning messages will be displayed if set to `TRUE`.

Value

A Testing object.

`createTestLayer` *createTestLayer*

Description

Creates and stores a TestLayer on the Testing object passed as argument.

Usage

```
createTestLayer(testing, test_layer_id, test_data)
```

Arguments

`testing`	`Testing` Testing object where the created layer will be stored.
`test_layer_id`	`character` ID of the testing layer to be created.
`test_data`	`data.frame` Data modality to be stored in TestData.

Value

The updated Testing object (with the new layer) is returned.

createTraining *createTraining*

Description

Creates a Training object. A training object is designed to encapsulate training layers and training meta-layer. Functions createTrainLayer and createTrainMetaLayer are available to add the training layer and the training meta-layer to a training object.

Usage

```
createTraining(
  id,
  target_df,
  ind_col,
  target,
  problem_type = "classification",
  verbose = TRUE
)
```

Arguments

`id`	`character` Training's ID.
`target_df`	`data.frame` Observed target values. A data frame with two columns: individual IDs and response variable values.
`ind_col`	`character` Name of column of individuals IDs.
`target`	`character` Name of the target variable.
`problem_type`	`character` Either "classification" or "regression".
`verbose`	`boolean` Warning and processing information (including those of cross-validation) will be displayed if set to TRUE.

Value

The created Training object is returned.

See Also

createTrainLayer, createTrainMetaLayer and fusemlr.

`createTrainLayer` *createTrainLayer*

Description

Creates and stores a TrainLayer on the Training object passed as argument. The main components of a training layer are training data modality, a variable selection methods, and a modality-specific learner.

Usage

```
createTrainLayer(
  training,
  train_layer_id,
  train_data,
  varsel_package = NULL,
  varsel_fct = NULL,
  varsel_param = list(),
  lrner_package = NULL,
  lrn_fct,
  param_train_list = list(),
  param_pred_list = list(),
  na_action = "na.rm",
  x_varsel = "x",
  y_varsel = "y",
  x_lrn = "x",
  y_lrn = "y",
  object = "object",
  data = "data",
  extract_pred_fct = NULL,
  extract_var_fct = NULL
)
```

Arguments

`training`	`Training` Training object for storing the created layer.
`train_layer_id`	`character` ID of the TrainLayer to be created.
`train_data`	`data.frame` Data modality to be stored on the layer.
`varsel_package`	`character` Package name containing the variable selection algorithm function. Defaults to `NULL` if the function exists in the current working environment.

varsel_fct	character Variable selection function name. Default value is NULL for no variable selection. If specified, the function must accept at least two parameters: x (predictors) and y (response values), and return a vector of selected variables. Alternatively, use the interface parameters x_varsel and y_varsel to map the original argument names, and extract_var_fct to specify how to extract the vector of selected variables. An exception is made for the Boruta function, which includes an internal adjustment and requires no additional modifications.
varsel_param	list List of arguments to be passed to varsel_fct.
lrner_package	character Name of the package containing the learning algorithm function. Defaults to NULL if the function is available in the current working environment.
lrn_fct	character Name of the learning function. The function must accept at least two parameters: x (predictors) and y (response values) and return a model. Alternatively, use the interface parameters x_lrn and y_lrn to map these names to the original arguments in your function. The returned model must support the generic predict function (with arguments object and data) to generate predictions for new data. Predictions should be either a vector or a list containing a vector named predictions with the predicted values. If the arguments object and data have different names in your predict function, use the interface parameters below to map them to the original names. Additionally, if predictions are stored as a matrix or data.frame (e.g., predicted probabilities for dichotomous classification), only the second column (assumed to be class 1 probabilities) will be used. If the predicted values are not returned in one of the formats mentioned above, use the extract_pred_fct argument below to specify how to extract the predicted values from the prediction object.
param_train_list	character List of arguments to be passed to lrn_fct.
param_pred_list	character List of arguments to be passed to predict when generating predictions.
na_action	character Handling of missing values in data during training. Set to "na.keep" to retain missing values, or "na.rm" to remove instances with missing values.
x_varsel	character If the name of the argument used by the provided original variable selection function to pass the matrix of independent variable is not x, use this argument to specify how it is called in the provided function.
y_varsel	character If the name of the argument used by the provided original variable selection function to pass the target variable is not y, use this argument to specify how it is called in the provided function.
x_lrn	character If the name of the argument used by the provided original learning function to

	pass the matrix of independent variable is not `x`, use this argument to specify how it is called in the provided function.
`y_lrn`	`character` If the name of the argument used by the provided original learning function to pass the target variable is not `y`, use this argument to specify how it is called in the provided function.
`object`	`character` The generic function `predict` uses the parameter `object` to pass a model. If the corresponding argument is named differently in your `predict` function, specify its name.
`data`	`character` The generic function `predict` uses a parameter `data` to pass new data. If the corresponding argument is named differently in your predict function, specify the name.
`extract_pred_fct`	`character or function` If the `predict` function called for the model does not return a vector, use this argument to specify a function (or the name of a function) to extract the vector of predictions. The default value is `NULL` if predictions are returned as a vector.
`extract_var_fct`	`character or function` If the variable selection function does not return a vector, use this argument to specify a function (or the name of a function) to extract the vector of selected variables.

Value

The updated Training object (with the new layer) is returned.

References

Fouodo C.J.K, Bleskina M. and Szymczak S. (2024). fuseMLR: An R package for integrative prediction modeling of multi-omics data, paper submitted.

See Also

createTrainMetaLayer and fusemlr.

`createTrainMetaLayer` *createTrainMetaLayer*

Description

Creates and store a TrainMetaLayer on the Training object passed as argument. The meta-layer encapsulates the meta-learner and the fold predictions (internally created) of the layer-specific base models.

Usage

```
createTrainMetaLayer(
  training,
  meta_layer_id,
  lrner_package = NULL,
  lrn_fct,
  param_train_list = list(),
  param_pred_list = list(),
  na_action = "na.impute",
  x_lrn = "x",
  y_lrn = "y",
  object = "object",
  data = "data",
  extract_pred_fct = NULL
)
```

Arguments

`training`	`Training` Training object for storing the created meta-layer.
`meta_layer_id`	`character` ID of the layer to be created.
`lrner_package`	`character` Package name containing the variable selection algorithm function. Defaults to `NULL` if the function exists in the current working environment.
`lrn_fct`	`character` Name of the learning function. The function must accept at least two parameters: `x` (predictors) and `y` (response values), and return a model. If not, use the interface parameters `x_lrn` and `y_lrn` below to map these argument names to the original arguments in your function. The returned model must support the generic `predict` function (with arguments `object` and `data`) to make predictions for new data, and the predictions should be a vector or a `list` containing a vector called `predictions` with the predicted values. If the arguments `object` and `data` are named differently in your predict function, use the interface parameters `object` and `data` below to specify the original names. See the details below about meta-learners.
`param_train_list`	`character` List of arguments to be passed to `lrn_fct`.
`param_pred_list`	`list` List of arguments to be passed to `predict` when computing predictions.
`na_action`	`character` Handling of missing values in modality-specific predictions during training. Set to `"na.keep"` to keep missing values, `"na.rm"` to remove individuals with missing values or `"na.impute"` to impute missing values in modality-specific predictions. Only median and mode based imputations are actually handled. With

	the `"na.keep"` option, ensure that the provided meta-learner can handle missing values.
`x_lrn`	`character` If the argument name used by the provided original function to pass the matrix of independent variables is not `x`, use this argument to specify the name used in the function.
`y_lrn`	`character` If the argument name used by the provided original function to pass the target variable is not `y`, use this argument to specify the name used in the function.
`object`	`character` The generic function `predict` uses a parameter `object` to pass a model. If the corresponding argument is named differently in your predict function, specify the name.
`data`	`character` The generic function `predict` uses a parameter `data` to pass new data. If the corresponding argument is named differently in your predict function, specify the name.
`extract_pred_fct`	`character or function` If the predict function that is called for the model does not return a vector, then use this argument to specify a (or a name of a) function that can be used to extract vector of predictions. Defaults to NULL, if predictions are a vector.

Details

Internal meta-learners are available in the package.

The cobra meta-learner implements the COBRA (COmBined Regression Alternative), an aggregation method for combining predictions from multiple individual learners (Biau et al. 2014). This method aims to tune key parameters for achieving optimal predictions by averaging the target values of similar candidates in the training dataset's predictions. Only the training points that are sufficiently similar to the test point (based on the proximity threshold `epsilon`) are used for prediction. If no suitable training points are found, the function returns `NA`.

The weightedMeanLearner evaluates the prediction performance of modality-specific learners and uses these estimates to weight the base models, aggregating their predictions accordingly.

The bestLayerLearner evaluates the prediction performance of modality-specific learners and returns predictions made by the best learner as the meta-prediction.

Beyond the internal meta-learners, any other learning algorithm can be used.

Value

The updated Training object (with the new layer) is returned.

References

Fouodo C.J.K, Bleskina M. and Szymczak S. (2024). fuseMLR: An R package for integrative prediction modeling of multi-omics data, paper submitted.
Biau, G., Fischer, A., Guedj, B., & Malley, J. D. (2014). COBRA: A combined regression strategy. The Journal of Multivariate Analysis 46:18-28

See Also

createTrainLayer, varSelection, and fusemlr.

`createWeights` *Create weights for COBRA Predictions*

Description

The `createWeights` function is used to calculate weights for predictions.

Usage

```
createWeights(train, test, n_train, n_test, nlearners, eps, alpha)
```

Arguments

`train`	A `matrix` representing the training data. Rows represent observations, and columns contain predictions from individual learners for these observations. In cases where a prediction is unavailable for a specific observation, `NA` is used.
`test`	A `matrix` representing the test data. Rows represent observations, and columns contain predictions from individual learners for these observations. In cases where a prediction is unavailable for a specific observation, `NA` is used.
`n_train`	An `integer` specifying the number of training observations.
`n_test`	An `integer` specifying the number of test observations.
`nlearners`	An `integer` representing the number of learners.
`eps`	A `numeric` value representing the threshold for proximity between two predictions.
`alpha`	A value that determines the optimal number of learners in the neighborhood (only for alpha optimization).

`Data` *Abstract class Data*

Description

As abstract, a Data object cannot be stored on any layer. Instead, extended TrainData or TestData objects can be stored on a layer.

Methods

Public methods:

- `Data$new()`
- `Data$print()`
- `Data$getIndSubset()`
- `Data$impute()`
- `Data$getVarSubset()`
- `Data$getSetDiff()`
- `Data$getDataFrame()`
- `Data$setDataFrame()`
- `Data$getCompleteData()`
- `Data$getId()`
- `Data$getData()`
- `Data$getIndCol()`
- `Data$clone()`

Method `new()`**:** Constructor of class Data.

Usage:

```
Data$new(id, ind_col, data_frame)
```

Arguments:

`id` `character`
Object ID.

`ind_col` `character`
Column name containing individual IDs.

`data_frame` `data.frame`
`data.frame` containing data.

Method `print()`**:** Printer

Usage:

```
Data$print(...)
```

Arguments:

`...` `any`

Method `getIndSubset()`**:** Retrieve a data subset for a given variable name and values, a data subset.

Usage:

```
Data$getIndSubset(var_name, value)
```

Arguments:

`var_name` `character`
Variable name of interest.

`value` `vector`
Values of interest.

Returns: The data subset is returned.

Method `impute()`: Imputes missing values in modality-specific predictions. Only mode and median based imputations are actually supported.

Usage:

```
Data$impute(impute_fct, impute_param, target_name)
```

Arguments:

`impute_fct` `character`
An imputation function to use instead of median or mode imputation. Not yet implemented!

`impute_param` `list`

`target_name` `character`
Name of the target variable. The list of parameters to call the imputation function.

Returns: A new object with the predicted values is returned.

Method `getVarSubset()`: Retrieve a subset of variables from data.

Usage:

```
Data$getVarSubset(var_name)
```

Arguments:

`var_name` `character`
Variable names of interest.

Returns: The data subset is returned.

Method `getSetDiff()`: For the given variable name, non existing values in the current dataset are returned.

Usage:

```
Data$getSetDiff(var_name, value)
```

Arguments:

`var_name` `character`
Variable name of interest.

`value` `vector`
Values of interest.

Returns: The subset difference is returned.

Method `getDataFrame()`: Getter of the `data.frame`.

Usage:

```
Data$getDataFrame()
```

Returns: The `data.frame` of the current object is returned.

Method `setDataFrame()`: Set a new `data.frame` to the current object.

Usage:

```
Data$setDataFrame(data_frame)
```

Arguments:

`data_frame` `data.frame`

Returns: The current object is returned.

Method `getCompleteData()`**:** Getter of the complete dataset without missing values.

Usage:

```
Data$getCompleteData()
```

Returns: The complete dataset is returned.

Method `getId()`**:** Getter of the current object ID.

Usage:

```
Data$getId()
```

Returns: The current object ID is returned.

Method `getData()`**:** Getter of the current Data. This function is re-implemented by TrainData and TestData.

Usage:

```
Data$getData()
```

Returns: Do not use on this class.

Method `getIndCol()`**:** Getter of the individual column variable.

Usage:

```
Data$getIndCol()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

TrainData and TestData

`extractData` *extractData*

Description

Extracts data stored on each layers; base data and modality-specific predictions (for Training) are extracted.

Usage

```
extractData(object)
```

Arguments

`object`	`Training or Testing` The object of interest.

Value

A list of data is returned.

`extractModel` *extractModel*

Description

Extracts models stored on each layers; base and meta models are extracted.

Usage

```
extractModel(training)
```

Arguments

`training`	`Training` The Training object of interest.

Value

A list of models is returned.

`fusemlr` *fusemlr*

Description

Trains the Training object passed as argument. A training object must contain the training layers and a training meta-layer. A training layer encapsulates data modalities, a variable selection method and a learner. Use the function createTraining to create a training object, createTrainLayer to add training layers to the created training object, and createTrainMetaLayer to add a meta-layer with the corresponding meta-learner to the training object. The function `fusemlr` is designed to train all training layers and the meta-learner. After training the layer-specific base models and the meta-model will be stored in the training object which can be used for predictions.

Usage

```
fusemlr(
  training,
  ind_subset = NULL,
  use_var_sel = FALSE,
  resampling_method = NULL,
  resampling_arg = list(),
  seed = NULL
)
```

Arguments

`training`	`Training` Training object for storing training layers.
`ind_subset`	`vector` ID subset to be used for training.
`use_var_sel`	`boolean` If `TRUE` and no variable selection has been performed for the provide training object, then a variable selection will proceed the training. Otherwise, if variable selection has been previously performed, the selected variables will be used for training.
`resampling_method`	`function` Function for internal validation. If not specify, the `resampling` function from the package `caret` is used for a 10-folds cross-validation.
`resampling_arg`	`list` List of arguments to be passed to the function.
`seed`	`integer` Random seed used for resampling. Default is NULL, which generates the seed from `R`.

Value

The current object is returned, with each learner trained on each layer.

References

Fouodo C.J.K, Bleskina M. and Szymczak S. (2024). fuseMLR: An R package for integrative prediction modeling of multi-omics data, paper submitted.

See Also

createTrainLayer, createTrainMetaLayer, extractModel and extractData.

`HashTable` *Class HashTable*

Description

Hashtable to contain object modalities. Storage objects like Training and TrainLayer are extensions of this class.

Methods

Public methods:

- `HashTable$new()`
- `HashTable$print()`
- `HashTable$add2HashTable()`
- `HashTable$getFromHashTable()`
- `HashTable$getKeyClass()`
- `HashTable$removeFromHashTable()`
- `HashTable$getId()`
- `HashTable$getHashTable()`
- `HashTable$checkClassExist()`

Method `new()`**:** Initialize a default parameters list.

Usage:

```
HashTable$new(id)
```

Arguments:

`id` `character`
  ID of the hash table. It must be unique.

Method `print()`**:** Printer

Usage:

```
HashTable$print(...)
```

Arguments:

`...` `any`

Method `add2HashTable()`**:** Function to add a key-value pair to the hash table.

Usage:

```
HashTable$add2HashTable(key, value, .class)
```

Arguments:

`key` `character`
  The key to be added.

`value` `object`
  Object to be added.

`.class` `character`
Class of the object to be added.

Method `getFromHashTable():` Getter of the object which the key passed as argument.

Usage:

```
HashTable$getFromHashTable(key)
```

Arguments:

`key` `character`
Key of the required object.

Method `getKeyClass():` Getter of the `data.frame` that stores all key class pairs.

Usage:

```
HashTable$getKeyClass()
```

Returns: data.frame

Method `removeFromHashTable():` Remove the object with the corresponding key from the hashtable.

Usage:

```
HashTable$removeFromHashTable(key)
```

Arguments:

`key` Key of the object to be removed.

Method `getId():` Getter of the current object ID.

Usage:

```
HashTable$getId()
```

Method `getHashTable():` Getter of the current hashtable.

Usage:

```
HashTable$getHashTable()
```

Method `checkClassExist():` Check whether object from a class has already been stored.

Usage:

```
HashTable$checkClassExist(.class)
```

Arguments:

`.class` `character`

Returns: Boolean value

`Lrner` *Lrner Class*

Description

This class implements a learner. A Lrner object can only exist as a component of a TrainLayer or a TrainMetaLayer object.

Methods

Public methods:

- `Lrner$new()`
- `Lrner$print()`
- `Lrner$summary()`
- `Lrner$interface()`
- `Lrner$train()`
- `Lrner$getTrainLayer()`
- `Lrner$getNaRm()`
- `Lrner$getNaAction()`
- `Lrner$getId()`
- `Lrner$getPackage()`
- `Lrner$getIndSubset()`
- `Lrner$getVarSubset()`
- `Lrner$getParamPred()`
- `Lrner$getParamInterface()`
- `Lrner$getExtractPred()`

Method `new()`**:** Initialize a default parameters list.

Usage:

```
Lrner$new(
  id,
  package = NULL,
  lrn_fct,
  param_train_list,
  param_pred_list = list(),
  train_layer,
  na_action = "na.rm"
)
```

Arguments:

`id` `character`
Learner ID.

`package` `character`
Package that implements the learn function. If NULL, the

`lrn_fct` `character`
: learn function is called from the current environment.

`param_train_list` `list`
: List of parameter for training.

`param_pred_list` `list`
: List of parameter for testing. Learn parameters.

`train_layer` `TrainLayer`
: Layer on which the learner is stored.

`na_action` `character`
: Handling of missing values. Set to "na.keep" to keep missing values, "na.rm" to remove individuals with missing values or "na.impute" (only applicable on meta-data) to impute missing values in meta-data. Only median and mode based imputations are actually handled. With the "na.keep" option, ensure that the provided learner can handle missing values.

Method `print()`**:** Printer

Usage:

```
Lrner$print(...)
```

Arguments:

`...` any

Method `summary()`**:** Printer

Usage:

```
Lrner$summary(...)
```

Arguments:

`...` any

Method `interface()`**:** Learner and prediction parameter interface. Use this function to provide how the following parameters are named in the learning function (`lrn_fct`) you provided when creating the learner, or in the predicting function.

Usage:

```
Lrner$interface(
  x = "x",
  y = "y",
  object = "object",
  data = "data",
  extract_pred_fct = NULL
)
```

Arguments:

`x` `character`
: Name of the argument to pass the matrix of independent variables in the original learning function.

`y` `character`
: Name of the argument to pass the response variable in the original learning function.

`object` `character`
: Name of the argument to pass the model in the original predicting function.

`data` character
: Name of the argument to pass new data in the original predicting function.

`extract_pred_fct` character or `function`
: If the predict function that is called for the model does not return a vector, then use this argument to specify a (or a name of a) function that can be used to extract vector of predictions. Default value is NULL, if predictions are in a vector.

Method `train()`**:** Tains the current learner (from class Lrner) on the current training data (from class TrainData).

Usage:

```
Lrner$train(ind_subset = NULL, use_var_sel = FALSE, verbose = TRUE)
```

Arguments:

`ind_subset` vector
: Individual ID subset on which the training will be performed.

`use_var_sel` boolean
: If TRUE, variable selection is performed before training.

`verbose` boolean
: Warning messages will be displayed if set to TRUE.

Returns: The resulting model, from class Model, is returned.

Method `getTrainLayer()`**:** The current layer is returned.

Usage:

```
Lrner$getTrainLayer()
```

Returns: TrainLayer object.

Method `getNaRm()`**:** The current layer is returned.

Usage:

```
Lrner$getNaRm()
```

Method `getNaAction()`**:** The current layer is returned.

Usage:

```
Lrner$getNaAction()
```

Method `getId()`**:** Getter of the current learner ID.

Usage:

```
Lrner$getId()
```

Returns: The current learner ID.

Method `getPackage()`**:** Getter of the learner package implementing the learn function.

Usage:

```
Lrner$getPackage()
```

Returns: The name of the package implementing the learn function.

Method `getIndSubset()`**:** Getter of the learner package implementing the learn function.

Usage:

```
Lrner$getIndSubset()
```

Returns: The name of the package implementing the learn function.

Method `getVarSubset()`**:** Getter of the variable subset used for training.

Usage:

```
Lrner$getVarSubset()
```

Returns: The list of variables used for training is returned.

Method `getParamPred()`**:** Getter predicting parameter list.

Usage:

```
Lrner$getParamPred()
```

Returns: The list of predicting parameters.

Method `getParamInterface()`**:** The current parameter interface is returned.

Usage:

```
Lrner$getParamInterface()
```

Returns: A data.frame of interface.

Method `getExtractPred()`**:** The function to extract predicted values is returned.

Usage:

```
Lrner$getExtractPred()
```

Returns: A data.frame of interface.

`Model` *Model Class*

Description

This class implements a model. A Model object can only exist as element of a TrainLayer or a TrainMetaLayer object. A Model object is automatically created by fitting a learner on a training data.

A Model object can compute predictions for a TestData object. See the `predict` function below.

Methods

Public methods:

- `Model$new()`
- `Model$print()`
- `Model$summary()`
- `Model$getBaseModel()`
- `Model$getTrainData()`
- `Model$getTrainLabel()`

- `Model$getLrner()`
- `Model$setId()`
- `Model$predict()`
- `Model$clone()`

Method `new()`**:** Constructor of Model class.

Usage:

```
Model$new(lrner, train_data, base_model, train_layer)
```

Arguments:

`lrner` `Lrner`
  The learner.

`train_data` `TrainData(1)`
  Training data.

`base_model` `object`
  Base model as returned by the original learn function.

`train_layer` `TrainLayer`
  The current training layer on which the model is stored.

Returns: An object is returned.

Method `print()`**:** Printer

Usage:

```
Model$print(...)
```

Arguments:

`...` `any`

Method `summary()`**:** Summary

Usage:

```
Model$summary(...)
```

Arguments:

`...` `any`

Method `getBaseModel()`**:** Getter of the base model

Usage:

```
Model$getBaseModel()
```

Method `getTrainData()`**:** Getter of the traning data

Usage:

```
Model$getTrainData()
```

Method `getTrainLabel()`**:** Getter of the individual ID column in the training data.

Usage:

```
Model$getTrainLabel()
```

Arguments:

`...` any

Method `getLrner()`**:** Getter of the learner use to fit the model.

Usage:

`Model$getLrner()`

Method `setId()`**:** Setter of the model ID.

Usage:

`Model$setId(id)`

Arguments:

`id character`
ID value

Method `predict()`**:** Predict target values for the new data (from class TestData) taken as into.

Usage:

`Model$predict(testing_data, use_var_sel, ind_subset = NULL)`

Arguments:

`testing_data TestData`
An object from class TestData.

`use_var_sel boolean`
If TRUE, selected variables available at each layer are used.

`ind_subset vector`
Subset of individual IDs to be predicted.

`...` Further parameters to be passed to the basic predict function.

Returns: The predicted object are returned. The predicted object must be either a vector or a list containing a field predictions with predictions.

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

Usage:

`Model$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`multi_omics` *Simulated multiomics data for 70 training participants and 23 testing participants, each with an effect size of 20 on each layer. Each layer includes 50 participants for training and 20 for testing. Participants do not perfectly overlap across layers. The simulation is based on the R package* `interSIM`.

Description

The dataset is a list containing training and testing data, called `training` and `testing` respectively. Each data is a list containing the following multi_omics at each layer.

Usage

```
data(multi_omics)
```

Format

A list with training and testing data contaning methylation, gene expressions and protein expressions data.

Details

- `methylation`: A `data.frame` containing the simulated methylation dataset.
- `genexpr` : A `data.frame` containing the gene expression dataset.
- `proteinexpr`: A `data.frame` containing the protein expression dataset.
- `target`: A `data.frame` with two columns, containing patient IDs and values of target variable.

`predict.bestLayerLearner` *Best specific Learner prediction.*

Description

Predict function for models from class `bestLayerLearner`.

Usage

```
## S3 method for class 'bestLayerLearner'
predict(object, data, ...)
```

Arguments

`object`	`bestLayerLearner` An object from class bestLayerLearner
`data`	`data.frame` New data to predicted.
`...`	`any` Further arguments passed to or from other methods.

Value

Predicted target values are returned.

Examples

```
set.seed(20240625)
x = data.frame(x1 = runif(n = 50L, min = 0, max = 1))
y <- sample(x = 0:1, size = 50L, replace = TRUE)
my_model <- bestLayerLearner(x = x, y = y)
x_new <- data.frame(x1 = rnorm(10L))
my_predictions <- predict(object = my_model, data = x_new)
```

---

predict.cobra *Predict Using COBRA object*

Description

#' The `predict.cobra` function makes predictions on new data using a trained COBRA object.

Usage

```
## S3 method for class 'cobra'
predict(object, data, ...)
```

Arguments

`object`	An object of class "cobra" created by the `cobra` function.
`data`	A `data.frame` of new data, where rows are observations and columns are predictions from individual learners. Use `NA` for missing predictions.
`...`	Additional arguments (currently not used).

Value

A vector of predictions for the new data.

Examples

```
# Example usage
set.seed(123)
x_train <- data.frame(a = rnorm(10L), b = rnorm(10L))
y_train <- sample(0L:1L, size = 10L, replace = TRUE)

# Train the model with epsilon optimization
cobra_model <- cobra(x = x_train, y = y_train, tune = "epsilon")

# Make predictions on new data
set.seed(156)
x_new <- data.frame(a = rnorm(5L), b = rnorm(5L))
prediction <- predict(object = cobra_model, data = x_new)
```

`predict.Training` *predict.Training*

Description

Computes predictions for the Testing object passed as argument.

Usage

```
## S3 method for class 'Training'
predict(object, testing, ind_subset = NULL, ...)
```

Arguments

`object`	`Training` A trained Training object to be used to compute predictions.
`testing`	`Testing` A new testing object to be predicted.
`ind_subset`	`vector` Vector of IDs to be predicted.
`...`	`any` Further arguments passed to or from other methods.

Value

The final predicted object. All layers and the meta layer are predicted.

`predict.weightedMeanLearner` *Weighted mean prediction.*

Description

Predict function for models from class `weightedMeanLearner`.

Usage

```
## S3 method for class 'weightedMeanLearner'
predict(object, data, na_rm = FALSE, ...)
```

Arguments

`object`	`weightedMeanLearner(1)` An object from class weightedMeanLearner
`data`	`data.frame` `data.frame` to be predicted.
`na_rm`	`boolean` Removes NAs when TRUE.
`...`	`any` Further arguments.

Value

Predicted target values are returned.

Examples

```
set.seed(20240625)
x <- data.frame(x1 = rnorm(50L))
y <- sample(x = 0:1, size = 50L, replace = TRUE)
my_model <- weightedMeanLearner(x = x, y = y)
x_new <- data.frame(x1 = rnorm(10L))
my_predictions <- predict(object = my_model, data = x_new)
```

`PredictData` *PredictData Class*

Description

This class implements PredictData object to be predicted. A PredictData object can only exist as a component of a PredictLayer or a PredictMetaLayer object.

Super class

`fuseMLR::Data` -> `PredictData`

Methods

Public methods:

- `PredictData$new()`
- `PredictData$print()`
- `PredictData$getPredictData()`
- `PredictData$getPredictLayer()`
- `PredictData$setPredictLayer()`
- `PredictData$clone()`

Method `new()`**:** Initialize a new object from the current class.

Usage:

```
PredictData$new(id, ind_col, data_frame)
```

Arguments:

`id` `character`
: Object ID.

`ind_col` `character`
: Column name containing individual IDs.

`data_frame` `data.frame`
: `data.frame` containing data.

Method `print()`**:** Printer

Usage:

```
PredictData$print(...)
```

Arguments:

`...` `any`

Method `getPredictData()`**:** Getter of the current predicted `data.frame` wihtout individual ID variable.

Usage:

```
PredictData$getPredictData()
```

Returns: The `data.frame` without individual ID nor target variables is returned.

Method `getPredictLayer()`**:** Getter of the current layer.

Usage:

```
PredictData$getPredictLayer()
```

Returns: The layer (from class PredictLayer) on which the current train data are stored is returned.

Method `setPredictLayer()`**:** Assigns a predicted layer to the predicted data.

Usage:

```
PredictData$setPredictLayer(predict_layer)
```

Arguments:

`predict_layer` `PredictLayer(1)`

Returns: The current object

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

TrainData, TestData

`Predicting` *Predicting Class*

Description

This class is designed for predictions.

The Predicting is structured as followed:

- PredictLayer: Exists for each modality.
  - PredictData: Related class for modality-specific predictions.
- PredictMetaLayer: Related class for meta predictions.
  - PredictData: Specific to the meta layer, it is set up internally after cross-validation.

Use the function `train` for training and `predict` for predicting.

TODO: Do not export me.

Super class

`fuseMLR::HashTable` -> `Predicting`

Methods

Public methods:

- `Predicting$new()`
- `Predicting$print()`
- `Predicting$createMetaTestData()`
- `Predicting$getIndIDs()`
- `Predicting$getPredictMetaLayer()`
- `Predicting$getIndCol()`

Method `new()`**:** constructor

Usage:

```
Predicting$new(id, ind_col)
```

Arguments:

`id` `character`
Predicting id.

`ind_col` `character` Name of column of individuals IDS

Method `print()`**:** Printer

Usage:

```
Predicting$print(...)
```

Arguments:

`...` any

Method `createMetaTestData():` Creates a new modality-specific predictions dataset based on layer predictions.

Usage:

```
Predicting$createMetaTestData(meta_layer_id)
```

Arguments:

`meta_layer_id (character(1))`
ID of the meta layer where the testing meta data will be stored.

Returns: A TestData is returned.

Method `getIndIDs():` Gather individual IDs from all layer.

Usage:

```
Predicting$getIndIDs()
```

Returns: A `data.frame` containing individuals IDs.

Method `getPredictMetaLayer():` Getter of the meta layer.

Usage:

```
Predicting$getPredictMetaLayer()
```

Returns: Object from class PredictMetaLayer

Method `getIndCol():` Getter of the individual column name.

Usage:

```
Predicting$getIndCol()
```

See Also

TrainLayer

`PredictLayer` *PredictLayer Class*

Description

This class implements a layer. A PredictLayer object can only exist as a component of a Predicting object.

A predicted layer can only contain PredictData.

Super class

`fuseMLR::HashTable` -> `PredictLayer`

Methods

Public methods:

- `PredictLayer$new()`
- `PredictLayer$print()`
- `PredictLayer$getPredicting()`
- `PredictLayer$getIndIDs()`
- `PredictLayer$getPredictData()`
- `PredictLayer$setPredicting()`
- `PredictLayer$summary()`

Method `new()`**:** constructor

Usage:

```
PredictLayer$new(id)
```

Arguments:

`id` `character`
The layer ID.

Method `print()`**:** Printer

Usage:

```
PredictLayer$print(...)
```

Arguments:

`...` any

Method `getPredicting()`**:** Getter of the current predicting object

Usage:

```
PredictLayer$getPredicting()
```

Returns: The current predicting object is returned.

Method `getIndIDs()`**:** Getter of IDS from the current layer.

Usage:

```
PredictLayer$getIndIDs()
```

Returns: A `data.frame` containing individuals IDs values.

Method `getPredictData()`**:** Getter of the predicted data stored on the current layer.

Usage:

```
PredictLayer$getPredictData()
```

Returns: The stored PredictData object is returned.

Method `setPredicting()`**:** Assigns a predicting object to the predicted layer.

Usage:

```
PredictLayer$setPredicting(predicting)
```

Arguments:

`predicting` Predicting

Returns: The current object

Method `summary()`**:** Generate summary.

Usage:

```
PredictLayer$summary()
```

See Also

Training, Lrner, TrainData, TestData and Model

`PredictMetaLayer` *PredictMetaLayer Class*

Description

This class implement a predicted meta layer. A PredictMetaLayer can only exist as unique element of a Training object.

A predicted meta layer can only contain a PredictData object.

Super class

`fuseMLR::HashTable` -> `PredictMetaLayer`

Methods

Public methods:

- `PredictMetaLayer$new()`
- `PredictMetaLayer$print()`
- `PredictMetaLayer$getPredicting()`
- `PredictMetaLayer$getIndIDs()`
- `PredictMetaLayer$getPredictData()`
- `PredictMetaLayer$openAccess()`
- `PredictMetaLayer$closeAccess()`
- `PredictMetaLayer$getAccess()`

Method `new()`**:** constructor

Usage:

```
PredictMetaLayer$new(id, predicting)
```

Arguments:

`id` character

`predicting` Predicting

Method `print()`**:** Printer

Usage:

```
PredictMetaLayer$print(...)
```

Arguments:

`...` any

Method `getPredicting()`**:** Getter of the current predicting object

Usage:

```
PredictMetaLayer$getPredicting()
```

Returns: The current predicting object is returned.

Method `getIndIDs()`**:** Getter of IDS from the current layer.

Usage:

```
PredictMetaLayer$getIndIDs()
```

Returns: A `data.frame` containing individuals IDs values.

Method `getPredictData()`**:** Getter of the predicted data.

Usage:

```
PredictMetaLayer$getPredictData()
```

Returns: The stored PredictData object is returned.

Method `openAccess()`**:** Open access to the meta layer. A meta learner is only modifiable if the access is opened.

Usage:

```
PredictMetaLayer$openAccess()
```

Method `closeAccess()`**:** Close access to the meta layer to avoid accidental modification.

Usage:

```
PredictMetaLayer$closeAccess()
```

Method `getAccess()`**:** Getter of the current access to the meta layer.

Usage:

```
PredictMetaLayer$getAccess()
```

`summary.Testing` *Testing object Summaries*

Description

Summaries a `fuseMLR` Testing object.

Usage

```
## S3 method for class 'Testing'
summary(object, ...)
```

Arguments

`object`	`Testing` The Testing object of interest.
`...`	`any` Further arguments.

`summary.Training` *Training object Summaries*

Description

Summaries a `fuseMLR` Training object.

Usage

```
## S3 method for class 'Training'
summary(object, ...)
```

Arguments

`object`	`Training` The Training object of interest.
`...`	`any` Further arguments.

`Target` *Target Class*

Description

This class implements the target object. A Target object can only exist as a component of a Training object.

Super class

`fuseMLR::Data` -> `Target`

Methods

Public methods:

- `Target$new()`
- `Target$print()`
- `Target$summary()`
- `Target$getData()`
- `Target$getTargetValues()`
- `Target$getTargetName()`
- `Target$getTraining()`
- `Target$setData()`
- `Target$clone()`

Method `new()`**:** Initialize a new object from the current class.

Usage:

```
Target$new(id, data_frame, training)
```

Arguments:

`id character`
The Object ID.

`data_frame data.frame`
`data.frame` containing data.

`training Training`
Training where to store the current object.

Method `print()`**:** Printer

Usage:

```
Target$print(...)
```

Arguments:

`... any`

Method `summary()`**:** Summary

Usage:

```
Target$summary(...)
```

Arguments:

`...` any

Method `getData():` Getter of the current `data.frame` wihtout individual ID nor target variables.

Usage:

```
Target$getData()
```

Returns: The `data.frame` without individual ID nor target variables is returned.

Method `getTargetValues():` Getter of target values stored on the current training layer.

Usage:

```
Target$getTargetValues()
```

Returns: The observed target values stored on the current training layer are returned.

Method `getTargetName():` Getter of the target variable name.

Usage:

```
Target$getTargetName()
```

Method `getTraining():` Getter of the current training object.

Usage:

```
Target$getTraining()
```

Returns: The training layer (from class Training) on which the current train data are stored is returned.

Method `setData():` Getter of the current `data.frame` wihtout individual ID nor target variables.

Usage:

```
Target$setData(data_frame)
```

Arguments:

`data_frame data.frame`
data.frame to be set.
Title

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

TrainLayer, Lrner, Model, TestData

`TestData` *TestData Class*

Description

This class implements TestData object to be predicted. A TestData object can only exist as a component of a TestLayer or a TestMetaLayer object.

Super class

`fuseMLR::Data` -> `TestData`

Methods

Public methods:

- `TestData$new()`
- `TestData$print()`
- `TestData$getData()`
- `TestData$getTestLayer()`
- `TestData$clone()`

Method `new()`**:** Initialize a new object from the current class.

Usage:

```
TestData$new(id, data_frame, new_layer)
```

Arguments:

`id` `character`
Object ID.

`data_frame` `data.frame`
`data.frame` containing data.

`new_layer` `TestLayer`
Layer where to store the current object.

`ind_col` `character`
Column name containing individual IDs.

Method `print()`**:** Printer

Usage:

```
TestData$print(...)
```

Arguments:

`...` `any`

Method `getData()`**:** Getter of the current `data.frame` wihtout individual ID variable.

Usage:

```
TestData$getData()
```

Returns: The `data.frame` without individual ID nor target variables is returned.

Method `getTestLayer()`**:** Getter of the current layer.

Usage:

```
TestData$getTestLayer()
```

Returns: The layer (from class TestLayer) on which the current train data are stored is returned.

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

Usage:

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

Arguments:

`deep`	Whether to make a deep clone.

See Also

TrainData

`Testing` *Testing Class*

Description

This is a primary classes of fuseMLR. An object from this class is designed to contain multiple layers, but only one new meta layer.

A Testing object is structured as followed:

- TestLayer
- TestMetaLayer

Super class

`fuseMLR::HashTable` -> `Testing`

Methods

Public methods:

- `Testing$new()`
- `Testing$print()`
- `Testing$getIndIDs()`
- `Testing$getTestMetaLayer()`
- `Testing$getIndCol()`
- `Testing$getVerbose()`
- `Testing$getData()`
- `Testing$upset()`

- `Testing$summary()`

Method `new()`**:** constructor

Usage:

`Testing$new(id, ind_col, verbose = TRUE)`

Arguments:

`id` `character`
    Testing id.

`ind_col` `character` Name of column of individuals IDS in testing data.frame.

`verbose` `boolean`
    Warning messages will be displayed if set to TRUE.

Method `print()`**:** Printer

Usage:

`Testing$print(...)`

Arguments:

`...` `any`

Method `getIndIDs()`**:** Gather individual IDs from all layer.

Usage:

`Testing$getIndIDs()`

Returns: A `data.frame` containing individuals IDs.

Method `getTestMetaLayer()`**:** Getter of the meta layer.

Usage:

`Testing$getTestMetaLayer()`

Returns: Object from class TestMetaLayer

Method `getIndCol()`**:** Getter of the individual column name.

Usage:

`Testing$getIndCol()`

Method `getVerbose()`**:** Getter of the verbose setting.

Usage:

`Testing$getVerbose()`

Method `getData()`**:** Retrieve modality-specific prediction data.

Usage:

`Testing$getData()`

Returns: A `list` containing all (base and meta) models.

Method `upset()`**:** UpSet plot to show an overview of the overlap of individuals across various layers.

Usage:

`Testing$upset(...)`

Arguments:

`...` `any`
Further parameters to be passed to the the `upset` function from package `UpSetR`.

Method `summary()`**:** Generate testing summary

Usage:

`Testing$summary()`

See Also

TrainLayer

`TestLayer` *TestLayer Class*

Description

This class implements a layer. A TestLayer object can only exist as a component of a Predicting object.

A predicted layer can only contain TestData.

Super class

`fuseMLR::HashTable` -> `TestLayer`

Methods

Public methods:

- `TestLayer$new()`
- `TestLayer$print()`
- `TestLayer$getTesting()`
- `TestLayer$getIndIDs()`
- `TestLayer$getTestData()`
- `TestLayer$checkTestDataExist()`
- `TestLayer$summary()`

Method `new()`**:** constructor

Usage:

`TestLayer$new(id, testing)`

Arguments:

`id` `character`
Testing layer id.

`testing` Testing

Method `print()`**:** Printer

Usage:

```
TestLayer$print(...)
```

Arguments:

`...` any

Method `getTesting()`**:** Getter of the current Testing object.

Usage:

```
TestLayer$getTesting()
```

Returns: The current Testing object is returned.

Method `getIndIDs()`**:** Getter of IDS from the current layer.

Usage:

```
TestLayer$getIndIDs()
```

Returns: A `data.frame` containing individuals IDs values.

Method `getTestData()`**:** Getter of the predicted data stored on the current layer.

Usage:

```
TestLayer$getTestData()
```

Returns: The stored TestData object is returned.

Method `checkTestDataExist()`**:** Check whether a new data has been already stored.

Usage:

```
TestLayer$checkTestDataExist()
```

Returns: Boolean value

Method `summary()`**:** Generate summary.

Usage:

```
TestLayer$summary()
```

See Also

Training, Lrner, TrainData, TestData and Model

`TestMetaLayer` *TestMetaLayer Class*

Description

This class implement a predicted meta layer. A TestMetaLayer can only exist as unique element of a Training object.

A predicted meta layer can only contain a TestData object.

Super class

`fuseMLR::HashTable` -> `TestMetaLayer`

Methods

Public methods:

- `TestMetaLayer$new()`
- `TestMetaLayer$print()`
- `TestMetaLayer$getTesting()`
- `TestMetaLayer$getTestData()`
- `TestMetaLayer$openAccess()`
- `TestMetaLayer$closeAccess()`
- `TestMetaLayer$getAccess()`
- `TestMetaLayer$setTestData()`
- `TestMetaLayer$checkTestDataExist()`

Method `new()`**:** constructor

Usage:

```
TestMetaLayer$new(id, testing)
```

Arguments:

`id` `character`
Testing meta-layer id.

`testing` `Testing`

Method `print()`**:** Printer

Usage:

```
TestMetaLayer$print(...)
```

Arguments:

`...` `any`

Method `getTesting()`**:** Getter of the current testing object.

Usage:

```
TestMetaLayer$getTesting()
```

Returns: The current testing object is returned.

Method `getTestData()`**:** Getter of the training dataset stored on the current layer.

Usage:

```
TestMetaLayer$getTestData()
```

Returns: The stored TestData object is returned.

Method `openAccess()`**:** Open access to the meta layer. A meta learner is only modifiable if the access is opened.

Usage:

```
TestMetaLayer$openAccess()
```

Method `closeAccess()`**:** Close access to the meta layer to avoid accidental modification.

Usage:

```
TestMetaLayer$closeAccess()
```

Method `getAccess()`**:** Getter of the current access to the meta layer.

Usage:

```
TestMetaLayer$getAccess()
```

Method `setTestData()`**:** Create and set an TestData object to the current new meta learner.

Usage:

```
TestMetaLayer$setTestData(id, ind_col, data_frame)
```

Arguments:

`id character(1)`
  ID of the TestData object to be instanciated.

`ind_col character(1)`
  Name of individual column IDs.

`data_frame data.frame(1)`
  `data.frame` of layer specific predictions.

Method `checkTestDataExist()`**:** Check whether a new data has been already stored.

Usage:

```
TestMetaLayer$checkTestDataExist()
```

Returns: Boolean value

`TrainData` *TrainData Class*

Description

This class implements the training data. A TrainData object can only exist as a component of a TrainLayer or a TrainMetaLayer object.

Super class

`fuseMLR::Data` -> `TrainData`

Methods

Public methods:

- `TrainData$new()`
- `TrainData$print()`
- `TrainData$summary()`
- `TrainData$getData()`
- `TrainData$getTargetValues()`
- `TrainData$getTargetName()`
- `TrainData$getTrainLayer()`
- `TrainData$getTestLayer()`
- `TrainData$setDataFrame()`
- `TrainData$clone()`

Method `new()`**:** Initialize a new object from the current class.

Usage:

```
TrainData$new(id, data_frame, train_layer)
```

Arguments:

`id` `character`
The Object ID.

`data_frame` `data.frame`
`data.frame` containing data.

`train_layer` `TrainLayer`
Training layer where to store the current object.

Method `print()`**:** Printer

Usage:

```
TrainData$print(...)
```

Arguments:

`...` `any`

Method `summary()`**:** Summary

Usage:

```
TrainData$summary(...)
```

Arguments:

`...` any

Method `getData():` Getter of the current `data.frame` wihtout individual ID nor target variables.

Usage:

```
TrainData$getData()
```

Returns: The `data.frame` without individual ID nor target variables is returned.

Method `getTargetValues():` Getter of target values stored on the current training layer.

Usage:

```
TrainData$getTargetValues()
```

Returns: The observed target values stored on the current training layer are returned.

Method `getTargetName():` Getter of the target variable name.

Usage:

```
TrainData$getTargetName()
```

Method `getTrainLayer():` Getter of the current training layer.

Usage:

```
TrainData$getTrainLayer()
```

Returns: The training layer (from class TrainLayer) on which the current train data are stored is returned.

Method `getTestLayer():` Getter of the current layer.

Usage:

```
TrainData$getTestLayer()
```

Returns: The layer (from class TestLayer) on which the current train data are stored is returned.

Method `setDataFrame():` Set a new `data.frame` to the current object.

Usage:

```
TrainData$setDataFrame(data_frame)
```

Arguments:

`data_frame` `data.frame`

Returns: The current object is returned.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

TrainLayer, Lrner, Model, TestData

`Training` *Training Class*

Description

This is a primary classes of fuseMLR. An object from this class is designed to contain multiple training layers, but only one meta training layer.

The Training class is structured as followed:

- TrainLayer: Specific layer containing:
  - Lrner: Specific learner. This must be set by the user.
  - TrainData: Specific training dataset. This must be set up by the user.
  - Model: Specific model. This is set up by training the learner on the training data.
- TrainMetaLayer: Basically a TrainLayer, but with some specific properties.
  - Lrner: This is the meta learner, it must be set up by the user.
  - TrainData: Specific modality-specific prediction data. This is set up internally after cross-validation.
  - Model: Specific meta model. This is set up by training the learner on the training data.

Use the function `train` for training and `predict` for predicting.

Super class

`fuseMLR::HashTable` -> `Training`

Methods

Public methods:

- `Training$new()`
- `Training$print()`
- `Training$trainLayer()`
- `Training$predictLayer()`
- `Training$createMetaTrainData()`
- `Training$train()`
- `Training$predict()`
- `Training$varSelection()`
- `Training$getTargetValues()`
- `Training$getIndIDs()`
- `Training$getLayer()`
- `Training$getTrainMetaLayer()`
- `Training$getModel()`
- `Training$getData()`
- `Training$removeLayer()`

- `Training$removeTrainMetaLayer()`
- `Training$getIndCol()`
- `Training$getTarget()`
- `Training$getVerbose()`
- `Training$getUseVarSel()`
- `Training$getVarSelDone()`
- `Training$increaseNbTrainedLayer()`
- `Training$checkTargetExist()`
- `Training$getTargetObj()`
- `Training$getProblemTyp()`
- `Training$setImpute()`
- `Training$testOverlap()`
- `Training$upset()`
- `Training$summary()`

Method `new()`**:** constructor

Usage:

```
Training$new(
  id,
  ind_col,
  target,
  target_df,
  problem_type = "classification",
  verbose = TRUE
)
```

Arguments:

`id` `character`

`ind_col` `character`
: Name of column of individuals IDS.

`target` `character`
: Name of the target variable.

`target_df` `data.frame`
: Data frame with two columns: individual IDs and response variable values.

`problem_type` `character`
: Either "classification" or "regression".

`verbose` `boolean`
: Warning messages will be displayed if set to TRUE.

Method `print()`**:** Printer

Usage:

```
Training$print(...)
```

Arguments:

`...` `any`

Method `trainLayer()`**:** Train each layer of the current Training.

Usage:

```
Training$trainLayer(ind_subset = NULL, use_var_sel = FALSE, verbose = TRUE)
```

Arguments:

`ind_subset character`
Subset of individuals IDs to be used for training.

`use_var_sel boolean`
If TRUE, selected variables available at each layer are used.

`verbose boolean`
Warning messages will be displayed if set to TRUE.

Returns: Returns the object itself, with a model for each layer.

Method `predictLayer()`**:** Predicts values given new data.

Usage:

```
Training$predictLayer(testing, ind_subset = NULL)
```

Arguments:

`testing TestData`
Object of class TestData.

`ind_subset vector`
Subset of individuals IDs to be used for training.

Returns: A new Training with predicted values for each layer.

Method `createMetaTrainData()`**:** Creates a meta training dataset and assigns it to the meta layer.

Usage:

```
Training$createMetaTrainData(
  resampling_method,
  resampling_arg,
  use_var_sel,
  impute = TRUE
)
```

Arguments:

`resampling_method function`
Function for internal validation.

`resampling_arg list`
List of arguments to be passed to the function.

`use_var_sel boolean`
If TRUE, selected variables available at each layer are used.

`impute boolean`
If TRUE, mode or median based imputation is performed on the modality-specific predictions.

Returns: The current object is returned, with a meta training dataset assigned to the meta layer.

Method `train()`**:** Trains the current object. All leaners and the meta learner are trained.

Usage:

```
Training$train(
  ind_subset = NULL,
  use_var_sel = FALSE,
  resampling_method = NULL,
  resampling_arg = list(),
  seed = NULL
)
```

Arguments:

`ind_subset` `vector`
ID subset to be used for training.

`use_var_sel` `boolean`
If TRUE, variable selection is performed before training.

`resampling_method` `function`
Function for internal validation. If not specify, the `resampling` function from the package `caret` is used for a 10-folds cross-validation.

`resampling_arg` `list`
List of arguments to be passed to the function.

`seed` `integer`
Random seed. Default is NULL, which generates the seed from `R`.

Returns: The current object is returned, with each learner trained on each layer.

Method `predict()`**:** Compute predictions for a testing object.

Usage:

```
Training$predict(testing, ind_subset = NULL)
```

Arguments:

`testing` `Testing`
A new testing object to be predicted.

`ind_subset` `vector`
Vector of IDs to be predicted.

Returns: The predicted object. All layers and the meta layer are predicted. This is the final predicted object.

Method `varSelection()`**:** Variable selection on the current training object.

Usage:

```
Training$varSelection(ind_subset = NULL, verbose = TRUE)
```

Arguments:

`ind_subset` `vector`
ID subset of individuals to be used for variable selection.

`verbose` `boolean`
Warning messages will be displayed if set to TRUE.

Returns: The current layer is returned with the resulting model.

Method `getTargetValues()`**:** Gather target values from all layer.

Usage:

`Training$getTargetValues()`

Returns: A `data.frame` containing individuals IDs and corresponding target values.

Method `getIndIDs()`**:** Gather individual IDs from all layer.

Usage:

`Training$getIndIDs()`

Returns: A `data.frame` containing individuals IDs.

Method `getLayer()`**:** Get a layer of a given ID.

Usage:

`Training$getLayer(id)`

Arguments:

`id character`
The ID of the layer to be returned.

Returns: The TrainLayer object is returned for the given ID.

Method `getTrainMetaLayer()`**:** Getter of the meta layer.

Usage:

`Training$getTrainMetaLayer()`

Returns: Object from class TrainMetaLayer

Method `getModel()`**:** Retrieve models from all layer.

Usage:

`Training$getModel()`

Returns: A `list` containing all (base and meta) models.

Method `getData()`**:** Retrieve modality-specific predictions.

Usage:

`Training$getData()`

Returns: A `list` containing all (base and meta) models.

Method `removeLayer()`**:** Remove a layer of a given ID.

Usage:

`Training$removeLayer(id)`

Arguments:

`id character`
The ID of the layer to be removed.

Returns: The TrainLayer object is returned for the given ID.

Method `removeTrainMetaLayer()`**:** Remove the meta layer from the current Training object.

Usage:

`Training$removeTrainMetaLayer()`

Method `getIndCol()`**:** Getter of the individual column name.

Usage:

```
Training$getIndCol()
```

Method `getTarget()`**:** Getter of the target variable name.

Usage:

```
Training$getTarget()
```

Method `getVerbose()`**:** Getter of the verbose setting.

Usage:

```
Training$getVerbose()
```

Method `getUseVarSel()`**:** Getter of the use_var_sel field.

Usage:

```
Training$getUseVarSel()
```

Method `getVarSelDone()`**:** Getter of the use_var_sel field.

Usage:

```
Training$getVarSelDone()
```

Method `increaseNbTrainedLayer()`**:** Increase the number of trained layer.

Usage:

```
Training$increaseNbTrainedLayer()
```

Method `checkTargetExist()`**:** Check whether a target object has already been stored.

Usage:

```
Training$checkTargetExist()
```

Returns: Boolean value

Method `getTargetObj()`**:** Getter of the target object.

Usage:

```
Training$getTargetObj()
```

Method `getProblemTyp()`**:** Getter of the problem type.

Usage:

```
Training$getProblemTyp()
```

Method `setImpute()`**:** Set imputation action na.action.

Usage:

```
Training$setImpute(impute)
```

Arguments:

`impute` `character`
    How to handle missing values.

Method `testOverlap()`**:** Test that individuals overlap over layers. At least five individuals must overlapped.

Usage:

```
Training$testOverlap()
```

Method `upset()`**:** UpSet plot to show an overview of the overlap of individuals across various layers.

Usage:

```
Training$upset(...)
```

Arguments:

`...` any
Further parameters to be passed to the `upset` function from package `UpSetR`.

Method `summary()`**:** Generate training summary

Usage:

```
Training$summary()
```

See Also

TrainLayer

Testing and Predicting

`TrainLayer` *TrainLayer Class*

Description

This class implements a traning layer. A TrainLayer object can only exist as a component of a Training object.

A training layer is structured as followed:

- TrainData: Data to be used to train the learner.
- Lrner: Includes a learning function and the package implementing the function.
- Model: The result of training the learner on the training data.
- VarSel: Includes a variable selection function and the package implementing the function.

A training layer can train its learner on its training data and store the resulting model. See the public function `Layer$train()` below.

A training layer can make predictions for a new layer passed as argument to its predict function. See the public function `Layer$predict()` below.

Super class

`fuseMLR::HashTable` -> `TrainLayer`

Methods

Public methods:

- `TrainLayer$new()`
- `TrainLayer$print()`
- `TrainLayer$getTraining()`
- `TrainLayer$getTargetObj()`
- `TrainLayer$train()`
- `TrainLayer$varSelection()`
- `TrainLayer$predict()`
- `TrainLayer$getTrainData()`
- `TrainLayer$getTargetValues()`
- `TrainLayer$getIndIDs()`
- `TrainLayer$getTestData()`
- `TrainLayer$getLrner()`
- `TrainLayer$getVarSel()`
- `TrainLayer$getModel()`
- `TrainLayer$checkLrnerExist()`
- `TrainLayer$checkModelExist()`
- `TrainLayer$checkVarSelExist()`
- `TrainLayer$checkTrainDataExist()`
- `TrainLayer$summary()`

Method `new()`**:** constructor

Usage:

```
TrainLayer$new(id, training)
```

Arguments:

`id` `character`
  Training layer id.

`training` `Training`

Method `print()`**:** Printer

Usage:

```
TrainLayer$print(...)
```

Arguments:

`...` `any`

Method `getTraining()`**:** Getter of the current training object.

Usage:

```
TrainLayer$getTraining()
```

Returns: The current training object is returned.

Method `getTargetObj()`**:** Getter of the target object.

Usage:

```
TrainLayer$getTargetObj()
```

Method `train()`**:** Trains the current layer.

Usage:

```
TrainLayer$train(ind_subset = NULL, use_var_sel = FALSE, verbose = TRUE)
```

Arguments:

`ind_subset` `vector`
ID subset of individuals to be used for training.

`use_var_sel` `boolean`
If TRUE, variable selection is performed before training.

`verbose` `boolean`
Warning messages will be displayed if set to TRUE.

Returns: The current layer is returned with the resulting model.

Method `varSelection()`**:** Variable selection on the current layer.

Usage:

```
TrainLayer$varSelection(ind_subset = NULL, verbose = TRUE)
```

Arguments:

`ind_subset` `vector`
ID subset of individuals to be used for variable selection.

`verbose` `boolean`
Warning messages will be displayed if set to TRUE.

Returns: The current layer is returned with the resulting model.

Method `predict()`**:** Predicts values for the new layer taking as argument.

Usage:

```
TrainLayer$predict(new_layer, use_var_sel, ind_subset = NULL)
```

Arguments:

`new_layer` `TrainLayer`

`use_var_sel` `boolean`
If TRUE, selected variables available at each layer are used.

`ind_subset` `vector`

Returns: A new PredictLayer object with the predicted data is returned.

Method `getTrainData()`**:** Getter of the training dataset stored on the current layer.

Usage:

```
TrainLayer$getTrainData()
```

Returns: The stored TrainData object is returned.

Method `getTargetValues()`**:** Getter of target values from the current layer.

Usage:

```
TrainLayer$getTargetValues()
```

Returns: A `data.frame` containing individuals IDs and corresponding target values.

Method `getIndIDs()`**:** Getter of IDS from the current layer.

Usage:

```
TrainLayer$getIndIDs()
```

Returns: A `data.frame` containing individuals IDs values.

Method `getTestData()`**:** Getter of the new data.

Usage:

```
TrainLayer$getTestData()
```

Returns: The stored TestData object is returned.

Method `getLrner()`**:** Getter of the learner.

Usage:

```
TrainLayer$getLrner()
```

Returns: The stored Lrner object is returned.

Method `getVarSel()`**:** Getter of the variable selector.

Usage:

```
TrainLayer$getVarSel()
```

Returns: The stored VarSel object is returned.

Method `getModel()`**:** Getter of the model.

Usage:

```
TrainLayer$getModel()
```

Returns: The stored Model object is returned.

Method `checkLrnerExist()`**:** Check whether a learner has been already stored.

Usage:

```
TrainLayer$checkLrnerExist()
```

Returns: Boolean value

Method `checkModelExist()`**:** Check whether a model has been already stored.

Usage:

```
TrainLayer$checkModelExist()
```

Returns: Boolean value

Method `checkVarSelExist()`**:** Check whether a variable selection tool has been already stored.

Usage:

```
TrainLayer$checkVarSelExist()
```

Returns: Boolean value

Method `checkTrainDataExist()`**:** Check whether a training data has been already stored.

Usage:

```
TrainLayer$checkTrainDataExist()
```

Returns: Boolean value

Method `summary()`**:** Generate summary.

Usage:

```
TrainLayer$summary()
```

See Also

Training, Lrner, TrainData, TestData and Model

`TrainMetaLayer` *TrainMetaLayer Class*

Description

This class implement a meta meta layer. A TrainMetaLayer can only exist as unique element of a Training object.

A layer is structured as followed:

- Lrner: It is set by the user to be trained on the meta training data.
- TrainData: It are modality-specific prediction data, automatically created by the internal cross validation.
- Model: The meta model, result of training the learner on the training data, and therefore, not to be set by the user.
- TestData: The meta new data to be predicted, consisting in predictions obtained from each layer.

A meta layer can train its meta learner on the meta training data and store the resulting meta model. The meta layer can predict values given a new meta layer.

Super class

`fuseMLR::HashTable` -> `TrainMetaLayer`

Methods

Public methods:

- `TrainMetaLayer$new()`
- `TrainMetaLayer$print()`
- `TrainMetaLayer$getTraining()`
- `TrainMetaLayer$getTargetObj()`
- `TrainMetaLayer$train()`

- `TrainMetaLayer$predict()`
- `TrainMetaLayer$impute()`
- `TrainMetaLayer$getTrainData()`
- `TrainMetaLayer$getLrner()`
- `TrainMetaLayer$getModel()`
- `TrainMetaLayer$openAccess()`
- `TrainMetaLayer$closeAccess()`
- `TrainMetaLayer$getAccess()`
- `TrainMetaLayer$setTrainData()`
- `TrainMetaLayer$checkLrnerExist()`
- `TrainMetaLayer$checkModelExist()`
- `TrainMetaLayer$checkTrainDataExist()`
- `TrainMetaLayer$set2NotTrained()`
- `TrainMetaLayer$summary()`

Method `new()`**:** constructor

Usage:

```
TrainMetaLayer$new(id, training)
```

Arguments:

`id` `character`
Id of training meta-layer.

`training` `Training`

Method `print()`**:** Printer

Usage:

```
TrainMetaLayer$print(...)
```

Arguments:

`...` `any`

Method `getTraining()`**:** Getter of the current training object.

Usage:

```
TrainMetaLayer$getTraining()
```

Returns: The current training object is returned.

Method `getTargetObj()`**:** Getter of the target object.

Usage:

```
TrainMetaLayer$getTargetObj()
```

Method `train()`**:** Trains the current layer.

Usage:

```
TrainMetaLayer$train(ind_subset = NULL, verbose = TRUE)
```

Arguments:

`ind_subset` vector
: ID subset of individuals to be used for training.

`verbose` boolean
: Warning messages will be displayed if set to TRUE.

Returns: The current layer is returned with the resulting model.

Method `predict()`**:** Predicts values for the new layer taking as argument.

Usage:

```
TrainMetaLayer$predict(new_layer, ind_subset = NULL)
```

Arguments:

`new_layer` TrainLayer
: A trained TrainLayer object.

`ind_subset` vector
: Index subset.

Returns: A new object with the predicted values is returned.

Method `impute()`**:** Imputes missing values in modality-specific predictions. Only mode and median based imputations are actually supported.

Usage:

```
TrainMetaLayer$impute(impute_fct = NULL, impute_param = NULL)
```

Arguments:

`impute_fct` character
: An imputation function to use instead of median or mode imputation. This parameter is actually not used. This corresponds to median or mode based imputation.

`impute_param` list
: The list of parameters to call the imputation function. Not yet implemented!

Returns: A new object with the predicted values is returned.

Method `getTrainData()`**:** Getter of the training dataset stored on the current layer.

Usage:

```
TrainMetaLayer$getTrainData()
```

Returns: The stored TrainData object is returned.

Method `getLrner()`**:** Getter of the learner.

Usage:

```
TrainMetaLayer$getLrner()
```

Returns: The stored Lrner object is returned.

Method `getModel()`**:** Getter of the model.

Usage:

```
TrainMetaLayer$getModel()
```

Returns: The stored Model object is returned.

Method `openAccess()`: Open access to the meta layer. A meta learner is only modifiable if the access is opened.

Usage:

```
TrainMetaLayer$openAccess()
```

Method `closeAccess()`: Close access to the meta layer to avoid accidental modification.

Usage:

```
TrainMetaLayer$closeAccess()
```

Method `getAccess()`: Getter of the current access to the meta layer.

Usage:

```
TrainMetaLayer$getAccess()
```

Method `setTrainData()`: Create and set an TrainData object to the current meta learner.

Usage:

```
TrainMetaLayer$setTrainData(id, ind_col, data_frame)
```

Arguments:

`id character`
: ID of the TrainData object to be instanciated.

`ind_col character`
: Name of individual column IDs.

`data_frame data.frame`
: `data.frame` of layer specific predictions.

Method `checkLrnerExist()`: Check whether a training data has been already stored.

Usage:

```
TrainMetaLayer$checkLrnerExist()
```

Returns: Boolean value

Method `checkModelExist()`: Check whether a model has been already stored.

Usage:

```
TrainMetaLayer$checkModelExist()
```

Returns: Boolean value

Method `checkTrainDataExist()`: Check whether a training data has been already stored.

Usage:

```
TrainMetaLayer$checkTrainDataExist()
```

Returns: Boolean value

Method `set2NotTrained()`: Only usefull to reset status FALSE after cross validation.

Usage:

```
TrainMetaLayer$set2NotTrained()
```

Method `summary()`: Generate summary.

Usage:

```
TrainMetaLayer$summary()
```

`upsetplot` *upsetplot*

Description

An upset plot of overlapping individuals.

Usage

```
upsetplot(object, ...)
```

Arguments

`object`	`Training or Testing` Training or testing object for each the upset plot will be created.
`...`	`any` Further arguments to be passed to the `upset` function from package `UpSetR`.

`VarSel` *Varsel Class*

Description

This class implements a learner. A VarSel object can only exist as a component of a TrainLayer or a TrainMetaLayer object.

Methods

Public methods:

- `VarSel$new()`
- `VarSel$print()`
- `VarSel$summary()`
- `VarSel$interface()`
- `VarSel$varSelection()`
- `VarSel$getTrainLayer()`
- `VarSel$getId()`
- `VarSel$getPackage()`
- `VarSel$getVarSubSet()`
- `VarSel$getParamInterface()`
- `VarSel$getNaAction()`
- `VarSel$getExtractVar()`

Method `new()`**:** Variable selection parameter list.

Learner ID.

Usage:

```
VarSel$new(
  id,
  package = NULL,
  varsel_fct,
  varsel_param,
  train_layer,
  na_action = "na.rm"
)
```

Arguments:

`id` `character`
: Package that implements the variable selection function. If NULL, the variable selection function is called from the current environment.

`package` `character`
: Variable selection function name. Note: Variable selection functions, except `Boruta`, must return a vector of selected variables.

`varsel_fct` `character`
: Variable selection parameters.

`varsel_param` `list`
: Layer on which the learner is stored.

`train_layer` `TrainLayer`
: The training layer where to store the learner.

`na_action` `character`
: Handling of missing values in meta-data. Set to "na.keep" to keep missing values, "na.rm" to remove individuals with missing values or "na.impute" (only applicable on meta-data) to impute missing values in meta-data. Only median and mode based imputations are actually handled. With the "na.keep" option, ensure that the provided learner can handle missing values. If `TRUE`, the individuals with missing predictor values will be removed from the training dataset.

Method `print()`**:** Printer

Usage:

```
VarSel$print(...)
```

Arguments:

`...` `any`

Method `summary()`**:** Summary

Usage:

```
VarSel$summary(...)
```

Arguments:

`...` `any`

Method `interface()`**:** Learner and prediction parameter interface. Use this function to provide how the following parameters are named in the learning function (`lrn_fct`) you provided when creating the learner, or in the predicting function.

Usage:

```
VarSel$interface(
  x = "x",
  y = "y",
  object = "object",
  data = "data",
  extract_var_fct = NULL
)
```

Arguments:

`x` string
: Name of the argument to pass the matrix of independent variables in the original learning function.

`y` string
: Name of the argument to pass the response variable in the original learning function.

`object` string
: Name of the argument to pass the model in the original predicting function.

`data` character
: Name of the argument to pass new data in the original predicting function.

`extract_var_fct` character or function
: If the variable selection function that is called does not return a vector, then use this argument to specify a (or a name of a) function that can be used to extract vector of selected variables. Default value is NULL, if selected variables are in a vector.

Method `varSelection()`**:** Tains the current learner (from class Lrner) on the current training data (from class TrainData).

Usage:

```
VarSel$varSelection(ind_subset = NULL)
```

Arguments:

`ind_subset` vector
: Individual ID subset on which the training will be performed.

Returns: The resulting model, from class Model, is returned.

Method `getTrainLayer()`**:** The current layer is returned.

Usage:

```
VarSel$getTrainLayer()
```

Returns: TrainLayer object.

Method `getId()`**:** Getter of the current learner ID.

Usage:

```
VarSel$getId()
```

Returns: The current learner ID.

Method `getPackage()`**:** Getter of the variable selection package implementing the variable selection function.

Usage:

```
VarSel$getPackage()
```

Returns: The name of the package implementing the variable selection function.

Method `getVarSubSet()`**:** Getter of the list of selected variables.

Usage:

```
VarSel$getVarSubSet()
```

Returns: List of selected variables..

Method `getParamInterface()`**:** The current parameter interface is returned.

Usage:

```
VarSel$getParamInterface()
```

Returns: A data.frame of interface.

Method `getNaAction()`**:** The current layer is returned.

Usage:

```
VarSel$getNaAction()
```

Method `getExtractVar()`**:** The function to extract selected variables is returned.

Usage:

```
VarSel$getExtractVar()
```

Returns: A data.frame of interface.

`varSelection` *varSelection*

Description

Variable selection on the training object passed as argument.

Usage

```
varSelection(training, ind_subset = NULL)
```

Arguments

`training`	`Training` Training object for storing the created layer.
`ind_subset`	`vector` ID subset of individuals to be used for variable selection.

Value

A `data.frame` with two columns: layer and selected variables.

References

Fouodo C.J.K, Bleskina M. and Szymczak (2024). fuseMLR: An R package for integrative prediction modeling of multi-omics data, paper submitted.

`weightedMeanLearner` *The weighted mean meta-learner*

Description

Modality-specific learner are assessed and weighted based on their predictions. This function is intended to be (internally) used as meta-learner in fuseMLR.

Usage

```
weightedMeanLearner(x, y, weighted = TRUE, perf = NULL, na_rm = FALSE)
```

Arguments

`x`	`data.frame` Modality-specific predictions. Each column of the `data.frame` content the predictions a specific learner.
`y`	`vector` True target values. If classification, either binary or two level factor variable.
`weighted`	`boolean` If TRUE, a weighted sum is computed. As default, weights are estimated based on Brier Score for classification setting and mean squared error for regression. Otherwise, use argument `perf` below to specify the function to use estimate learner performance.
`perf`	`function` Function to compute layer-specific performance of learners. If NULL, the Brier Score (classification) or a mean squared error (regression) is used by default as performance measure. Otherwise, the performance function must accept two parameters: `observed` (observed values) and `predicted` (predicted values).
`na_rm`	`boolean` Should missing values be removed when computing the weights?

Value

Object of class `weightedMeanLearner` with the vector of estimated weights pro layer.

Examples

```
set.seed(20240624L)
x = data.frame(x1 = runif(n = 50L, min = 0, max = 1),
               x2 = runif(n = 50L, min = 0, max = 1))
y = sample(x = 0L:1L, size = 50L, replace = TRUE)
my_model = weightedMeanLearner(x = x, y = y)
```

Index