Package 'less'

October 13, 2022

Title Learning with Subset Stacking

Version 0.1.0

Description ``Learning with Subset Stacking'' is a supervised learning algorithm that is based on training many local estimators on subsets of a given dataset, and then passing their predictions to a global estimator. You can find the details about LESS in our manuscript at <arXiv:2112.06251>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.1

Imports caret, e1071, FNN, MLmetrics, pracma, R6, randomForest, RANN, rpart, wordspace

Depends R (>= 2.10)

LazyData true

NeedsCompilation no

Author Ilker Birbil [aut], Burhan Ozer Cavdar [aut, trl, cre]

Maintainer Burhan Ozer Cavdar <bcavdar17@ku.edu.tr>

Repository CRAN

Date/Publication 2022-09-27 11:00:02 UTC

R topics documented:

`abalone` *Abalone Data Set*

Description

The number of rings is the value to predict.

Usage

```
abalone
```

Format

A dataframe with 4177 rows and 8 variables

Length Longest shell measurement in mm

Diameter perpendicular to length in mm

Height with meat in shell in mm

Whole weight whole abalone in grams

Shucked weight weight of meat in grams

Viscera weight gut weight (after bleeding) in grams

Shell weight after being dried in grams

Rings +1.5 gives the age in years

Source

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository http://archive.ics.uci.edu/ml/. Irvine, CA: University of California, School of Information and Computer Science.

Examples

```
data(abalone)
```

`BaseEstimator` *BaseEstimator*

Description

A dummy base R6 class that provides get_all_fields, get_attributes and set_random_state functionalities for estimators

Value

R6 Class of BaseEstimator

Methods

Public methods:

- `BaseEstimator$get_all_fields()`
- `BaseEstimator$get_attributes()`
- `BaseEstimator$set_random_state()`
- `BaseEstimator$clone()`

Method `get_all_fields()`**:** Auxiliary function returning the name of all private and public fields of the self class

Usage:

```
BaseEstimator$get_all_fields()
```

Examples:

```
TestClass <- R6::R6Class(classname = "TestClass",
inherit = BaseEstimator,
private = list(random_state = NULL))
exampleClass <- TestClass$new()
exampleClass$get_all_fields()
```

Method `get_attributes()`**:** Auxiliary function returning the name and values of all private and public fields of the self class

Usage:

```
BaseEstimator$get_attributes()
```

Examples:

```
exampleClass$get_attributes()
```

Method `set_random_state()`**:** Auxiliary function that sets random state attribute of the self class

Usage:

```
BaseEstimator$set_random_state(random_state)
```

Arguments:

`random_state` seed number to be set as random state

Returns: self

Examples:

```
exampleClass$set_random_state(2022)
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `BaseEstimator$get_all_fields`
## ------------------------------------------------

TestClass <- R6::R6Class(classname = "TestClass",
inherit = BaseEstimator,
private = list(random_state = NULL))
exampleClass <- TestClass$new()
exampleClass$get_all_fields()

## ------------------------------------------------
## Method `BaseEstimator$get_attributes`
## ------------------------------------------------

exampleClass$get_attributes()

## ------------------------------------------------
## Method `BaseEstimator$set_random_state`
## ------------------------------------------------

exampleClass$set_random_state(2022)
```

check_is_fitted *Fitting Control*

Description

Checks if the given estimator is fitted

Usage

```
check_is_fitted(estimator)
```

Arguments

estimator	An estimator with is_fitted attribute

Value

TRUE if the estimator is fitted, FALSE is not

comparison_plot *Comparison Plot*

Description

Plots the fitted functions obtained with various regressors (using their default values) on the one-dimensional dataset (X, y).

Usage

```
comparison_plot(X, y, model_list)
```

Arguments

X	Predictors
y	Response variables
model_list	List of models to be compared

Examples

```
sine_data_list <- less::synthetic_sine_curve()
X_sine <- sine_data_list[[1]]
y_sine <- sine_data_list[[2]]

model_list <- c(DecisionTreeRegressor$new(), LinearRegression$new(), KNeighborsRegressor$new())

comparison_plot(X_sine, y_sine, model_list)
```

CoverTree *CoverTree - Nearest Neighbor Search*

Description

Wrapper R6 Class of FNN::get.knnx function that can be used for LESSRegressor and LESSClassifier

Details

The cover tree is O(n) space data structure which allows us to answer queries in the same O(log(n)) time as kd tree given a fixed intrinsic dimensionality. Templated code from https://hunch.net/~jl/projects/cover_tree/cover_tree.html is used.

Value

R6 Class of CoverTree

Methods

Public methods:

- `CoverTree$new()`
- `CoverTree$query()`
- `CoverTree$clone()`

Method `new()`**:** Creates a new instance of R6 Class of CoverTree

Usage:

```
CoverTree$new(X = NULL)
```

Arguments:

`X` An **M x d** data.frame or matrix, where each of the **M** rows is a point or a (column) vector (where **d=1**).

Examples:

```
data(abalone)
ct <- CoverTree$new(abalone[1:100,])
```

Method `query()`**:** Finds the p number of near neighbours for each point in an input/output dataset.

Usage:

```
CoverTree$query(query_X = private$X, k = 1)
```

Arguments:

`query_X` A set of **N x d** points that will be queried against `X`. **d**, the number of columns, must be the same as `X`. If missing, defaults to `X`.

`k` The maximum number of nearest neighbours to compute (deafults to 1).

Returns: A `list` of length 2 with elements:

`nn.idx` A **N x k** integer matrix returning the near neighbour indices.

`nn.dists` A **N x k** matrix returning the near neighbour Euclidean distances

Examples:

```
res <- ct$query(abalone[1:3,], k=2)
print(res)
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`FNN::get.knnx()`

Examples

```
## ------------------------------------------------
## Method `CoverTree$new`
## ------------------------------------------------

data(abalone)
ct <- CoverTree$new(abalone[1:100,])

## ------------------------------------------------
## Method `CoverTree$query`
## ------------------------------------------------

res <- ct$query(abalone[1:3,], k=2)
print(res)
```

`DecisionTreeClassifier`

DecisionTreeClassifier

Description

Wrapper R6 Class of rpart::rpart function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of DecisionTreeClassifier

Super classes

less::BaseEstimator -> less::SklearnEstimator -> DecisionTreeClassifier

Methods

Public methods:

- DecisionTreeClassifier$new()
- DecisionTreeClassifier$fit()
- DecisionTreeClassifier$predict()
- DecisionTreeClassifier$get_estimator_type()
- DecisionTreeClassifier$clone()

Method `new()`**:** Creates a new instance of R6 Class of DecisionTreeClassifier

Usage:

```
DecisionTreeClassifier$new(
  min_samples_split = 2,
  min_samples_leaf = 1,
  cp = 0.001,
  xval = 10,
  surrogate_style = 0,
  max_depth = 30
)
```

Arguments:

`min_samples_split` The minimum number of observations that must exist in a node in order for a split to be attempted (defaults to 2).

`min_samples_leaf` The minimum number of observations in any terminal (leaf) node (defaults to 1).

`cp` Complexity Parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. This means that the overall R-squared must increase by cp at each step. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile. (defaults to 0.001)

`xval` Number of cross-validations (defaults to 10)

`surrogate_style` Controls the selection of a best surrogate. If set to 0 (default) the program uses the total number of correct classification for a potential surrogate variable, if set to 1 it uses the percent correct, calculated over the non-missing values of the surrogate. The first option more severely penalizes covariates with a large number of missing values.

`max_depth` The maximum depth of any node of the final tree, with the root node counted as depth 0. Values greater than 30 will give nonsense results on 32-bit machines.

Examples:

```
dt <- DecisionTreeClassifier$new()
dt <- DecisionTreeClassifier$new(min_samples_split = 10)
dt <- DecisionTreeClassifier$new(min_samples_leaf = 6, cp = 0.01)
```

Method `fit()`**:** Builds a decision tree regressor from the training set (X, y).

Usage:

```
DecisionTreeClassifier$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes labels

Returns: Fitted R6 Class of DecisionTreeClassifier

Examples:

```
data(iris)
split_list <- train_test_split(iris, test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

dt <- DecisionTreeClassifier$new()
dt$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X0.

Usage:

```
DecisionTreeClassifier$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: Factor of the predict classes.

Examples:

```
dt <- DecisionTreeClassifier$new()
dt$fit(X_train, y_train)
preds <- dt$predict(X_test)

dt <- DecisionTreeClassifier$new()
preds <- dt$fit(X_train, y_train)$predict(X_test)

preds <- DecisionTreeClassifier$new()$fit(X_train, y_train)$predict(X_test)
print(caret::confusionMatrix(data=preds, reference = factor(y_test)))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
DecisionTreeClassifier$get_estimator_type()
```

Examples:

```
dt$get_estimator_type()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

rpart::rpart()

Examples

```
## ------------------------------------------------
## Method `DecisionTreeClassifier$new`
## ------------------------------------------------

dt <- DecisionTreeClassifier$new()
dt <- DecisionTreeClassifier$new(min_samples_split = 10)
dt <- DecisionTreeClassifier$new(min_samples_leaf = 6, cp = 0.01)

## ------------------------------------------------
## Method `DecisionTreeClassifier$fit`
## ------------------------------------------------

data(iris)
split_list <- train_test_split(iris, test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

dt <- DecisionTreeClassifier$new()
dt$fit(X_train, y_train)

## ------------------------------------------------
## Method `DecisionTreeClassifier$predict`
## ------------------------------------------------

dt <- DecisionTreeClassifier$new()
dt$fit(X_train, y_train)
preds <- dt$predict(X_test)

dt <- DecisionTreeClassifier$new()
preds <- dt$fit(X_train, y_train)$predict(X_test)

preds <- DecisionTreeClassifier$new()$fit(X_train, y_train)$predict(X_test)
print(caret::confusionMatrix(data=preds, reference = factor(y_test)))

## ------------------------------------------------
## Method `DecisionTreeClassifier$get_estimator_type`
## ------------------------------------------------

dt$get_estimator_type()
```

DecisionTreeRegressor *DecisionTreeRegressor*

Description

Wrapper R6 Class of rpart::rpart function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of DecisionTreeRegressor

Super classes

`less::BaseEstimator` -> `less::SklearnEstimator` -> `DecisionTreeRegressor`

Methods

Public methods:

- `DecisionTreeRegressor$new()`
- `DecisionTreeRegressor$fit()`
- `DecisionTreeRegressor$predict()`
- `DecisionTreeRegressor$get_estimator_type()`
- `DecisionTreeRegressor$clone()`

Method `new()`**:** Creates a new instance of R6 Class of DecisionTreeRegressor

Usage:

```
DecisionTreeRegressor$new(
  min_samples_split = 2,
  min_samples_leaf = 1,
  cp = 0.001,
  xval = 10,
  surrogate_style = 0,
  max_depth = 30
)
```

Arguments:

`min_samples_split` The minimum number of observations that must exist in a node in order for a split to be attempted (defaults to 2).

`min_samples_leaf` The minimum number of observations in any terminal (leaf) node (defaults to 1).

`cp` Complexity Parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. This means that the overall R-squared must increase by cp at each step. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile. (defaults to 0.001)

`xval` Number of cross-validations (defaults to 10)

`surrogate_style` Controls the selection of a best surrogate. If set to 0 (default) the program uses the total number of correct classification for a potential surrogate variable, if set to 1 it uses the percent correct, calculated over the non-missing values of the surrogate. The first option more severely penalizes covariates with a large number of missing values.

`max_depth` The maximum depth of any node of the final tree, with the root node counted as depth 0. Values greater than 30 will give nonsense results on 32-bit machines.

Examples:

```
dt <- DecisionTreeRegressor$new()
dt <- DecisionTreeRegressor$new(min_samples_split = 10)
dt <- DecisionTreeRegressor$new(min_samples_leaf = 6, cp = 0.01)
```

Method `fit()`**:** Builds a decision tree regressor from the training set (X, y).

Usage:

```
DecisionTreeRegressor$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes response variables

Returns: Fitted R6 Class of DecisionTreeRegressor

Examples:

```
data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

dt <- DecisionTreeRegressor$new()
dt$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X0.

Usage:

```
DecisionTreeRegressor$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: The predict values.

Examples:

```
dt <- DecisionTreeRegressor$new()
dt$fit(X_train, y_train)
preds <- dt$predict(X_test)

dt <- DecisionTreeRegressor$new()
preds <- dt$fit(X_train, y_train)$predict(X_test)

preds <- DecisionTreeRegressor$new()$fit(X_train, y_train)$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
DecisionTreeRegressor$get_estimator_type()
```

Examples:

```
dt$get_estimator_type()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

`rpart::rpart()`

Examples

```
## ------------------------------------------------
## Method `DecisionTreeRegressor$new`
## ------------------------------------------------

dt <- DecisionTreeRegressor$new()
dt <- DecisionTreeRegressor$new(min_samples_split = 10)
dt <- DecisionTreeRegressor$new(min_samples_leaf = 6, cp = 0.01)

## ------------------------------------------------
## Method `DecisionTreeRegressor$fit`
## ------------------------------------------------

data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

dt <- DecisionTreeRegressor$new()
dt$fit(X_train, y_train)

## ------------------------------------------------
## Method `DecisionTreeRegressor$predict`
## ------------------------------------------------

dt <- DecisionTreeRegressor$new()
dt$fit(X_train, y_train)
preds <- dt$predict(X_test)

dt <- DecisionTreeRegressor$new()
preds <- dt$fit(X_train, y_train)$predict(X_test)

preds <- DecisionTreeRegressor$new()$fit(X_train, y_train)$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))
```

```
## ------------------------------------------------
## Method `DecisionTreeRegressor$get_estimator_type`
## ------------------------------------------------

dt$get_estimator_type()
```

`get_functions` *Get Functions*

Description

Prints the available regressors, clustering methods, tree functions and helper functions within LESS package.

Usage

```
get_functions()
```

Examples

```
get_functions()
```

`HierarchicalClustering` *Hierarchical Clustering*

Description

Wrapper R6 Class of stats::hclust function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of HierarchicalClustering

Super class

`less::BaseEstimator` -> `HierarchicalClustering`

Methods

Public methods:

- HierarchicalClustering$new()
- HierarchicalClustering$fit()
- HierarchicalClustering$get_cluster_centers()
- HierarchicalClustering$get_labels()
- HierarchicalClustering$clone()

Method `new()`**:** Creates a new instance of R6 Class of HierarchicalClustering

Usage:

```
HierarchicalClustering$new(linkage = "ward.D2", n_clusters = 8)
```

Arguments:

`linkage` the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC) (defaults to ward.D2).

`n_clusters` the number of clusters (defaults to 8).

Examples:

```
hc <- HierarchicalClustering$new()
hc <- HierarchicalClustering$new(n_clusters = 10)
hc <- HierarchicalClustering$new(n_clusters = 10, linkage = "complete")
```

Method `fit()`**:** Perform hierarchical clustering on a data matrix.

Usage:

```
HierarchicalClustering$fit(X)
```

Arguments:

`X` numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).

Returns: Fitted R6 class of HierarchicalClustering() that has 'labels' attribute

Examples:

```
data(abalone)
hc <- HierarchicalClustering$new()
hc$fit(abalone[1:100,])
```

Method `get_cluster_centers()`**:** Auxiliary function returning the cluster centers

Usage:

```
HierarchicalClustering$get_cluster_centers()
```

Examples:

```
print(hc$get_cluster_centers())
```

Method `get_labels()`**:** Auxiliary function returning a vector of integers (from 1:k) indicating the cluster to which each point is allocated.

Usage:

```
HierarchicalClustering$get_labels()
```

Examples:

```
print(hc$get_labels())
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

stats::hclust()

Examples

```
## ------------------------------------------------
## Method `HierarchicalClustering$new`
## ------------------------------------------------

hc <- HierarchicalClustering$new()
hc <- HierarchicalClustering$new(n_clusters = 10)
hc <- HierarchicalClustering$new(n_clusters = 10, linkage = "complete")

## ------------------------------------------------
## Method `HierarchicalClustering$fit`
## ------------------------------------------------

data(abalone)
hc <- HierarchicalClustering$new()
hc$fit(abalone[1:100,])

## ------------------------------------------------
## Method `HierarchicalClustering$get_cluster_centers`
## ------------------------------------------------

print(hc$get_cluster_centers())

## ------------------------------------------------
## Method `HierarchicalClustering$get_labels`
## ------------------------------------------------

print(hc$get_labels())
```

KDTree *KDTree - Nearest Neighbor Search*

Description

Wrapper R6 Class of RANN::nn2 function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of KDTree

Methods

Public methods:

- `KDTree$new()`
- `KDTree$query()`
- `KDTree$clone()`

Method `new()`**:** Creates a new instance of R6 Class of KDTree

Usage:

```
KDTree$new(X = NULL)
```

Arguments:

`X` An **M x d** data.frame or matrix, where each of the **M** rows is a point or a (column) vector (where **d=1**).

Examples:

```
data(abalone)
kdt <- KDTree$new(abalone[1:100,])
```

Method `query()`**:** Finds the p number of near neighbours for each point in an input/output dataset. The advantage of the kd-tree is that it runs in O(M log M) time.

Usage:

```
KDTree$query(query_X = private$X, k = 1)
```

Arguments:

`query_X` A set of **N x d** points that will be queried against `X`. **d**, the number of columns, must be the same as `X`. If missing, defaults to `X`.

`k` The maximum number of nearest neighbours to compute (deafults to 1).

Returns: A `list` of length 2 with elements:

`nn.idx` A **N x k** integer matrix returning the near neighbour indices.

`nn.dists` A **N x k** matrix returning the near neighbour Euclidean distances

Examples:

```
res <- kdt$query(abalone[1:3,], k=2)
print(res)
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`RANN::nn2()`

Examples

```
## ------------------------------------------------
## Method `KDTree$new`
## ------------------------------------------------

data(abalone)
kdt <- KDTree$new(abalone[1:100,])

## ------------------------------------------------
## Method `KDTree$query`
## ------------------------------------------------

res <- kdt$query(abalone[1:3,], k=2)
print(res)
```

`KMeans` *KMeans Clustering*

Description

Wrapper R6 Class of stats::kmeans function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of KMeans

Super class

`less::BaseEstimator` -> `KMeans`

Methods

Public methods:

- KMeans$new()
- KMeans$fit()
- KMeans$get_cluster_centers()
- KMeans$get_labels()
- KMeans$clone()

Method `new()`**:** Creates a new instance of R6 Class of KMeans

Usage:

```
KMeans$new(n_clusters = 8, n_init = 10, max_iter = 300, random_state = NULL)
```

Arguments:

`n_clusters` the number of clusters. A random set of (distinct) rows in X is chosen as the initial centres (default to 8)

`n_init` how many random sets should be chosen? (default to 10)

`max_iter` the maximum number of iterations allowed (default to 300).

`random_state` seed number to be used for fixing the randomness (default to NULL).

Examples:

```
km <- KMeans$new()
km <- KMeans$new(n_clusters = 10)
km <- KMeans$new(n_clusters = 10, random_state = 100)
```

Method `fit()`**:** Perform k-means clustering on a data matrix.

Usage:

```
KMeans$fit(X)
```

Arguments:

`X` numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).

Returns: Fitted R6 class of KMeans() that has 'cluster_centers' and 'labels' attributes

Examples:

```
data(abalone)
km <- KMeans$new()
km$fit(abalone[1:100,])
```

Method `get_cluster_centers()`**:** Auxiliary function returning the cluster centers

Usage:

```
KMeans$get_cluster_centers()
```

Examples:

```
print(km$get_cluster_centers())
```

Method `get_labels()`**:** Auxiliary function returning a vector of integers (from 1:k) indicating the cluster to which each point is allocated.

Usage:

```
KMeans$get_labels()
```

Examples:

```
print(km$get_labels())
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

`stats::kmeans()`

Examples

```
## ------------------------------------------------
## Method `KMeans$new`
## ------------------------------------------------

km <- KMeans$new()
km <- KMeans$new(n_clusters = 10)
km <- KMeans$new(n_clusters = 10, random_state = 100)

## ------------------------------------------------
## Method `KMeans$fit`
## ------------------------------------------------

data(abalone)
km <- KMeans$new()
km$fit(abalone[1:100,])

## ------------------------------------------------
## Method `KMeans$get_cluster_centers`
## ------------------------------------------------

print(km$get_cluster_centers())

## ------------------------------------------------
## Method `KMeans$get_labels`
## ------------------------------------------------

print(km$get_labels())
```

KNeighborsClassifier *KNeighborsClassifier*

Description

Wrapper R6 Class of caret::knnreg function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of KNeighborsClassifier

Super classes

less::BaseEstimator -> less::SklearnEstimator -> KNeighborsClassifier

Methods

Public methods:

- KNeighborsClassifier$new()
- KNeighborsClassifier$fit()
- KNeighborsClassifier$predict()
- KNeighborsClassifier$get_estimator_type()
- KNeighborsClassifier$clone()

Method `new()`**:** Creates a new instance of R6 Class of KNeighborsClassifier

Usage:

```
KNeighborsClassifier$new(k = 5)
```

Arguments:

`k` Number of neighbors considered (defaults to 5).

Examples:

```
knc <- KNeighborsClassifier$new()
knc <- KNeighborsClassifier$new(k = 5)
```

Method `fit()`**:** Fit the k-nearest neighbors regressor from the training set (X, y).

Usage:

```
KNeighborsClassifier$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes labels

Returns: Fitted R6 Class of KNeighborsClassifier

Examples:

```
data(iris)
split_list <- train_test_split(iris, test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

knc <- KNeighborsClassifier$new()
knc$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X0.

Usage:

```
KNeighborsClassifier$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: Factor of the predict classes.

Examples:

```
knc <- KNeighborsClassifier$new()
knc$fit(X_train, y_train)
preds <- knc$predict(X_test)

knc <- KNeighborsClassifier$new()
preds <- knc$fit(X_train, y_train)$predict(X_test)

preds <- KNeighborsClassifier$new()$fit(X_train, y_train)$predict(X_test)
print(caret::confusionMatrix(data=factor(preds), reference = factor(y_test)))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
KNeighborsClassifier$get_estimator_type()
```

Examples:

```
knc$get_estimator_type()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`caret::knn3()`

Examples

```
## ------------------------------------------------
## Method `KNeighborsClassifier$new`
## ------------------------------------------------

knc <- KNeighborsClassifier$new()
knc <- KNeighborsClassifier$new(k = 5)

## ------------------------------------------------
## Method `KNeighborsClassifier$fit`
## ------------------------------------------------

data(iris)
split_list <- train_test_split(iris, test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

knc <- KNeighborsClassifier$new()
knc$fit(X_train, y_train)

## ------------------------------------------------
## Method `KNeighborsClassifier$predict`
## ------------------------------------------------

knc <- KNeighborsClassifier$new()
knc$fit(X_train, y_train)
preds <- knc$predict(X_test)

knc <- KNeighborsClassifier$new()
preds <- knc$fit(X_train, y_train)$predict(X_test)

preds <- KNeighborsClassifier$new()$fit(X_train, y_train)$predict(X_test)
print(caret::confusionMatrix(data=factor(preds), reference = factor(y_test)))

## ------------------------------------------------
## Method `KNeighborsClassifier$get_estimator_type`
## ------------------------------------------------

knc$get_estimator_type()
```

KNeighborsRegressor *KNeighborsRegressor*

Description

Wrapper R6 Class of caret::knnreg function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of KNeighborsRegressor

Super classes

`less::BaseEstimator` -> `less::SklearnEstimator` -> `KNeighborsRegressor`

Methods

Public methods:

- `KNeighborsRegressor$new()`
- `KNeighborsRegressor$fit()`
- `KNeighborsRegressor$predict()`
- `KNeighborsRegressor$get_estimator_type()`
- `KNeighborsRegressor$clone()`

Method `new()`**:** Creates a new instance of R6 Class of KNeighborsRegressor

Usage:

```
KNeighborsRegressor$new(k = 5)
```

Arguments:

`k` Number of neighbors considered (defaults to 5).

Examples:

```
knr <- KNeighborsRegressor$new()
knr <- KNeighborsRegressor$new(k = 5)
```

Method `fit()`**:** Fit the k-nearest neighbors regressor from the training set (X, y).

Usage:

```
KNeighborsRegressor$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes response variables

Returns: Fitted R6 Class of KNeighborsRegressor

Examples:

```
data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

knr <- KNeighborsRegressor$new()
knr$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X0.

Usage:

```
KNeighborsRegressor$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: The predict values.

Examples:

```
knr <- KNeighborsRegressor$new()
knr$fit(X_train, y_train)
preds <- knr$predict(X_test)

knr <- KNeighborsRegressor$new()
preds <- knr$fit(X_train, y_train)$predict(X_test)

preds <- KNeighborsRegressor$new()$fit(X_train, y_train)$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
KNeighborsRegressor$get_estimator_type()
```

Examples:

```
knr$get_estimator_type()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`caret::knnreg()`

Examples

```
## ------------------------------------------------
## Method `KNeighborsRegressor$new`
## ------------------------------------------------

knr <- KNeighborsRegressor$new()
knr <- KNeighborsRegressor$new(k = 5)

## ------------------------------------------------
## Method `KNeighborsRegressor$fit`
## ------------------------------------------------
```

```
data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

knr <- KNeighborsRegressor$new()
knr$fit(X_train, y_train)

## ------------------------------------------------
## Method `KNeighborsRegressor$predict`
## ------------------------------------------------

knr <- KNeighborsRegressor$new()
knr$fit(X_train, y_train)
preds <- knr$predict(X_test)

knr <- KNeighborsRegressor$new()
preds <- knr$fit(X_train, y_train)$predict(X_test)

preds <- KNeighborsRegressor$new()$fit(X_train, y_train)$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))

## ------------------------------------------------
## Method `KNeighborsRegressor$get_estimator_type`
## ------------------------------------------------

knr$get_estimator_type()
```

---

k_fold_cv *k-Fold Cross Validation*

Description

Applies k-Fold cross validation to the given model on the given data

Usage

```
k_fold_cv(
  data = NULL,
  model = NULL,
  random_state = NULL,
  k = 5,
  y_index = ncol(data)
)
```

Arguments

data	The dataset to be used
model	A classification or a regression model (from LESS package)
random_state	A seed number to get reproducable result
k	Number of splits on the training set (defaults to 5)
y_index	Column index of the response variable on the given **data**. Default is the last column.

Value

A vector consists of metric of the individual folds and the average metric over the folds

Examples

```
k_fold_cv(data = iris, model = KNeighborsClassifier$new(), k = 3)
```

---

laplacian *Laplacian kernel - L1 norm*

Description

An alternative distance function that can be used in LESS.

Usage

```
laplacian(data, center, coeff = 0.01)
```

Arguments

data	Data that includes points in shape of **(M x d)**
center	A constant point in shape of **(1 x d)**
coeff	Coefficient value for Laplacian kernel

Value

A numeric vector containing the laplacian kernel distance between each point in **data** and **center**.

Examples

```
data <- matrix(1:12, nrow=3)
center <- c(2, 7, 1, 3)
distances <- laplacian(data, center)
print(distances)
```

LESSBase *LESSBase*

Description

The base class for LESSRegressor and LESSClassifier

Value

R6 class of LESSBase

Super classes

`less::BaseEstimator` -> `less::SklearnEstimator` -> `LESSBase`

Methods

Public methods:

- `LESSBase$new()`
- `LESSBase$set_random_state()`
- `LESSBase$get_n_subsets()`
- `LESSBase$get_n_neighbors()`
- `LESSBase$get_frac()`
- `LESSBase$get_n_replications()`
- `LESSBase$get_d_normalize()`
- `LESSBase$get_scaling()`
- `LESSBase$get_val_size()`
- `LESSBase$get_random_state()`
- `LESSBase$get_isFitted()`
- `LESSBase$get_replications()`
- `LESSBase$clone()`

Method `new()`**:** Creates a new instance of R6 Class of LESSBase

Usage:

```
LESSBase$new(replications = NULL, scobject = NULL, isFitted = FALSE)
```

Arguments:

`replications` List to store the replications

`scobject` Scaling object used for normalization (less::StandardScaler)

`isFitted` Flag to check whether LESS is fitted

Method `set_random_state()`**:** Auxiliary function that sets random state attribute of the self class

Usage:

```
LESSBase$set_random_state(random_state)
```

Arguments:
`random_state` seed number to be set as random state
Returns: self

Method `get_n_subsets()`**:** Auxiliary function returning the number of subsets
Usage:
```
LESSBase$get_n_subsets()
```

Method `get_n_neighbors()`**:** Auxiliary function returning the number of neighbors
Usage:
```
LESSBase$get_n_neighbors()
```

Method `get_frac()`**:** Auxiliary function returning the percentage of samples used to set the number of neighbors
Usage:
```
LESSBase$get_frac()
```

Method `get_n_replications()`**:** Auxiliary function returning the number of replications
Usage:
```
LESSBase$get_n_replications()
```

Method `get_d_normalize()`**:** Auxiliary function returning the flag for normalization
Usage:
```
LESSBase$get_d_normalize()
```

Method `get_scaling()`**:** Auxiliary function returning the flag for scaling
Usage:
```
LESSBase$get_scaling()
```

Method `get_val_size()`**:** Auxiliary function returning the validation set size
Usage:
```
LESSBase$get_val_size()
```

Method `get_random_state()`**:** Auxiliary function returning the random seed
Usage:
```
LESSBase$get_random_state()
```

Method `get_isFitted()`**:** Auxiliary function returning the isFitted flag
Usage:
```
LESSBase$get_isFitted()
```

Method `get_replications()`**:** Auxiliary function returning the isFitted flag
Usage:
```
LESSBase$get_replications()
```

Method `clone()`**:** The objects of this class are cloneable with this method.
Usage:
```
LESSBase$clone(deep = FALSE)
```
Arguments:
`deep` Whether to make a deep clone.

LESSBinaryClassifier *LESSBinaryClassifier*

Description

Auxiliary binary classifier for Learning with Subset Stacking (LESS)

Value

R6 class of LESSBinaryClassifier

Super classes

less::BaseEstimator -> less::SklearnEstimator -> less::LESSBase -> LESSBinaryClassifier

Methods

Public methods:

- LESSBinaryClassifier$new()
- LESSBinaryClassifier$fit()
- LESSBinaryClassifier$predict_proba()
- LESSBinaryClassifier$get_global_estimator()
- LESSBinaryClassifier$set_random_state()
- LESSBinaryClassifier$clone()

Method `new()`**:** Creates a new instance of R6 Class of LESSBinaryClassifier

Usage:

```
LESSBinaryClassifier$new(
  frac = NULL,
  n_neighbors = NULL,
  n_subsets = NULL,
  n_replications = 20,
  d_normalize = TRUE,
  val_size = NULL,
  random_state = NULL,
  tree_method = function(X) KDTree$new(X),
  cluster_method = NULL,
  local_estimator = LinearRegression$new(),
  global_estimator = DecisionTreeClassifier$new(),
  distance_function = NULL,
  scaling = TRUE,
  warnings = TRUE
)
```

Arguments:

`frac` fraction of total samples used for the number of neighbors (default is 0.05)

`n_neighbors` number of neighbors (default is NULL)

`n_subsets` number of subsets (default is NULL)

`n_replications` number of replications (default is 20)

`d_normalize` distance normalization (default is TRUE)

`val_size` percentage of samples used for validation (default is NULL - no validation)

`random_state` initialization of the random seed (default is NULL)

`tree_method` method used for constructing the nearest neighbor tree, e.g., less::KDTree (default)

`cluster_method` method used for clustering the subsets, e.g., less::KMeans (default is NULL)

`local_estimator` estimator for the local models (default is less::LinearRegression)

`global_estimator` estimator for the global model (default is less::DecisionTreeRegressor)

`distance_function` distance function evaluating the distance from a subset to a sample, e.g., df(subset, sample) which returns a vector of distances (default is RBF(subset, sample, 1.0/n_subsets^2))

`scaling` flag to normalize the input data (default is TRUE)

`warnings` flag to turn on (TRUE) or off (FALSE) the warnings (default is TRUE)

Method `fit()`**:** Dummy fit function that calls the proper method according to validation and clustering parameters Options are:

- Default fitting (no validation set, no clustering)
- Fitting with validation set (no clustering)
- Fitting with clustering (no) validation set)
- Fitting with validation set and clustering

Usage:

```
LESSBinaryClassifier$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes response variables

Returns: Fitted R6 Class of LESSBinaryClassifier

Method `predict_proba()`**:** Prediction probabilities are evaluated for the test samples in X0

Usage:

```
LESSBinaryClassifier$predict_proba(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Method `get_global_estimator()`**:** Auxiliary function returning the global_estimator

Usage:

```
LESSBinaryClassifier$get_global_estimator()
```

Method `set_random_state()`**:** Auxiliary function that sets random state attribute of the self class

Usage:

```
LESSBinaryClassifier$set_random_state(random_state)
```

Arguments:

`random_state` seed number to be set as random state

Returns: self

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`LESSClassifier` *LESSClassifier*

Description

Classifier for Learning with Subset Stacking (LESS)

Value

R6 class of LESSClassifier

Super classes

`less::BaseEstimator` -> `less::SklearnEstimator` -> `less::LESSBase` -> `LESSClassifier`

Methods

Public methods:

- `LESSClassifier$new()`
- `LESSClassifier$fit()`
- `LESSClassifier$predict()`
- `LESSClassifier$get_estimator_type()`
- `LESSClassifier$set_random_state()`
- `LESSClassifier$clone()`

Method `new()`**:** Creates a new instance of R6 Class of LESSClassifier

Usage:

```
LESSClassifier$new(
  frac = NULL,
  n_neighbors = NULL,
  n_subsets = NULL,
  n_replications = 20,
  d_normalize = TRUE,
```

```
  val_size = NULL,
  random_state = NULL,
  tree_method = function(X) KDTree$new(X),
  cluster_method = NULL,
  local_estimator = LinearRegression$new(),
  global_estimator = DecisionTreeClassifier$new(),
  distance_function = NULL,
  scaling = TRUE,
  warnings = TRUE,
  multiclass = "ovr"
)
```

Arguments:

`frac` fraction of total samples used for the number of neighbors (default is 0.05)

`n_neighbors` number of neighbors (default is NULL)

`n_subsets` number of subsets (default is NULL)

`n_replications` number of replications (default is 20)

`d_normalize` distance normalization (default is TRUE)

`val_size` percentage of samples used for validation (default is NULL - no validation)

`random_state` initialization of the random seed (default is NULL)

`tree_method` method used for constructing the nearest neighbor tree, e.g., less::KDTree (default)

`cluster_method` method used for clustering the subsets, e.g., less::KMeans (default is NULL)

`local_estimator` estimator for the local models (default is less::LinearRegression)

`global_estimator` estimator for the global model (default is less::DecisionTreeRegressor)

`distance_function` distance function evaluating the distance from a subset to a sample, e.g., df(subset, sample) which returns a vector of distances (default is RBF(subset, sample, 1.0/n_subsets^2))

`scaling` flag to normalize the input data (default is TRUE)

`warnings` flag to turn on (TRUE) or off (FALSE) the warnings (default is TRUE)

`multiclass` available strategies are 'ovr' (one-vs-rest, default), 'ovo' (one-vs-one), 'occ' (output-code-classifier) (default is 'ovr')

Examples:

```
lessclassifier <- LESSClassifier$new()
lessclassifier <- LESSClassifier$new(multiclass = "ovo")
```

Method `fit()`**:** Dummy fit function that calls the fit method of the multiclass strategy

Usage:

```
LESSClassifier$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes response variables

Returns: Fitted R6 Class of LESSClassifier

Examples:

```
data(iris)
set.seed(2022)
shuffled_iris <- iris[sample(1:nrow(iris)),]
split_list <- train_test_split(shuffled_iris[1:10,], test_size =  0.3, random_state = 1)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

lessclassifier <- LESSClassifier$new()
lessclassifier$fit(X_train, y_train)
```

Method `predict()`**:** Dummy predict function that calls the predict method of the multiclass strategy

Usage:

```
LESSClassifier$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: Predicted values of the given predictors

Examples:

```
preds <- lessclassifier$predict(X_test)
print(caret::confusionMatrix(data=factor(preds), reference = factor(y_test)))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
LESSClassifier$get_estimator_type()
```

Examples:

```
lessclassifier$get_estimator_type()
```

Method `set_random_state()`**:** Auxiliary function that sets random state attribute of the self class

Usage:

```
LESSClassifier$set_random_state(random_state)
```

Arguments:

`random_state` seed number to be set as random state

Returns: self

Examples:

```
lessclassifier$set_random_state(2022)
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `LESSClassifier$new`
## ------------------------------------------------

lessclassifier <- LESSClassifier$new()
lessclassifier <- LESSClassifier$new(multiclass = "ovo")

## ------------------------------------------------
## Method `LESSClassifier$fit`
## ------------------------------------------------

data(iris)
set.seed(2022)
shuffled_iris <- iris[sample(1:nrow(iris)),]
split_list <- train_test_split(shuffled_iris[1:10,], test_size =  0.3, random_state = 1)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

lessclassifier <- LESSClassifier$new()
lessclassifier$fit(X_train, y_train)

## ------------------------------------------------
## Method `LESSClassifier$predict`
## ------------------------------------------------

preds <- lessclassifier$predict(X_test)
print(caret::confusionMatrix(data=factor(preds), reference = factor(y_test)))

## ------------------------------------------------
## Method `LESSClassifier$get_estimator_type`
## ------------------------------------------------

lessclassifier$get_estimator_type()

## ------------------------------------------------
## Method `LESSClassifier$set_random_state`
## ------------------------------------------------

lessclassifier$set_random_state(2022)
```

LESSRegressor *LESSRegressor*

Description

Regressor for Learning with Subset Stacking (LESS)

Value

R6 class of LESSRegressor

Super classes

less::BaseEstimator -> less::SklearnEstimator -> less::LESSBase -> LESSRegressor

Methods

Public methods:

- LESSRegressor$new()
- LESSRegressor$fit()
- LESSRegressor$predict()
- LESSRegressor$get_estimator_type()
- LESSRegressor$clone()

Method `new()`**:** Creates a new instance of R6 Class of LESSRegressor

Usage:

```
LESSRegressor$new(
  frac = NULL,
  n_neighbors = NULL,
  n_subsets = NULL,
  n_replications = 20,
  d_normalize = TRUE,
  val_size = NULL,
  random_state = NULL,
  tree_method = function(X) KDTree$new(X),
  cluster_method = NULL,
  local_estimator = LinearRegression$new(),
  global_estimator = DecisionTreeRegressor$new(),
  distance_function = NULL,
  scaling = TRUE,
  warnings = TRUE
)
```

Arguments:

`frac` fraction of total samples used for the number of neighbors (default is 0.05)

`n_neighbors` number of neighbors (default is NULL)

`n_subsets` number of subsets (default is NULL)

`n_replications` number of replications (default is 20)

`d_normalize` distance normalization (default is TRUE)

`val_size` percentage of samples used for validation (default is NULL - no validation)

`random_state` initialization of the random seed (default is NULL)

`tree_method` method used for constructing the nearest neighbor tree, e.g., less::KDTree (default)

`cluster_method` method used for clustering the subsets, e.g., less::KMeans (default is NULL)

`local_estimator` estimator for the local models (default is less::LinearRegression)

`global_estimator` estimator for the global model (default is less::DecisionTreeRegressor)

`distance_function` distance function evaluating the distance from a subset to a sample, e.g., df(subset, sample) which returns a vector of distances (default is RBF(subset, sample, 1.0/n_subsets^2))

`scaling` flag to normalize the input data (default is TRUE)

`warnings` flag to turn on (TRUE) or off (FALSE) the warnings (default is TRUE)

Examples:

```
lessRegressor <- LESSRegressor$new()
lessRegressor <- LESSRegressor$new(val_size = 0.3)
lessRegressor <- LESSRegressor$new(cluster_method = less::KMeans$new())
lessRegressor <- LESSRegressor$new(val_size = 0.3, cluster_method = less::KMeans$new())
```

Method `fit()`**:** Dummy fit function that calls the proper method according to validation and clustering parameters Options are:

- Default fitting (no validation set, no clustering)
- Fitting with validation set (no clustering)
- Fitting with clustering (no) validation set)
- Fitting with validation set and clustering

Usage:

```
LESSRegressor$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes response variables

Returns: Fitted R6 Class of LESSRegressor

Examples:

```
data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

lessRegressor <- LESSRegressor$new()
lessRegressor$fit(X_train, y_train)
```

Method `predict()`**:** Predictions are evaluated for the test samples in X0

Usage:

```
LESSRegressor$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: Predicted values of the given predictors

Examples:

```
preds <- lessRegressor$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
LESSRegressor$get_estimator_type()
```

Examples:

```
lessRegressor$get_estimator_type()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

LESSBase

Examples

```
## ------------------------------------------------
## Method `LESSRegressor$new`
## ------------------------------------------------

lessRegressor <- LESSRegressor$new()
lessRegressor <- LESSRegressor$new(val_size = 0.3)
lessRegressor <- LESSRegressor$new(cluster_method = less::KMeans$new())
lessRegressor <- LESSRegressor$new(val_size = 0.3, cluster_method = less::KMeans$new())

## ------------------------------------------------
## Method `LESSRegressor$fit`
## ------------------------------------------------

data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

lessRegressor <- LESSRegressor$new()
lessRegressor$fit(X_train, y_train)

## ------------------------------------------------
## Method `LESSRegressor$predict`
## ------------------------------------------------
```

```
preds <- lessRegressor$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))

## ------------------------------------------------
## Method `LESSRegressor$get_estimator_type`
## ------------------------------------------------

lessRegressor$get_estimator_type()
```

`LinearRegression` *LinearRegression*

Description

Wrapper R6 Class of stats::lm function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of LinearRegression

Super classes

`less::BaseEstimator` -> `less::SklearnEstimator` -> `LinearRegression`

Methods

Public methods:

- `LinearRegression$fit()`
- `LinearRegression$predict()`
- `LinearRegression$get_estimator_type()`
- `LinearRegression$clone()`

Method `fit()`**:** Fits a linear model (y ~ X)

Usage:

```
LinearRegression$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes response variables

Returns: Fitted R6 Class of LinearRegression

Examples:

```
data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
```

```
y_test <- split_list[[4]]

lr <- LinearRegression$new()
lr$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X.

Usage:

```
LinearRegression$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: The predict values.

Examples:

```
lr <- LinearRegression$new()
lr$fit(X_train, y_train)
preds <- lr$predict(X_test)

lr <- LinearRegression$new()
preds <- lr$fit(X_train, y_train)$predict(X_test)

preds <- LinearRegression$new()$fit(X_train, y_train)$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
LinearRegression$get_estimator_type()
```

Examples:

```
lr$get_estimator_type()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`stats::lm()`

Examples

```
## ------------------------------------------------
## Method `LinearRegression$fit`
## ------------------------------------------------
```

```
data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

lr <- LinearRegression$new()
lr$fit(X_train, y_train)

## ------------------------------------------------
## Method `LinearRegression$predict`
## ------------------------------------------------

lr <- LinearRegression$new()
lr$fit(X_train, y_train)
preds <- lr$predict(X_test)

lr <- LinearRegression$new()
preds <- lr$fit(X_train, y_train)$predict(X_test)

preds <- LinearRegression$new()$fit(X_train, y_train)$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))

## ------------------------------------------------
## Method `LinearRegression$get_estimator_type`
## ------------------------------------------------

lr$get_estimator_type()
```

`prepareDataset` *Prepare a Dataset*

Description

Takes X and y datasets and merges them into a dataframe with column names (y, X_1, X_2 ...)

Usage

```
prepareDataset(X, y)
```

Arguments

`X`	Independent variables
`y`	Response variables

Value

A named dataframe which consists of X and y combined

Examples

```
X <- matrix(1:20, nrow = 4)
y <- c(5:8)
prepareDataset(X, y)
```

`prepareXset` *Prepare a Dataset*

Description

Takes X dataset and convert it into a dataframe with column names (X_1, X_2 ...)

Usage

```
prepareXset(X)
```

Arguments

`X`	Independent variables

Value

A named dataframe which consists of X

Examples

```
X <- matrix(1:20, nrow = 4)
prepareXset(X)
```

`RandomForestClassifier`
RandomForestClassifier

Description

Wrapper R6 Class of randomForest::randomForest function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of RandomForestClassifier

Super classes

`less::BaseEstimator` -> `less::SklearnEstimator` -> `RandomForestClassifier`

Methods

Public methods:

- RandomForestClassifier$new()
- RandomForestClassifier$fit()
- RandomForestClassifier$predict()
- RandomForestClassifier$get_estimator_type()
- RandomForestClassifier$clone()

Method `new()`**:** Creates a new instance of R6 Class of RandomForestClassifier

Usage:

```
RandomForestClassifier$new(
  n_estimators = 100,
  random_state = NULL,
  min_samples_leaf = 1,
  max_leaf_nodes = NULL
)
```

Arguments:

`n_estimators` Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times (defaults to 100).

`random_state` Seed number to be used for fixing the randomness (default to NULL).

`min_samples_leaf` Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time) (defaults to 1)

`max_leaf_nodes` Maximum number of terminal nodes trees in the forest can have. If not given, trees are grown to the maximum possible (subject to limits by nodesize). If set larger than maximum possible, a warning is issued. (defaults to NULL)

Examples:

```
rf <- RandomForestClassifier$new()
rf <- RandomForestClassifier$new(n_estimators = 500)
rf <- RandomForestClassifier$new(n_estimators = 500, random_state = 100)
```

Method `fit()`**:** Builds a random forest regressor from the training set (X, y).

Usage:

```
RandomForestClassifier$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes labels

Returns: Fitted R6 Class of RandomForestClassifier

Examples:

```
data(iris)
split_list <- train_test_split(iris, test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
```

```
y_test <- split_list[[4]]

rf <- RandomForestClassifier$new()
rf$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X0.

Usage:

```
RandomForestClassifier$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: Factor of the predict classes.

Examples:

```
rf <- RandomForestClassifier$new()
rf$fit(X_train, y_train)
preds <- rf$predict(X_test)

rf <- RandomForestClassifier$new()
preds <- rf$fit(X_train, y_train)$predict(X_test)

preds <- RandomForestClassifier$new()$fit(X_train, y_train)$predict(X_test)
print(caret::confusionMatrix(data=preds, reference = factor(y_test)))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
RandomForestClassifier$get_estimator_type()
```

Examples:

```
rf$get_estimator_type()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`randomForest::randomForest()`

Examples

```
## ------------------------------------------------
## Method `RandomForestClassifier$new`
## ------------------------------------------------
```

```
rf <- RandomForestClassifier$new()
rf <- RandomForestClassifier$new(n_estimators = 500)
rf <- RandomForestClassifier$new(n_estimators = 500, random_state = 100)

## ------------------------------------------------
## Method `RandomForestClassifier$fit`
## ------------------------------------------------

data(iris)
split_list <- train_test_split(iris, test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

rf <- RandomForestClassifier$new()
rf$fit(X_train, y_train)

## ------------------------------------------------
## Method `RandomForestClassifier$predict`
## ------------------------------------------------

rf <- RandomForestClassifier$new()
rf$fit(X_train, y_train)
preds <- rf$predict(X_test)

rf <- RandomForestClassifier$new()
preds <- rf$fit(X_train, y_train)$predict(X_test)

preds <- RandomForestClassifier$new()$fit(X_train, y_train)$predict(X_test)
print(caret::confusionMatrix(data=preds, reference = factor(y_test)))

## ------------------------------------------------
## Method `RandomForestClassifier$get_estimator_type`
## ------------------------------------------------

rf$get_estimator_type()
```

RandomForestRegressor *RandomForestRegressor*

Description

Wrapper R6 Class of randomForest::randomForest function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of RandomForestRegressor

Super classes

less::BaseEstimator -> less::SklearnEstimator -> RandomForestRegressor

Methods

Public methods:

- RandomForestRegressor$new()
- RandomForestRegressor$fit()
- RandomForestRegressor$predict()
- RandomForestRegressor$get_estimator_type()
- RandomForestRegressor$clone()

Method `new()`**:** Creates a new instance of R6 Class of RandomForestRegressor

Usage:

```
RandomForestRegressor$new(
  n_estimators = 100,
  random_state = NULL,
  min_samples_leaf = 1,
  max_leaf_nodes = NULL
)
```

Arguments:

`n_estimators` Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times (defaults to 100).

`random_state` Seed number to be used for fixing the randomness (default to NULL).

`min_samples_leaf` Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time) (defaults to 1)

`max_leaf_nodes` Maximum number of terminal nodes trees in the forest can have. If not given, trees are grown to the maximum possible (subject to limits by nodesize). If set larger than maximum possible, a warning is issued. (defaults to NULL)

Examples:

```
rf <- RandomForestRegressor$new()
rf <- RandomForestRegressor$new(n_estimators = 500)
rf <- RandomForestRegressor$new(n_estimators = 500, random_state = 100)
```

Method `fit()`**:** Builds a random forest regressor from the training set (X, y).

Usage:

```
RandomForestRegressor$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes response variables

Returns: Fitted R6 Class of RandomForestRegressor

Examples:

```
data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

rf <- RandomForestRegressor$new()
rf$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X0.

Usage:

```
RandomForestRegressor$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: The predict values.

Examples:

```
preds <- rf$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
RandomForestRegressor$get_estimator_type()
```

Examples:

```
rf$get_estimator_type()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`randomForest::randomForest()`

Examples

```
## ------------------------------------------------
## Method `RandomForestRegressor$new`
## ------------------------------------------------

rf <- RandomForestRegressor$new()
rf <- RandomForestRegressor$new(n_estimators = 500)
```

```
rf <- RandomForestRegressor$new(n_estimators = 500, random_state = 100)

## ------------------------------------------------
## Method `RandomForestRegressor$fit`
## ------------------------------------------------

data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

rf <- RandomForestRegressor$new()
rf$fit(X_train, y_train)

## ------------------------------------------------
## Method `RandomForestRegressor$predict`
## ------------------------------------------------

preds <- rf$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))

## ------------------------------------------------
## Method `RandomForestRegressor$get_estimator_type`
## ------------------------------------------------

rf$get_estimator_type()
```

---

rbf *RBF kernel - L2 norm*

Description

The default distance function in LESS.

Usage

```
rbf(data, center, coeff = 0.01)
```

Arguments

data	Data that includes points in shape of **(M x d)**
center	A constant point in shape of **(1 x d)**
coeff	Coefficient value for RBF kernel

Value

A numeric vector containing the Radial basis function kernel distance between each point in **data** and **center**.

Examples

```
data <- matrix(1:12, nrow=3)
center <- c(2, 7, 1, 3)
distances <- rbf(data, center)
print(distances)
```

SklearnEstimator *SklearnEstimator*

Description

A dummy base R6 class that includes fit, predict functions for estimators

Value

R6 Class of SklearnEstimator

Super class

less::BaseEstimator -> SklearnEstimator

Methods

Public methods:

- SklearnEstimator$fit()
- SklearnEstimator$predict()
- SklearnEstimator$get_type()
- SklearnEstimator$get_isFitted()
- SklearnEstimator$clone()

Method `fit()`**:** Dummy fit function

Usage:

```
SklearnEstimator$fit()
```

Examples:

```
sklearn <- SklearnEstimator$new()
sklearn$fit()
```

Method `predict()`**:** Dummy predict function

Usage:

```
SklearnEstimator$predict()
```

Examples:

```
sklearn$predict()
```

Method `get_type()`**:** Auxiliary function returning the type of the class e.g 'estimator'

Usage:

```
SklearnEstimator$get_type()
```

Examples:

```
sklearn$get_type()
```

Method `get_isFitted()`**:** Auxiliary function returning the isFitted flag

Usage:

```
SklearnEstimator$get_isFitted()
```

Examples:

```
sklearn$get_isFitted()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SklearnEstimator$fit`
## ------------------------------------------------

sklearn <- SklearnEstimator$new()
sklearn$fit()

## ------------------------------------------------
## Method `SklearnEstimator$predict`
## ------------------------------------------------

sklearn$predict()

## ------------------------------------------------
## Method `SklearnEstimator$get_type`
## ------------------------------------------------

sklearn$get_type()

## ------------------------------------------------
## Method `SklearnEstimator$get_isFitted`
## ------------------------------------------------

sklearn$get_isFitted()
```

SVC *Support Vector Classification*

Description

Wrapper R6 Class of e1071::svm function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of SVC

Super classes

`less::BaseEstimator` -> `less::SklearnEstimator` -> SVC

Methods

Public methods:

- `SVC$new()`
- `SVC$fit()`
- `SVC$predict()`
- `SVC$get_estimator_type()`
- `SVC$clone()`

Method `new()`**:** Creates a new instance of R6 Class of SVC

Usage:

```
SVC$new(
  scale = TRUE,
  kernel = "radial",
  degree = 3,
  gamma = NULL,
  coef0 = 0,
  cost = 1,
  cache_size = 40,
  tolerance = 0.001,
  epsilon = 0.1,
  shrinking = TRUE,
  cross = 0,
  probability = FALSE,
  fitted = TRUE
)
```

Arguments:

`scale` A logical vector indicating the variables to be scaled. If scale is of length 1, the value is recycled as many times as needed. Per default, data are scaled internally (both x and y variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions (default: TRUE)

`kernel` The kernel used in training and predicting. Possible values are: "linear", "polynomial", "radial", "sigmoid" (default is "radial")

`degree` Parameter needed for kernel of type polynomial (default: 3)

`gamma` Parameter needed for all kernels except linear (default: 1/(data dimension))

`coef0` Parameter needed for kernels of type polynomial and sigmoid (default: 0)

`cost` Cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation (default: 1)

`cache_size` Cache memory in MB (default: 40)

`tolerance` Tolerance of termination criterion (default: 0.001)

`epsilon` Epsilon in the insensitive-loss function (default: 0.1)

`shrinking` Option whether to use the shrinking-heuristics (default: TRUE)

`cross` If a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the accuracy rate for classification and the Mean Squared Error for regression (default: 0)

`probability` Logical indicating whether the model should allow for probability predictions (default: FALSE)

`fitted` Logical indicating whether the fitted values should be computed and included in the model or not (default: TRUE)

Examples:

```
svc <- SVC$new()
svc <- SVC$new(kernel = "polynomial")
```

Method `fit()`**:** Fit the SVM model from the training set (X, y).

Usage:

```
SVC$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes labels

Returns: Fitted R6 Class of SVC

Examples:

```
data(iris)
split_list <- train_test_split(iris, test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

svc <- SVC$new()
svc$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X0.

Usage:

```
SVC$predict(X0)
```

Arguments:

X0 2D matrix or dataframe that includes predictors

Returns: Factor of the predict classes.

Examples:

```
svc <- SVC$new()
svc$fit(X_train, y_train)
preds <- svc$predict(X_test)

svc <- SVC$new()
preds <- svc$fit(X_train, y_train)$predict(X_test)

preds <- SVC$new()$fit(X_train, y_train)$predict(X_test)
print(caret::confusionMatrix(data=preds, reference = factor(y_test)))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
SVC$get_estimator_type()
```

Examples:

```
svc$get_estimator_type()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

e1071::svm()

Examples

```
## ------------------------------------------------
## Method `SVC$new`
## ------------------------------------------------

svc <- SVC$new()
svc <- SVC$new(kernel = "polynomial")

## ------------------------------------------------
## Method `SVC$fit`
## ------------------------------------------------

data(iris)
split_list <- train_test_split(iris, test_size =  0.3)
X_train <- split_list[[1]]
```

```
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

svc <- SVC$new()
svc$fit(X_train, y_train)

## ------------------------------------------------
## Method `SVC$predict`
## ------------------------------------------------

svc <- SVC$new()
svc$fit(X_train, y_train)
preds <- svc$predict(X_test)

svc <- SVC$new()
preds <- svc$fit(X_train, y_train)$predict(X_test)

preds <- SVC$new()$fit(X_train, y_train)$predict(X_test)
print(caret::confusionMatrix(data=preds, reference = factor(y_test)))

## ------------------------------------------------
## Method `SVC$get_estimator_type`
## ------------------------------------------------

svc$get_estimator_type()
```

SVR *Support Vector Regression*

Description

Wrapper R6 Class of e1071::svm function that can be used for LESSRegressor and LESSClassifier

Value

R6 Class of SVR

Super classes

`less::BaseEstimator` -> `less::SklearnEstimator` -> SVR

Methods

Public methods:

- `SVR$new()`
- `SVR$fit()`
- `SVR$predict()`
- `SVR$get_estimator_type()`

- SVR$clone()

Method `new()`**:** Creates a new instance of R6 Class of SVR

Usage:

```
SVR$new(
  scale = TRUE,
  kernel = "radial",
  degree = 3,
  gamma = NULL,
  coef0 = 0,
  cost = 1,
  cache_size = 40,
  tolerance = 0.001,
  epsilon = 0.1,
  shrinking = TRUE,
  cross = 0,
  probability = FALSE,
  fitted = TRUE
)
```

Arguments:

`scale` A logical vector indicating the variables to be scaled. If scale is of length 1, the value is recycled as many times as needed. Per default, data are scaled internally (both x and y variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions (default: TRUE)

`kernel` The kernel used in training and predicting. Possible values are: "linear", "polynomial", "radial", "sigmoid" (default is "radial")

`degree` Parameter needed for kernel of type polynomial (default: 3)

`gamma` Parameter needed for all kernels except linear (default: 1/(data dimension))

`coef0` Parameter needed for kernels of type polynomial and sigmoid (default: 0)

`cost` Cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation (default: 1)

`cache_size` Cache memory in MB (default: 40)

`tolerance` Tolerance of termination criterion (default: 0.001)

`epsilon` Epsilon in the insensitive-loss function (default: 0.1)

`shrinking` Option whether to use the shrinking-heuristics (default: TRUE)

`cross` If a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the accuracy rate for classification and the Mean Squared Error for regression (default: 0)

`probability` Logical indicating whether the model should allow for probability predictions (default: FALSE)

`fitted` Logical indicating whether the fitted values should be computed and included in the model or not (default: TRUE)

Examples:

```
svr <- SVR$new()
svr <- SVR$new(kernel = "polynomial")
```

Method `fit()`**:** Fit the SVM model from the training set (X, y).

Usage:

```
SVR$fit(X, y)
```

Arguments:

`X` 2D matrix or dataframe that includes predictors

`y` 1D vector or (n,1) dimensional matrix/dataframe that includes response variables

Returns: Fitted R6 Class of SVR

Examples:

```
data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

svr <- SVR$new()
svr$fit(X_train, y_train)
```

Method `predict()`**:** Predict regression value for X0.

Usage:

```
SVR$predict(X0)
```

Arguments:

`X0` 2D matrix or dataframe that includes predictors

Returns: The predict values.

Examples:

```
svr <- SVR$new()
svr$fit(X_train, y_train)
preds <- svr$predict(X_test)

svr <- SVR$new()
preds <- svr$fit(X_train, y_train)$predict(X_test)

preds <- SVR$new()$fit(X_train, y_train)$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))
```

Method `get_estimator_type()`**:** Auxiliary function returning the estimator type e.g 'regressor', 'classifier'

Usage:

```
SVR$get_estimator_type()
```

Examples:

```
svr$get_estimator_type()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

e1071::svm()

Examples

```
## ------------------------------------------------
## Method `SVR$new`
## ------------------------------------------------

svr <- SVR$new()
svr <- SVR$new(kernel = "polynomial")

## ------------------------------------------------
## Method `SVR$fit`
## ------------------------------------------------

data(abalone)
split_list <- train_test_split(abalone[1:100,], test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
y_test <- split_list[[4]]

svr <- SVR$new()
svr$fit(X_train, y_train)

## ------------------------------------------------
## Method `SVR$predict`
## ------------------------------------------------

svr <- SVR$new()
svr$fit(X_train, y_train)
preds <- svr$predict(X_test)

svr <- SVR$new()
preds <- svr$fit(X_train, y_train)$predict(X_test)

preds <- SVR$new()$fit(X_train, y_train)$predict(X_test)
print(head(matrix(c(y_test, preds), ncol = 2, dimnames = (list(NULL, c("True", "Prediction"))))))

## ------------------------------------------------
## Method `SVR$get_estimator_type`
## ------------------------------------------------

svr$get_estimator_type()
```

---

synthetic_sine_curve *Synthetic Sine Curve*

Description

A simple function to generate n_samples from sine curve in the range (-10, 10) with some amplitude. The function returns the dataset (X, y), and plots the function (curve) along with the dataset (circles)

Usage

```
synthetic_sine_curve(n_samples = 200)
```

Arguments

n_samples	Number of data points to be generated

Examples

```
sine_data_list <- synthetic_sine_curve()
X_sine <- sine_data_list[[1]]
y_sine <- sine_data_list[[2]]
```

---

test_timing *Compare Fitting Time*

Description

Plots a histogram chart which shows the fitting time obtained from various regressors/classifiers (using their default values) on the given dataset (X, y).

Usage

```
test_timing(type = 1, X, y)
```

Arguments

type	1 to compare regressors, 2 for comparing classifiers
X	Predictors
y	Response variables

Examples

```
X <- matrix(sample(100, 20), nrow = 10)
y <- sample(100, 10)
test_timing(1, X, y)
```

`train_test_split` *Dataset splitting*

Description

Split dataframes or matrices into random train and test subsets. Takes the column at the **y_index** of **data** as response variable **(y)** and the rest as the independent variables **(X)**

Usage

```
train_test_split(
  data,
  test_size = 0.3,
  random_state = NULL,
  y_index = ncol(data)
)
```

Arguments

`data`	Dataset that is going to be split
`test_size`	Represents the proportion of the dataset to include in the test split. Should be between 0.0 and 1.0 (defaults to 0.3)
`random_state`	Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls (defaults to NULL)
`y_index`	Corresponding column index of the response variable **y** (defaults to last column of **data**)

Value

A `list` of length 4 with elements:

`X_train`	Training input variables
`X_test`	Test input variables
`y_train`	Training response variables
`y_test`	Test response variables

Examples

```
data(abalone)
split_list <- train_test_split(abalone, test_size =  0.3)
X_train <- split_list[[1]]
X_test <- split_list[[2]]
y_train <- split_list[[3]]
```

```
y_test <- split_list[[4]]

print(head(X_train))
print(head(X_test))
print(head(y_train))
print(head(y_test))
```

Index