Package 'daltoolbox'

June 28, 2025

Title Leveraging Experiment Lines to Data Analytics

Version 1.2.727

Description
The natural increase in the complexity of current research experiments and data demands better tools to enhance productivity in Data Analytics. The package is a framework designed to address the modern challenges in data analytics workflows. The package is inspired by Experiment Line concepts. It aims to provide seamless support for users in developing their data mining workflows by offering a uniform data model and method API. It enables the integration of various data mining activities, including data preprocessing, classification, regression, clustering, and time series prediction. It also offers options for hyper-parameter tuning and supports integration with existing libraries and languages. Overall, the package provides researchers with a comprehensive set of functionalities for data science, promoting ease of use, extensibility, and integration with various tools and libraries. Information on Experiment Line is based on Ogasawara et al. (2009) <doi:10.1007/978-3-642-02279-1_20>.

License MIT + file LICENSE

URL https://cefet-rj-dal.github.io/daltoolbox/,
https://github.com/cefet-rj-dal/daltoolbox

BugReports https://github.com/cefet-rj-dal/daltoolbox/issues

Encoding UTF-8

Depends R (>= 4.1.0)

RoxygenNote 7.3.2

Imports FNN, caret, class, cluster, dbscan, dplyr, e1071, ggplot2,
nnet, randomForest, reshape, tree

NeedsCompilation no

Author Eduardo Ogasawara [aut, ths, cre] (ORCID:
<https://orcid.org/0000-0002-0466-0626>),
Antonio Castro [aut],
Diego Salles [aut],
Janio Lima [aut],
Lucas Tavares [aut],
Diego Carvalho [ctb],
Eduardo Bezerra [ctb],

Rafaelli Coutinho [ctb],
CEFET/RJ [cph]

Maintainer Eduardo Ogasawara <eogasawara@ieee.org>

Repository CRAN

Date/Publication 2025-06-28 16:20:01 UTC

Contents

action *Action*

Description

Executes the action of model applied in provided data

Usage

```
action(obj, ...)
```

Arguments

obj	object: a dal_base object to apply the transformation on the input dataset.
...	optional arguments.

Value

returns the result of an action of the model applied in provided data

Examples

```
data(iris)
# an example is minmax normalization
trans <- minmax()
trans <- fit(trans, iris)
tiris <- action(trans, iris)
```

action.dal_transform *Action implementation for transform*

Description

A default function that defines the action to proxy transform method

Usage

```
## S3 method for class 'dal_transform'
action(obj, ...)
```

Arguments

obj	object
...	optional arguments

Value

returns a transformed data

Examples

```
#See ?minmax for an example of transformation
```

---

adjust_class_label *Adjust categorical mapping*

Description

Converts a vector into a categorical mapping, where each category is represented by a specific value. By default, the values represent binary categories (true/false)

Usage

```
adjust_class_label(x, valTrue = 1, valFalse = 0)
```

Arguments

x	vector to be categorized
valTrue	value to represent true
valFalse	value to represent false

Value

returns an adjusted categorical mapping

---

adjust_data.frame *Adjust to data frame*

Description

Converts a dataset to a data.frame if it is not already in that format

Usage

```
adjust_data.frame(data)
```

Arguments

data	dataset

Value

returns a data.frame

Examples

```
data(iris)
df <- adjust_data.frame(iris)
```

`adjust_factor` *Adjust factors*

Description

Converts a vector into a factor with specified levels and labels

Usage

```
adjust_factor(value, ilevels, slevels)
```

Arguments

`value`	vector to be converted into factor
`ilevels`	order for categorical values
`slevels`	labels for categorical values

Value

returns an adjusted factor

`adjust_matrix` *Adjust to matrix*

Description

Converts a dataset to a matrix format if it is not already in that format

Usage

```
adjust_matrix(data)
```

Arguments

`data`	dataset

Value

returns an adjusted matrix

Examples

```
data(iris)
mat <- adjust_matrix(iris)
```

---

autoenc_base_e *Autoencoder - Encode*

Description

Creates a base class for autoencoder.

Usage

```
autoenc_base_e(input_size, encoding_size)
```

Arguments

input_size	input size
encoding_size	encoding size

Value

returns a autoenc_base_e object.

Examples

```
#See an example of using `autoenc_base_e` at this
#https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_base_e.md
```

---

autoenc_base_ed *Autoencoder - Encode-decode*

Description

Creates a base class for autoencoder.

Usage

```
autoenc_base_ed(input_size, encoding_size)
```

Arguments

input_size	input size
encoding_size	encoding size

Value

returns a autoenc_base_ed object.

Examples

```
#See an example of using `autoenc_base_ed` at this
#https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_base_ed.md
```

Boston *Boston Housing Data (Regression)*

Description

housing values in suburbs of Boston.

- crim: per capita crime rate by town.
- zn: proportion of residential land zoned for lots over 25,000 sq.ft.
- indus: proportion of non-retail business acres per town
- chas: Charles River dummy variable (= 1 if tract bounds)
- nox: nitric oxides concentration (parts per 10 million)
- rm: average number of rooms per dwelling
- age: proportion of owner-occupied units built prior to 1940
- dis: weighted distances to five Boston employment centres
- rad: index of accessibility to radial highways
- tax: full-value property-tax rate per $10,000
- ptratio: pupil-teacher ratio by town
- black: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- lstat: percentage of lower status of the population
- medv: Median value of owner-occupied homes in $1000's

Usage

```
data(Boston)
```

Format

Regression Dataset.

Source

This dataset was obtained from the MASS library.

References

Creator: Harrison, D. and Rubinfeld, D.L. Hedonic prices and the demand for clean air, J. Environ. Economics & Management, vol.5, 81-102, 1978.

Examples

```
data(Boston)
head(Boston)
```

---

categ_mapping *Categorical mapping*

Description

Categorical mapping provides a way to map the levels of a categorical variable to new values. Each possible value is converted to a binary attribute.

Usage

```
categ_mapping(attribute)
```

Arguments

attribute	attribute to be categorized.

Value

returns a data frame with binary attributes, one for each possible category.

Examples

```
cm <- categ_mapping("Species")
iris_cm <- transform(cm, iris)

# can be made in a single column
species <- iris[,"Species", drop=FALSE]
iris_cm <- transform(cm, species)
```

classification *classification*

Description

Ancestor class for classification problems

Usage

```
classification(attribute, slevels)
```

Arguments

attribute	attribute target to model building
slevels	possible values for the target classification

Value

returns a classification object

Examples

```
#See ?cla_dtree for a classification example using a decision tree
```

cla_dtree *Decision Tree for classification*

Description

Creates a classification object that uses the Decision Tree algorithm for classification. It wraps the tree library.

Usage

```
cla_dtree(attribute, slevels)
```

Arguments

attribute	attribute target to model building
slevels	the possible values for the target classification

Value

returns a classification object

Examples

```
data(iris)
slevels <- levels(iris$Species)
model <- cla_dtree("Species", slevels)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, iris)
train <- sr$train
test <- sr$test

model <- fit(model, train)

prediction <- predict(model, test)
predictand <- adjust_class_label(test[,"Species"])
test_eval <- evaluate(model, predictand, prediction)
test_eval$metrics
```

---

cla_knn *K Nearest Neighbor Classification*

Description

Classifies using the K-Nearest Neighbor algorithm. It wraps the class library.

Usage

```
cla_knn(attribute, slevels, k = 1)
```

Arguments

attribute	attribute target to model building.
slevels	possible values for the target classification.
k	a vector of integers indicating the number of neighbors to be considered.

Value

returns a knn object.

Examples

```
data(iris)
slevels <- levels(iris$Species)
model <- cla_knn("Species", slevels, k=3)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, iris)
```

```
train <- sr$train
test <- sr$test

model <- fit(model, train)

prediction <- predict(model, test)
predictand <- adjust_class_label(test[,"Species"])
test_eval <- evaluate(model, predictand, prediction)
test_eval$metrics
```

`cla_majority` *Majority Classification*

Description

This function creates a classification object that uses the majority vote strategy to predict the target attribute. Given a target attribute, the function counts the number of occurrences of each value in the dataset and selects the one that appears most often.

Usage

```
cla_majority(attribute, slevels)
```

Arguments

`attribute`	attribute target to model building.
`slevels`	possible values for the target classification.

Value

returns a classification object.

Examples

```
data(iris)
slevels <- levels(iris$Species)
model <- cla_majority("Species", slevels)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, iris)
train <- sr$train
test <- sr$test

model <- fit(model, train)

prediction <- predict(model, test)
predictand <- adjust_class_label(test[,"Species"])
test_eval <- evaluate(model, predictand, prediction)
test_eval$metrics
```

cla_mlp *MLP for classification*

Description

Creates a classification object that uses the Multi-Layer Perceptron (MLP) method. It wraps the nnet library.

Usage

```
cla_mlp(attribute, slevels, size = NULL, decay = 0.1, maxit = 1000)
```

Arguments

attribute	attribute target to model building
slevels	possible values for the target classification
size	number of nodes that will be used in the hidden layer
decay	how quickly it decreases in gradient descent
maxit	maximum iterations

Value

returns a classification object

Examples

```
data(iris)
slevels <- levels(iris$Species)
model <- cla_mlp("Species", slevels, size=3, decay=0.03)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, iris)
train <- sr$train
test <- sr$test

model <- fit(model, train)

prediction <- predict(model, test)
predictand <- adjust_class_label(test[,"Species"])
test_eval <- evaluate(model, predictand, prediction)
test_eval$metrics
```

cla_nb *Naive Bayes Classifier*

Description

Classification using the Naive Bayes algorithm It wraps the e1071 library.

Usage

```
cla_nb(attribute, slevels)
```

Arguments

attribute	attribute target to model building.
slevels	possible values for the target classification.

Value

returns a classification object.

Examples

```
data(iris)
slevels <- levels(iris$Species)
model <- cla_nb("Species", slevels)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, iris)
train <- sr$train
test <- sr$test

model <- fit(model, train)

prediction <- predict(model, test)
predictand <- adjust_class_label(test[,"Species"])
test_eval <- evaluate(model, predictand, prediction)
test_eval$metrics
```

cla_rf *Random Forest for classification*

Description

Creates a classification object that uses the Random Forest method It wraps the randomForest library.

Usage

```
cla_rf(attribute, slevels, nodesize = 5, ntree = 10, mtry = NULL)
```

Arguments

attribute	attribute target to model building
slevels	possible values for the target classification
nodesize	node size
ntree	number of trees
mtry	number of attributes to build tree

Value

returns a classification object

Examples

```
data(iris)
slevels <- levels(iris$Species)
model <- cla_rf("Species", slevels, ntree=5)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, iris)
train <- sr$train
test <- sr$test

model <- fit(model, train)

prediction <- predict(model, test)
predictand <- adjust_class_label(test[,"Species"])
test_eval <- evaluate(model, predictand, prediction)
test_eval$metrics
```

---

cla_svm *SVM for classification*

Description

Creates a classification object that uses the Support Vector Machine (SVM) method for classification It wraps the e1071 and svm library.

Usage

```
cla_svm(attribute, slevels, epsilon = 0.1, cost = 10, kernel = "radial")
```

Arguments

attribute	attribute target to model building
slevels	possible values for the target classification
epsilon	parameter that controls the width of the margin around the separating hyperplane
cost	parameter that controls the trade-off between having a wide margin and correctly classifying training data points
kernel	the type of kernel function to be used in the SVM algorithm (linear, radial, polynomial, sigmoid)

Value

returns a SVM classification object

Examples

```
data(iris)
slevels <- levels(iris$Species)
model <- cla_svm("Species", slevels, epsilon=0.0,cost=20.000)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, iris)
train <- sr$train
test <- sr$test

model <- fit(model, train)

prediction <- predict(model, test)
predictand <- adjust_class_label(test[,"Species"])
test_eval <- evaluate(model, predictand, prediction)
test_eval$metrics
```

---

cla_tune *Classification Tune*

Description

This function performs a grid search or random search over specified hyperparameter values to optimize a base classification model

Usage

```
cla_tune(base_model, folds = 10, metric = "accuracy")
```

Arguments

base_model	base model for tuning
folds	number of folds for cross-validation
metric	metric used to optimize

Value

returns a cla_tune object

Examples

```
# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, iris)
train <- sr$train
test <- sr$test

# hyper parameter setup
tune <- cla_tune(cla_mlp("Species", levels(iris$Species)))
ranges <- list(size=c(3:5), decay=c(0.1))

# hyper parameter optimization
model <- fit(tune, train, ranges)

# testing optimization
test_prediction <- predict(model, test)
test_predictand <- adjust_class_label(test[,"Species"])
test_eval <- evaluate(model, test_predictand, test_prediction)
test_eval$metrics
```

---

cluster *Cluster*

Description

Defines a cluster method

Usage

```
cluster(obj, ...)
```

Arguments

obj	a clusterer object
...	optional arguments

Value

clustered data

Examples

```
#See ?cluster_kmeans for an example of transformation
```

`clusterer` *Clusterer*

Description

Ancestor class for clustering problems

Usage

```
clusterer()
```

Value

returns a `clusterer` object

Examples

```
#See ?cluster_kmeans for an example of transformation
```

`cluster_dbscan` *DBSCAN*

Description

Creates a clusterer object that uses the DBSCAN method It wraps the dbscan library.

Usage

```
cluster_dbscan(minPts = 3, eps = NULL)
```

Arguments

`minPts`	minimum number of points
`eps`	distance value

Value

returns a dbscan object

Examples

```
# setup clustering
model <- cluster_dbscan(minPts = 3)

#load dataset
data(iris)

# build model
model <- fit(model, iris[,1:4])
clu <- cluster(model, iris[,1:4])
table(clu)

# evaluate model using external metric
eval <- evaluate(model, clu, iris$Species)
eval
```

`cluster_kmeans` *k-means*

Description

Creates a clusterer object that uses the k-means method It wraps the stats library.

Usage

```
cluster_kmeans(k = 1)
```

Arguments

k	the number of clusters to form.

Value

returns a k-means object.

Examples

```
# setup clustering
model <- cluster_kmeans(k=3)

#load dataset
data(iris)

# build model
model <- fit(model, iris[,1:4])
clu <- cluster(model, iris[,1:4])
table(clu)

# evaluate model using external metric
eval <- evaluate(model, clu, iris$Species)
eval
```

cluster_pam *PAM*

Description

Creates a clusterer object that uses the Partition Around Medoids (PAM) method It wraps the cluster library.

Usage

```
cluster_pam(k = 1)
```

Arguments

k	the number of clusters to generate.

Value

returns PAM object.

Examples

```
# setup clustering
model <- cluster_pam(k = 3)

#load dataset
data(iris)

# build model
model <- fit(model, iris[,1:4])
clu <- cluster(model, iris[,1:4])
table(clu)

# evaluate model using external metric
eval <- evaluate(model, clu, iris$Species)
eval
```

clu_tune *Clustering Tune*

Description

Creates an object for tuning clustering models. This object can be used to fit and optimize clustering algorithms by specifying hyperparameter ranges

Usage

```
clu_tune(base_model)
```

Arguments

base_model	base model for tuning

Value

returns a clu_tune object.

Examples

```
data(iris)

# fit model
model <- clu_tune(cluster_kmeans(k = 0))
ranges <- list(k = 1:10)
model <- fit(model, iris[,1:4], ranges)
model$k
```

---

dal_base *Class dal_base*

Description

The dal_base class is an abstract class for all dal descendants classes. It provides both fit() and action() functions

Usage

```
dal_base()
```

Value

returns a dal_base object

Examples

```
trans <- dal_base()
```

dal_learner *DAL Learner*

Description

A ancestor class for clustering, classification, regression, and time series regression. It also provides the basis for specialized evaluation of learning performance.

An example of a learner is a decision tree (cla_dtree)

Usage

```
dal_learner()
```

Value

returns a learner

Examples

```
#See ?cla_dtree for a classification example using a decision tree
```

dal_transform *DAL Transform*

Description

A transformation method applied to a dataset. If needed, the fit can be called to adjust the transform.

Usage

```
dal_transform()
```

Value

returns a `dal_transform` object.

Examples

```
#See ?minmax for an example of transformation
```

dal_tune *DAL Tune*

Description

Creates an ancestor class for hyperparameter optimization, allowing the tuning of a base model using cross-validation.

Usage

```
dal_tune(base_model, folds = 10)
```

Arguments

base_model	base model for tuning
folds	number of folds for cross-validation

Value

returns a dal_tune object

Examples

```
#See ?cla_tune for classification tuning
#See ?reg_tune for regression tuning
#See ?ts_tune for time series tuning
```

data_sample *Data Sample*

Description

The data_sample function in R is used to randomly sample data from a given data frame. It can be used to obtain a subset of data for further analysis or modeling.

Two basic specializations of data_sample are sample_random and sample_stratified. They provide random sampling and stratified sampling, respectively.

Data sample provides both training and testing partitioning (train_test) and k-fold partitioning (k_fold) of data.

Usage

```
data_sample()
```

Value

returns an object of class data_sample

Examples

```
#using random sampling
sample <- sample_random()
tt <- train_test(sample, iris)

# distribution of train
table(tt$train$Species)

# preparing dataset into four folds
folds <- k_fold(sample, iris, 4)

# distribution of folds
tbl <- NULL
for (f in folds) {
 tbl <- rbind(tbl, table(f$Species))
}
head(tbl)
```

---

dt_pca *PCA*

Description

PCA (Principal Component Analysis) is an unsupervised dimensionality reduction technique used in data analysis and machine learning. It transforms a dataset of possibly correlated variables into a new set of uncorrelated variables called principal components.

Usage

```
dt_pca(attribute = NULL, components = NULL)
```

Arguments

attribute	target attribute to model building
components	number of components for PCA

Value

returns an object of class dt_pca

Examples

```
mypca <- dt_pca("Species")
# Automatically fitting number of components
mypca <- fit(mypca, iris)
iris.pca <- transform(mypca, iris)
head(iris.pca)
head(mypca$pca.transf)
# Manual establishment of number of components
```

```
mypca <- dt_pca("Species", 3)
mypca <- fit(mypca, datasets::iris)
iris.pca <- transform(mypca, iris)
head(iris.pca)
head(mypca$pca.transf)
```

---

evaluate *Evaluate*

Description

Evaluate learner performance. The actual evaluate varies according to the type of learner (clustering, classification, regression, time series regression)

Usage

```
evaluate(obj, ...)
```

Arguments

obj	object
...	optional arguments

Value

returns the evaluation

Examples

```
data(iris)
slevels <- levels(iris$Species)
model <- cla_dtree("Species", slevels)
model <- fit(model, iris)
prediction <- predict(model, iris)
predictand <- adjust_class_label(iris[,"Species"])
test_eval <- evaluate(model, predictand, prediction)
test_eval$metrics
```

fit *Fit*

Description

Applies the `fit` method to a model object to train or configure it using the provided data and optional arguments

Usage

```
fit(obj, ...)
```

Arguments

`obj`	object
`...`	optional arguments.

Value

returns a object after fitting

Examples

```
data(iris)
# an example is minmax normalization
trans <- minmax()
trans <- fit(trans, iris)
tiris <- action(trans, iris)
```

fit.cla_tune *tune hyperparameters of ml model*

Description

Tunes the hyperparameters of a machine learning model for classification

Usage

```
## S3 method for class 'cla_tune'
fit(obj, data, ranges, ...)
```

Arguments

`obj`	an object containing the model and tuning configuration
`data`	the dataset used for training and evaluation
`ranges`	a list of hyperparameter ranges to explore
`...`	optional arguments

Value

a fitted obj

`fit.cluster_dbscan` *fit dbscan model*

Description

Fits a DBSCAN clustering model by setting the `eps` parameter. If `eps` is not provided, it is estimated based on the k-nearest neighbor distances. It wraps dbscan library

Usage

```
## S3 method for class 'cluster_dbscan'
fit(obj, data, ...)
```

Arguments

`obj`	an object containing the DBSCAN model configuration, including `minPts` and optionally `eps`
`data`	the dataset to use for fitting the model
`...`	optional arguments

Value

returns a fitted obj with the `eps` parameter set

`fit_curvature_max` *maximum curvature analysis*

Description

Fitting a curvature model in a sequence of observations. It extracts the the maximum curvature computed.

Usage

```
fit_curvature_max()
```

Value

returns an object of class fit_curvature_max, which inherits from the fit_curvature and dal_transform classes. The object contains a list with the following elements:

- x: The position in which the maximum curvature is reached.
- y: The value where the the maximum curvature occurs.
- yfit: The value of the maximum curvature.

Examples

```
x <- seq(from=1,to=10,by=0.5)
dat <- data.frame(x = x, value = -log(x), variable = "log")
myfit <- fit_curvature_max()
res <- transform(myfit, dat$value)
head(res)
```

`fit_curvature_min` *minimum curvature analysis*

Description

Fitting a curvature model in a sequence of observations. It extracts the the minimum curvature computed.

Usage

```
fit_curvature_min()
```

Value

Returns an object of class fit_curvature_max, which inherits from the fit_curvature and dal_transform classes. The object contains a list with the following elements:

- x: The position in which the minimum curvature is reached.
- y: The value where the the minimum curvature occurs.
- yfit: The value of the minimum curvature.

Examples

```
x <- seq(from=1,to=10,by=0.5)
dat <- data.frame(x = x, value = log(x), variable = "log")
myfit <- fit_curvature_min()
res <- transform(myfit, dat$value)
head(res)
```

inverse_transform *Inverse Transform*

Description

Reverses the transformation applied to data.

Usage

```
inverse_transform(obj, ...)
```

Arguments

obj	a dal_transform object.
...	optional arguments.

Value

dataset inverse transformed.

Examples

```
#See ?minmax for an example of transformation
```

k_fold *K-fold sampling*

Description

k-fold partition of a dataset using a sampling method

Usage

```
k_fold(obj, data, k)
```

Arguments

obj	an object representing the sampling method
data	dataset to be partitioned
k	number of folds

Value

returns a list of k data frames

Examples

```
#using random sampling
sample <- sample_random()

# preparing dataset into four folds
folds <- k_fold(sample, iris, 4)

# distribution of folds
tbl <- NULL
for (f in folds) {
 tbl <- rbind(tbl, table(f$Species))
}
head(tbl)
```

`minmax` *Min-max normalization*

Description

The minmax performs scales data between [0,1].

$minmax = (x - min(x))/(max(x) - min(x))$

Usage

```
minmax()
```

Value

returns an object of class `minmax`

Examples

```
data(iris)
head(iris)

trans <- minmax()
trans <- fit(trans, iris)
tiris <- transform(trans, iris)
head(tiris)

itiris <- inverse_transform(trans, tiris)
head(itiris)
```

outliers_boxplot *outliers_boxplot*

Description

The outliers_boxplot class uses box-plot definition for outliers_boxplot. An outlier is a value that is below than $Q_1 - 1.5 \cdot IQR$ or higher than $Q_3 + 1.5 \cdot IQR$. The class remove outliers_boxplot for numeric attributes. Users can set alpha to 3 to remove extreme values.

Usage

```
outliers_boxplot(alpha = 1.5)
```

Arguments

alpha	boxplot outlier threshold (default 1.5, but can be 3.0 to remove extreme values)

Value

returns an outlier object

Examples

```
# code for outlier removal
out_obj <- outliers_boxplot() # class for outlier analysis
out_obj <- fit(out_obj, iris) # computing boundaries
iris.clean <- transform(out_obj, iris) # returning cleaned dataset

#inspection of cleaned dataset
nrow(iris.clean)

idx <- attr(iris.clean, "idx")
table(idx)
iris.outliers_boxplot <- iris[idx,]
iris.outliers_boxplot
```

outliers_gaussian *outliers_gaussian*

Description

The outliers_gaussian class uses box-plot definition for outliers_gaussian. An outlier is a value that is below than $\overline{x} - 3\sigma_x$ or higher than $\overline{x} + 3\sigma_x$. The class remove outliers_gaussian for numeric attributes.

Usage

```
outliers_gaussian(alpha = 3)
```

Arguments

alpha	gaussian threshold (default 3)

Value

returns an outlier object

Examples

```
# code for outlier removal
out_obj <- outliers_gaussian() # class for outlier analysis
out_obj <- fit(out_obj, iris) # computing boundaries
iris.clean <- transform(out_obj, iris) # returning cleaned dataset

#inspection of cleaned dataset
nrow(iris.clean)

idx <- attr(iris.clean, "idx")
table(idx)
iris.outliers_gaussian <- iris[idx,]
iris.outliers_gaussian
```

---

plot_bar *Plot bar graph*

Description

this function displays a bar graph from a data frame containing x-axis categories using ggplot2.

Usage

```
plot_bar(data, label_x = "", label_y = "", colors = NULL, alpha = 1)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector
alpha	level of transparency

Value

returns a ggplot2::ggplot graphic

Examples

```
#summarizing iris dataset
data <- iris |> dplyr::group_by(Species) |>
dplyr::summarize(Sepal.Length=mean(Sepal.Length))
head(data)

#ploting data
grf <- plot_bar(data, colors="blue")
plot(grf)
```

`plot_boxplot` *Plot boxplot*

Description

this function displays a boxplot graph from a data frame containing x-axis categories and numeric values using ggplot2.

Usage

```
plot_boxplot(data, label_x = "", label_y = "", colors = NULL, barwidth = 0.25)
```

Arguments

`data`	data.frame contain x, value, and variable
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	color vector
`barwidth`	width of bar

Value

returns a ggplot2::ggplot graphic

Examples

```
grf <- plot_boxplot(iris, colors="white")
plot(grf)
```

`plot_boxplot_class` *Boxplot per class*

Description

This function generates boxplots grouped by a specified class label from a data frame containing numeric values using ggplot2.

Usage

```
plot_boxplot_class(
  data,
  class_label,
  label_x = "",
  label_y = "",
  colors = NULL
)
```

Arguments

`data`	data.frame contain x, value, and variable
`class_label`	name of attribute for class label
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	color vector

Value

returns a ggplot2::ggplot graphic

Examples

```
grf <- plot_boxplot_class(iris |> dplyr::select(Sepal.Width, Species),
class = "Species", colors=c("red", "green", "blue"))
plot(grf)
```

plot_density *Plot density*

Description

This function generates a density plot from a data frame containing numeric values using ggplot2. If the data frame has multiple columns, densities can be grouped and plotted.

Usage

```
plot_density(
  data,
  label_x = "",
  label_y = "",
  colors = NULL,
  bin = NULL,
  alpha = 0.25
)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector
bin	bin width for density estimation
alpha	level of transparency

Value

returns a ggplot2::ggplot graphic

Examples

```
grf <- plot_density(iris |> dplyr::select(Sepal.Width), colors="blue")
plot(grf)
```

`plot_density_class` *Plot density per class*

Description

This function generates density plots using ggplot2 grouped by a specified class label from a data frame containing numeric values.

Usage

```
plot_density_class(
  data,
  class_label,
  label_x = "",
  label_y = "",
  colors = NULL,
  bin = NULL,
  alpha = 0.5
)
```

Arguments

`data`	data.frame contain x, value, and variable
`class_label`	name of attribute for class label
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	color vector
`bin`	bin width for density estimation
`alpha`	level of transparency

Value

returns a ggplot2::ggplot graphic

Examples

```
grf <- plot_density_class(iris |> dplyr::select(Sepal.Width, Species),
class = "Species", colors=c("red", "green", "blue"))
plot(grf)
```

plot_groupedbar *Plot grouped bar*

Description

This function generates a grouped bar plot from a given data frame using ggplot2.

Usage

```
plot_groupedbar(data, label_x = "", label_y = "", colors = NULL, alpha = 1)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector
alpha	level of transparency

Value

returns a ggplot2::ggplot graphic

Examples

```
#summarizing iris dataset
data <- iris |> dplyr::group_by(Species) |>
dplyr::summarize(Sepal.Length=mean(Sepal.Length), Sepal.Width=mean(Sepal.Width))
head(data)

#ploting data
grf <- plot_groupedbar(data, colors=c("blue", "red"))
plot(grf)
```

plot_hist *Plot histogram*

Description

This function generates a histogram from a specified data frame using ggplot2.

Usage

```
plot_hist(data, label_x = "", label_y = "", color = "white", alpha = 0.25)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
color	color vector
alpha	transparency level

Value

returns a ggplot2::ggplot graphic

Examples

```
grf <- plot_hist(iris |> dplyr::select(Sepal.Width), color=c("blue"))
plot(grf)
```

---

plot_lollipop *Plot lollipop*

Description

This function creates a lollipop chart using ggplot2.

Usage

```
plot_lollipop(
  data,
  label_x = "",
  label_y = "",
  colors = NULL,
  color_text = "black",
  size_text = 3,
  size_ball = 8,
  alpha_ball = 0.2,
  min_value = 0,
  max_value_gap = 1
)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector
color_text	color of text inside ball

size_text	size of text inside ball
size_ball	size of ball
alpha_ball	transparency of ball
min_value	minimum value
max_value_gap	maximum value gap

Value

returns a ggplot2::ggplot graphic

Examples

```
#summarizing iris dataset
data <- iris |> dplyr::group_by(Species) |>
dplyr::summarize(Sepal.Length=mean(Sepal.Length))
head(data)

#ploting data
grf <- plot_lollipop(data, colors="blue", max_value_gap=0.2)
plot(grf)
```

---

plot_pieplot *Plot pie*

Description

This function creates a pie chart using ggplot2.

Usage

```
plot_pieplot(
  data,
  label_x = "",
  label_y = "",
  colors = NULL,
  textcolor = "white",
  bordercolor = "black"
)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector
textcolor	text color
bordercolor	border color

Value

returns a ggplot2::ggplot graphic

Examples

```
#summarizing iris dataset
data <- iris |> dplyr::group_by(Species) |>
dplyr::summarize(Sepal.Length=mean(Sepal.Length))
head(data)

#ploting data
grf <- plot_pieplot(data, colors=c("red", "green", "blue"))
plot(grf)
```

---

plot_points *Plot points*

Description

This function creates a scatter plot using ggplot2.

Usage

```
plot_points(data, label_x = "", label_y = "", colors = NULL)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector

Value

returns a ggplot2::ggplot graphic

Examples

```
x <- seq(0, 10, 0.25)
data <- data.frame(x, sin=sin(x), cosine=cos(x)+5)
head(data)

grf <- plot_points(data, colors=c("red", "green"))
plot(grf)
```

plot_radar *Plot radar*

Description

This function creates a radar chart using ggplot2.

Usage

```
plot_radar(data, label_x = "", label_y = "", colors = NULL)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector

Value

returns a ggplot2::ggplot graphic

Examples

```
data <- data.frame(name = "Petal.Length", value = mean(iris$Petal.Length))
data <- rbind(data, data.frame(name = "Petal.Width", value = mean(iris$Petal.Width)))
data <- rbind(data, data.frame(name = "Sepal.Length", value = mean(iris$Sepal.Length)))
data <- rbind(data, data.frame(name = "Sepal.Width", value = mean(iris$Sepal.Width)))

grf <- plot_radar(data, colors="red") + ggplot2::ylim(0, NA)
plot(grf)
```

plot_scatter *Scatter graph*

Description

This function creates a scatter plot using ggplot2.

Usage

```
plot_scatter(data, label_x = "", label_y = "", colors = NULL)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector

Value

return a ggplot2::ggplot graphic

Examples

```
grf <- plot_scatter(iris |> dplyr::select(x = Sepal.Length,
value = Sepal.Width, variable = Species),
label_x = "Sepal.Length", label_y = "Sepal.Width",
colors=c("red", "green", "blue"))
plot(grf)
```

---

plot_series *Plot series*

Description

This function creates a time series plot using ggplot2.

Usage

```
plot_series(data, label_x = "", label_y = "", colors = NULL)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector

Value

returns a ggplot2::ggplot graphic

Examples

```
x <- seq(0, 10, 0.25)
data <- data.frame(x, sin=sin(x))
head(data)

grf <- plot_series(data, colors=c("red"))
plot(grf)
```

plot_stackedbar *Plot stacked bar*

Description

this function creates a stacked bar chart using ggplot2.

Usage

```
plot_stackedbar(data, label_x = "", label_y = "", colors = NULL, alpha = 1)
```

Arguments

data	data.frame contain x, value, and variable
label_x	x-axis label
label_y	y-axis label
colors	color vector
alpha	level of transparency

Value

returns a ggplot2::ggplot graphic

Examples

```
#summarizing iris dataset
data <- iris |> dplyr::group_by(Species) |>
dplyr::summarize(Sepal.Length=mean(Sepal.Length), Sepal.Width=mean(Sepal.Width))

#plotting data
grf <- plot_stackedbar(data, colors=c("blue", "red"))
plot(grf)
```

plot_ts *Plot time series chart*

Description

This function plots a time series chart with points and a line using ggplot2.

Usage

```
plot_ts(x = NULL, y, label_x = "", label_y = "", color = "black")
```

Arguments

x	input variable
y	output variable
label_x	x-axis label
label_y	y-axis label
color	color for time series

Value

returns a ggplot2::ggplot graphic

Examples

```
x <- seq(0, 10, 0.25)
y <- sin(x)

grf <- plot_ts(x = x, y = y, color=c("red"))
plot(grf)
```

---

plot_ts_pred *Plot a time series chart with predictions*

Description

This function plots a time series chart with three lines: the original series, the adjusted series, and the predicted series using ggplot2.

Usage

```
plot_ts_pred(
  x = NULL,
  y,
  yadj,
  ypred = NULL,
  label_x = "",
  label_y = "",
  color = "black",
  color_adjust = "blue",
  color_prediction = "green"
)
```

Arguments

x	time index
y	time series
yadj	adjustment of time series
ypred	prediction of the time series
label_x	x-axis title
label_y	y-axis title
color	color for the time series
color_adjust	color for the adjusted values
color_prediction	color for the predictions

Value

returns a ggplot2::ggplot graphic

Examples

```
x <- base::seq(0, 10, 0.25)
yvalues <- sin(x) + rnorm(41,0,0.1)
adjust <- sin(x[1:35])
prediction <- sin(x[36:41])
grf <- plot_ts_pred(y=yvalues, yadj=adjust, ypre=prediction)
plot(grf)
```

---

predictor *DAL Predict*

Description

Ancestor class for regression and classification It provides basis for fit and predict methods. Besides, action method proxies to predict.

An example of learner is a decision tree (cla_dtree)

Usage

```
predictor()
```

Value

returns a predictor object

Examples

```
#See ?cla_dtree for a classification example using a decision tree
```

regression *Regression*

Description

Ancestor class for regression problems. This ancestor class is used to define and manage the target attribute for regression tasks.

Usage

```
regression(attribute)
```

Arguments

attribute	attribute target to model building

Value

returns a regression object

Examples

```
#See ?reg_dtree for a regression example using a decision tree
```

reg_dtree *Decision Tree for regression*

Description

Creates a regression object that uses the Decision Tree method for regression It wraps the tree library.

Usage

```
reg_dtree(attribute)
```

Arguments

attribute	attribute target to model building.

Value

returns a decision tree regression object

Examples

```
data(Boston)
model <- reg_dtree("medv")

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, Boston)
train <- sr$train
test <- sr$test

model <- fit(model, train)

test_prediction <- predict(model, test)
test_predictand <- test[,"medv"]
test_eval <- evaluate(model, test_predictand, test_prediction)
test_eval$metrics
```

---

reg_knn *knn regression*

Description

Creates a regression object that uses the K-Nearest Neighbors (knn) method for regression

Usage

```
reg_knn(attribute, k)
```

Arguments

attribute	attribute target to model building
k	number of k neighbors

Value

returns a knn regression object

Examples

```
data(Boston)
model <- reg_knn("medv", k=3)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, Boston)
train <- sr$train
test <- sr$test

model <- fit(model, train)
```

```
test_prediction <- predict(model, test)
test_predictand <- test[,"medv"]
test_eval <- evaluate(model, test_predictand, test_prediction)
test_eval$metrics
```

---

reg_mlp *MLP for regression*

Description

Creates a regression object that uses the Multi-Layer Perceptron (MLP) method. It wraps the nnet library.

Usage

```
reg_mlp(attribute, size = NULL, decay = 0.05, maxit = 1000)
```

Arguments

attribute	attribute target to model building
size	number of neurons in hidden layers
decay	decay learning rate
maxit	number of maximum iterations for training

Value

returns a object of class reg_mlp

Examples

```
data(Boston)
model <- reg_mlp("medv", size=5, decay=0.54)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, Boston)
train <- sr$train
test <- sr$test

model <- fit(model, train)

test_prediction <- predict(model, test)
test_predictand <- test[,"medv"]
test_eval <- evaluate(model, test_predictand, test_prediction)
test_eval$metrics
```

reg_rf *Random Forest for regression*

Description

Creates a regression object that uses the Random Forest method. It wraps the randomForest library.

Usage

```
reg_rf(attribute, nodesize = 1, ntree = 10, mtry = NULL)
```

Arguments

attribute	attribute target to model building
nodesize	node size
ntree	number of trees
mtry	number of attributes to build tree

Value

returns an object of class reg_rfobj

Examples

```
data(Boston)
model <- reg_rf("medv", ntree=10)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, Boston)
train <- sr$train
test <- sr$test

model <- fit(model, train)

test_prediction <- predict(model, test)
test_predictand <- test[,"medv"]
test_eval <- evaluate(model, test_predictand, test_prediction)
test_eval$metrics
```

reg_svm *SVM for regression*

Description

Creates a regression object that uses the Support Vector Machine (SVM) method for regression It wraps the e1071 and svm library.

Usage

```
reg_svm(attribute, epsilon = 0.1, cost = 10, kernel = "radial")
```

Arguments

attribute	attribute target to model building
epsilon	parameter that controls the width of the margin around the separating hyperplane
cost	parameter that controls the trade-off between having a wide margin and correctly classifying training data points
kernel	the type of kernel function to be used in the SVM algorithm (linear, radial, polynomial, sigmoid)

Value

returns a SVM regression object

Examples

```
data(Boston)
model <- reg_svm("medv", epsilon=0.2,cost=40.000)

# preparing dataset for random sampling
sr <- sample_random()
sr <- train_test(sr, Boston)
train <- sr$train
test <- sr$test

model <- fit(model, train)

test_prediction <- predict(model, test)
test_predictand <- test[,"medv"]
test_eval <- evaluate(model, test_predictand, test_prediction)
test_eval$metrics
```

reg_tune *Regression Tune*

Description

Creates an object for tuning regression models

Usage

```
reg_tune(base_model, folds = 10)
```

Arguments

base_model	base model for tuning
folds	number of folds for cross-validation

Value

returns a reg_tune object.

Examples

```
# preparing dataset for random sampling
data(Boston)
sr <- sample_random()
sr <- train_test(sr, Boston)
train <- sr$train
test <- sr$test

# hyper parameter setup
tune <- reg_tune(reg_mlp("medv"))
ranges <- list(size=c(3), decay=c(0.1,0.5))

# hyper parameter optimization
model <- fit(tune, train, ranges)

test_prediction <- predict(model, test)
test_predictand <- test[,"medv"]
test_eval <- evaluate(model, test_predictand, test_prediction)
test_eval$metrics
```

sample_random *Sample Random*

Description

The sample_random function in R is used to generate a random sample of specified size from a given data set.

Usage

```
sample_random()
```

Value

returns an object of class 'sample_random

Examples

```
#using random sampling
sample <- sample_random()
tt <- train_test(sample, iris)

# distribution of train
table(tt$train$Species)

# preparing dataset into four folds
folds <- k_fold(sample, iris, 4)

# distribution of folds
tbl <- NULL
for (f in folds) {
 tbl <- rbind(tbl, table(f$Species))
}
head(tbl)
```

sample_stratified *Stratified Random Sampling*

Description

The sample_stratified function in R is used to generate a stratified random sample from a given dataset. Stratified sampling is a statistical method that is used when the population is divided into non-overlapping subgroups or strata, and a sample is selected from each stratum to represent the entire population. In stratified sampling, the sample is selected in such a way that it is representative of the entire population and the variability within each stratum is minimized.

Usage

```
sample_stratified(attribute)
```

Arguments

attribute	attribute target to model building

Value

returns an object of class sample_stratified

Examples

```
#using stratified sampling
sample <- sample_stratified("Species")
tt <- train_test(sample, iris)

# distribution of train
table(tt$train$Species)

# preparing dataset into four folds
folds <- k_fold(sample, iris, 4)

# distribution of folds
tbl <- NULL
for (f in folds) {
 tbl <- rbind(tbl, table(f$Species))
}
head(tbl)
```

---

select_hyper *Selection hyper parameters*

Description

Selects the optimal hyperparameters from a dataset resulting from k-fold cross-validation

Usage

```
select_hyper(obj, hyperparameters)
```

Arguments

obj	the object or model used for hyperparameter selection.
hyperparameters	data set with hyper parameters and quality measure from execution

Value

returns the index of selected hyper parameter

select_hyper.cla_tune *selection of hyperparameters*

Description

Selects the optimal hyperparameter by maximizing the average classification metric. It wraps dplyr library.

Usage

```
## S3 method for class 'cla_tune'
select_hyper(obj, hyperparameters)
```

Arguments

obj	an object representing the model or tuning process
hyperparameters	a dataframe with columns key (hyperparameter configuration) and metric (classification metric)

Value

returns a optimized key number of hyperparameters

set_params *Assign parameters*

Description

set_params function assigns all parameters to the attributes presented in the object.

Usage

```
set_params(obj, params)
```

Arguments

obj	object of class dal_base
params	parameters to set obj

Value

returns an object with parameters set

Examples

```
obj <- set_params(dal_base(), list(x = 0))
```

set_params.default *Default Assign parameters*

Description

Default method for set_params which returns the object unchanged

Usage

```
## Default S3 method:
set_params(obj, params)
```

Arguments

obj	object
params	parameters

Value

returns the object unchanged

smoothing *Smoothing*

Description

Smoothing is a statistical technique used to reduce the noise in a signal or a dataset by removing the high-frequency components. The smoothing level is associated with the number of bins used. There are alternative methods to establish the smoothing: equal interval, equal frequency, and clustering.

Usage

```
smoothing(n)
```

Arguments

n	number of bins

Value

returns an object of class smoothing

Examples

```
data(iris)
obj <- smoothing_inter(n = 2)
obj <- fit(obj, iris$Sepal.Length)
sl.bi <- transform(obj, iris$Sepal.Length)
table(sl.bi)
obj$interval

entro <- evaluate(obj, as.factor(names(sl.bi)), iris$Species)
entro$entropy
```

---

smoothing_cluster *Smoothing by cluster*

Description

Uses clustering method to perform data smoothing. The input vector is divided into clusters using the k-means algorithm. The mean of each cluster is then calculated and used as the smoothed value for all observations within that cluster.

Usage

```
smoothing_cluster(n)
```

Arguments

n	number of bins

Value

returns an object of class `smoothing_cluster`

Examples

```
data(iris)
obj <- smoothing_cluster(n = 2)
obj <- fit(obj, iris$Sepal.Length)
sl.bi <- transform(obj, iris$Sepal.Length)
table(sl.bi)
obj$interval

entro <- evaluate(obj, as.factor(names(sl.bi)), iris$Species)
entro$entropy
```

smoothing_freq *Smoothing by Freq*

Description

The 'smoothing_freq' function is used to smooth a given time series data by aggregating observations within a fixed frequency.

Usage

```
smoothing_freq(n)
```

Arguments

n	number of bins

Value

returns an object of class smoothing_freq

Examples

```
data(iris)
obj <- smoothing_freq(n = 2)
obj <- fit(obj, iris$Sepal.Length)
sl.bi <- transform(obj, iris$Sepal.Length)
table(sl.bi)
obj$interval

entro <- evaluate(obj, as.factor(names(sl.bi)), iris$Species)
entro$entropy
```

smoothing_inter *Smoothing by interval*

Description

The "smoothing by interval" function is used to apply a smoothing technique to a vector or time series data using a moving window approach.

Usage

```
smoothing_inter(n)
```

Arguments

n	number of bins

Value

returns an object of class `smoothing_inter`

Examples

```
data(iris)
obj <- smoothing_inter(n = 2)
obj <- fit(obj, iris$Sepal.Length)
sl.bi <- transform(obj, iris$Sepal.Length)
table(sl.bi)
obj$interval

entro <- evaluate(obj, as.factor(names(sl.bi)), iris$Species)
entro$entropy
```

`train_test` *Train-Test Partition*

Description

Partitions a dataset into training and test sets using a specified sampling method

Usage

```
train_test(obj, data, perc = 0.8, ...)
```

Arguments

`obj`	an object of a class that supports the `train_test` method
`data`	dataset to be partitioned
`perc`	a numeric value between 0 and 1 specifying the proportion of data to be used for training
`...`	additional optional arguments passed to specific methods.

Value

returns an list with two elements:

- train: A data frame containing the training set
- test: A data frame containing the test set

Examples

```
#using random sampling
sample <- sample_random()
tt <- train_test(sample, iris)

# distribution of train
table(tt$train$Species)
```

`train_test_from_folds` *k-fold training and test partition object*

Description

Splits a dataset into training and test sets based on k-fold cross-validation. The function takes a list of data partitions (folds) and a specified fold index k. It returns the data corresponding to the k-th fold as the test set, and combines all other folds to form the training set.

Usage

```
train_test_from_folds(folds, k)
```

Arguments

`folds`	data partitioned into folds
`k`	k-fold for test set, all reminder for training set

Value

returns a list with two elements:

- train: A data frame containing the combined data from all folds except the k-th fold, used as the training set.
- test: A data frame corresponding to the k-th fold, used as the test set.

Examples

```
# Create k-fold partitions of a dataset (e.g., iris)
folds <- k_fold(sample_random(), iris, k = 5)

# Use the first fold as the test set and combine the remaining folds for the training set
train_test_split <- train_test_from_folds(folds, k = 1)

# Display the training set
head(train_test_split$train)

# Display the test set
head(train_test_split$test)
```

transform *Transform*

Description

Defines a transformation method.

Usage

```
transform(obj, ...)
```

Arguments

obj	a dal_transform object.
...	optional arguments.

Value

returns a transformed data.

Examples

```
#See ?minmax for an example of transformation
```

zscore *Z-score normalization*

Description

Scale data using z-score normalization.
$zscore = (x - mean(x))/sd(x)$

Usage

```
zscore(nmean = 0, nsd = 1)
```

Arguments

nmean	new mean for normalized data
nsd	new standard deviation for normalized data

Value

returns the z-score transformation object

Examples

```
data(iris)
head(iris)

trans <- zscore()
trans <- fit(trans, iris)
tiris <- transform(trans, iris)
head(tiris)

itiris <- inverse_transform(trans, tiris)
head(itiris)
```

Index