Package: daltoolbox (via r-universe)

November 23, 2026

Title Leveraging Experiment Lines to Data Analytics

Version 1.4.707

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)

Imports methods, FNN, arules, class, dbscan, dplyr, e1071, ggplot2, mclust, nnet, randomForest, reshape, tree

Suggests adabag, glmnet, ipred, xgboost, rpart, MASS

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Eduardo Ogasawara [aut, ths, cre] (ORCID:
<https://orcid.org/0000-0002-0466-0626>), Ana Carolina Sa
[aut], Antonio Castro [aut], Caio Santos [aut], Diego Carvalho
[ctb], Diego Salles [aut], Eduardo Bezerra [ctb], Esther
Pacitti [ctb], Fabio Porto [ctb], Janio Lima [aut], Lucas
Tavares [aut], Rafaelli Coutinho [ctb], Rebecca Salles [aut],
Vinicius Saidy [aut], CEFET/RJ [cph]

Maintainer Eduardo Ogasawara <eogasawara@ieee.org>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-23 05:10:15 UTC

RemoteUrl https://github.com/cran/daltoolbox

RemoteRef HEAD

RemoteSha 2b10a6725836b915074dd011e93c2bf4aeb74afa

Contents

action 5
action.dal_transform 5
adjust_class_label 6
adjust_data.frame 7
adjust_factor 7
adjust_matrix 8
aggregation 8
autoenc_base_e 9
autoenc_base_ed 10
bal_oversampling 11
bal_subsampling 12
Boston 13
categ_mapping 14
cla_bagging 15
cla_boosting 15
cla_dtree 16
cla_glm 17
cla_glmnet 18
cla_knn 19
cla_majority 20
cla_mlp 21
cla_multinom 22
cla_nb 23
cla_rf 24
cla_rpart 25
cla_svm 26
cla_tune 27
cla_xgboost 28
classification 29
clu_tune 30

cluster . . . 30
cluster_cmeans . . . 31
cluster_dbscan . . . 32
cluster_gmm . . . 34
cluster_hclust . . . 35
cluster_kmeans . . . 36
cluster_pam . . . 37
clusterer . . . 39
cluutils . . . 40
dal_base . . . 41
dal_learner . . . 41
dal_transform . . . 42
dal_tune . . . 42
data_sample . . . 43
Diabetes . . . 44
discover . . . 45
dt_pca . . . 45
evaluate . . . 46
feature_generation . . . 47
feature_selection_corr . . . 48
feature_selection_fss . . . 49
feature_selection_info_gain . . . 50
feature_selection_lasso . . . 51
feature_selection_relief . . . 52
feature_selection_stepwise . . . 53
fit . . . 54
fit.cla_tune . . . 55
fit.cluster_dbscan . . . 55
fit_curvature_max . . . 56
fit_curvature_min . . . 56
hierarchy_cut . . . 57
imputation_predictive . . . 58
imputation_simple . . . 59
imputation_tree . . . 60
inverse_transform . . . 61
k_fold . . . 61
minmax . . . 62
na_removal . . . 63
outliers_boxplot . . . 64
outliers_gaussian . . . 65
pat_apriori . . . 66
pat_confidence_threshold . . . 67
pat_dara . . . 68
pat_filter_rules . . . 70
pat_prefix_span . . . 70
pat_rule_filter_dara . . . 72
pat_rule_filter_interest . . . 73
pat_rule_filter_none . . . 74

pat_rules_tidy . . . 75
pat_support_threshold . . . 75
pattern_miner . . . 76
patutils . . . 77
plot_bar . . . 78
plot_boxplot . . . 79
plot_boxplot_class . . . 80
plot_correlation . . . 81
plot_dendrogram . . . 82
plot_density . . . 83
plot_density_class . . . 84
plot_groupedbar . . . 85
plot_hist . . . 86
plot_lollipop . . . 87
plot_parallel . . . 88
plot_pieplot . . . 89
plot_pixel . . . 90
plot_points . . . 91
plot_radar . . . 92
plot_scatter . . . 93
plot_series . . . 94
plot_stackedbar . . . 95
plot_ts . . . 96
plot_ts_pred . . . 97
predictor . . . 98
predictor_prepare_fit . . . 98
reg_dtree . . . 99
reg_knn . . . 100
reg_lm . . . 101
reg_mlp . . . 102
reg_rf . . . 103
reg_svm . . . 104
reg_tune . . . 105
regression . . . 106
sample_balance . . . 107
sample_groups . . . 108
sample_random . . . 108
sample_simple . . . 109
sample_stratified . . . 110
select_hyper . . . 111
select_hyper.cla_tune . . . 112
set_params . . . 112
set_params.default . . . 113
smoothing . . . 113
smoothing_cluster . . . 114
smoothing_freq . . . 115
smoothing_inter . . . 116
smoothing_quantization . . . 116

train_test 117
train_test_from_folds 118
transform 119
transform.pat_rule_filter 120
zscore 120

Index **122**

`action` *Action*

Description

Generic to apply the object to data (e.g., `predict`, `transform`).

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

Default `action()` implementation that proxies to `transform()` for transforms.

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

One-hot encode a factor vector into a matrix of indicator columns.

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

Details

Values are mapped to valTrue/valFalse (default 1/0). The resulting matrix has column names equal to levels(x).

Value

returns an adjusted categorical mapping

adjust_data.frame *Adjust to data frame*

Description

Coerce an object to `data.frame` if needed (useful for S3 methods in this package).

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

Convert a vector to a factor with specified internal levels (`ilevels`) and labels (`slevels`).

Usage

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

Arguments

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

Details

Numeric vectors are first converted to factors with `ilevels` as the level order, then relabeled to `slevels`.

Value

returns an adjusted factor

---

adjust_matrix *Adjust to matrix*

Description

Coerce an object to `matrix` if needed (useful before algorithms that expect matrices).

Usage

```
adjust_matrix(data)
```

Arguments

`data`	dataset

Value

returns an adjusted matrix

Examples

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

---

aggregation *Aggregation by groups*

Description

Aggregate data by a grouping attribute using named expressions.

Usage

```
aggregation(group, ...)
```

Arguments

`group`	grouping column name (string)
`...`	named expressions evaluated per group

Value

returns an object of class `aggregation`

References

Han, J., Kamber, M. and Pei, J. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann. (Data aggregation)

Examples

```
data(iris)
agg <- aggregation(
 "Species",
 mean_sepal = mean(Sepal.Length),
 sd_sepal = sd(Sepal.Length),
 n = n()
)
iris_agg <- transform(agg, iris)
iris_agg
```

---

autoenc_base_e *Autoencoder base (encoder)*

Description

Base class for encoder-only autoencoders. Intended to be subclassed by concrete implementations that learn a lower-dimensional latent representation.

Usage

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

Arguments

`input_size`	dimensionality of the input vector
`encoding_size`	dimensionality of the latent (encoded) vector

Details

This base does not train or transform by itself (identity). Implementations should override `fit()` to learn parameters and `transform()` to output the encoded representation.

Value

returns an `autoenc_base_e` object

References

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. Science.

Examples

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

`autoenc_base_ed` *Autoencoder base (encoder + decoder)*

Description

Base class for autoencoders that both encode and decode. Intended to be subclassed by concrete implementations that learn to compress and reconstruct inputs.

Usage

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

Arguments

`input_size`	dimensionality of the input vector
`encoding_size`	dimensionality of the latent (encoded) vector

Details

This base does not train or transform by itself (identity). Implementations should override `fit()` to learn parameters and `transform()` to perform encode+decode.

Value

returns an `autoenc_base_ed` object

References

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. Science.

Examples

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

`bal_oversampling` *SMOTE-family class oversampling*

Description

Balance class distributions by generating synthetic minority examples with the SMOTE family (classic SMOTE, Borderline-SMOTE, ADASYN) or by random replication.

Usage

```
bal_oversampling(
  attribute,
  method = c("smote", "borderline", "adasyn", "random"),
  k = 5,
  m = 10,
  seed = NULL
)
```

Arguments

`attribute`	target class attribute name
`method`	oversampling strategy: `"smote"`, `"borderline"`, `"adasyn"` or `"random"`
`k`	number of nearest neighbors used to interpolate synthetic samples
`m`	number of neighbors used to assess the danger/difficulty of a minority instance (borderline/adasyn)
`seed`	optional seed for reproducibility

Details

All synthetic variants interpolate a base minority instance toward one of its `k` nearest minority neighbors; nominal features are set by majority vote among those neighbors (SMOTE-NC) instead of a coin flip. `"borderline"` synthesizes only from *danger* instances (those whose `m` nearest neighbors are mostly of other classes), concentrating new points near the decision boundary; `"adasyn"` allocates more synthetic samples to harder-to-learn minority instances.

Value

returns an object of class `bal_oversampling`

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16:321-357. Han, H., Wang, W.-Y., & Mao, B.-H. (2005). Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In ICIC 2005, 878-887. He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. In IJCNN 2008, 1322-1328.

Examples

```
data(iris)
iris_imb <- iris[c(1:50, 51:71, 101:111), ]
bal <- bal_oversampling("Species", method = "smote")
iris_bal <- transform(bal, iris_imb)
table(iris_bal$Species)
```

`bal_subsampling` *Random class undersampling*

Description

Balance class distributions by randomly reducing all classes to the minority count.

Usage

```
bal_subsampling(attribute)
```

Arguments

`attribute`	target class attribute name

Value

returns an object of class `bal_subsampling`

References

He, H. and Garcia, E. A. (2009). Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering, 21(9):1263-1284.

Examples

```
data(iris)
iris_imb <- iris[c(1:50, 51:71, 101:111), ]
bal <- bal_subsampling("Species")
iris_bal <- transform(bal, iris_imb)
table(iris_bal$Species)
```

`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 (one-hot encoding)*

Description

Convert a factor column into dummy variables (one-hot encoding) using `model.matrix` without intercept. Each level becomes a separate binary column named `<attribute>_<level>`.

Usage

```
categ_mapping(attribute)
```

Arguments

`attribute`	attribute to be categorized.

Details

The remaining columns are kept and the original attribute is replaced in place by its one-hot dummy columns (appended at the end). Keeping the other columns makes the transform safe to use inside a preprocessing pipeline.

Value

returns the input data frame with the attribute replaced by one binary column per category (`<attribute>_<level>`); all other columns are preserved.

References

Kuhn, M. and Johnson, K. (2013). Applied Predictive Modeling. Springer. Han, J., Kamber, M. and Pei, J. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann.

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)
```

cla_bagging *Bagging (ipred)*

Description

Bagging classifier using `ipred::bagging`.

Usage

```
cla_bagging(attribute, nbagg = 25)
```

Arguments

`attribute`	target attribute name
`nbagg`	number of bootstrap aggregations

Value

returns a `cla_bagging` object

References

Breiman, L. (1996). Bagging predictors. Machine Learning, 24(2):123-140.

Examples

```
if (requireNamespace("ipred", quietly = TRUE)) {
 data(iris)
 model <- cla_bagging("Species", nbagg = 25)
 model <- fit(model, iris)
 pred <- predict(model, iris)
 eval <- evaluate(model, adjust_class_label(iris$Species), pred)
 eval$metrics
}
```

cla_boosting *Boosting (adabag)*

Description

Boosting classifier using `adabag::boosting`.

Usage

```
cla_boosting(attribute, mfinal = 50)
```

Arguments

attribute	target attribute name
mfinal	number of boosting iterations

Value

returns a cla_boosting object

References

Freund, Y. and Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences, 55(1):119-139.

Examples

```
if (requireNamespace("adabag", quietly = TRUE)) {
 data(iris)
 model <- cla_boosting("Species", mfinal = 10)
 model <- fit(model, iris)
 pred <- predict(model, iris)
 eval <- evaluate(model, adjust_class_label(iris$Species), pred)
 eval$metrics
}
```

---

cla_dtree *Decision Tree for classification*

Description

Univariate decision tree for classification using recursive partitioning. This wrapper uses the tree package.

Usage

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

Arguments

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

Details

Decision trees split the feature space by maximizing node purity (e.g., Gini/entropy), yielding a human-readable set of rules. They are fast and interpretable, and often used as base learners in ensembles.

Value

returns a classification object

References

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). Classification and Regression Trees. Wadsworth.

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_glm *Logistic regression (GLM)*

Description

Logistic regression classifier using `stats::glm` with binomial family.

Usage

```
cla_glm(attribute, positive, features = NULL, threshold = 0.5)
```

Arguments

`attribute`	target attribute name
`positive`	positive class label
`features`	optional vector of feature names (default: all except attribute)
`threshold`	probability threshold for positive class

Value

returns a `cla_glm` object

References

Nelder, J. A. and Wedderburn, R. W. M. (1972). Generalized Linear Models. Journal of the Royal Statistical Society, Series A, 135(3):370-384. Hosmer, D. W. and Lemeshow, S. (2000). Applied Logistic Regression (2nd ed.). Wiley.

Examples

```
data(iris)
iris_bin <- iris
iris_bin$IsVersicolor <- factor(ifelse(
 iris_bin$Species == "versicolor",
 "versicolor",
 "not_versicolor"
))
model <- cla_glm("IsVersicolor", positive = "versicolor")
model <- suppressWarnings(fit(model, iris_bin))
pred <- predict(model, iris_bin)
eval <- evaluate(model, adjust_class_label(iris_bin$IsVersicolor), pred)
eval$metrics
```

---

cla_glmnet *LASSO logistic regression (glmnet)*

Description

Logistic regression with L1 penalty using `glmnet::cv.glmnet`.

Usage

```
cla_glmnet(attribute, lambda = c("lambda.min", "lambda.1se"))
```

Arguments

`attribute`	target attribute name (binary)
`lambda`	which lambda to use ("lambda.min" or "lambda.1se")

Value

returns a `cla_glmnet` object

References

Friedman, J., Hastie, T. and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1):1-22. Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society, Series B, 67(2):301-320.

Examples

```
if (requireNamespace("glmnet", quietly = TRUE)) {
 data(iris)
 iris_bin <- iris
 iris_bin$IsVersicolor <- factor(ifelse(
   iris_bin$Species == "versicolor",
   "versicolor",
   "not_versicolor"
 ))
 model <- cla_glmnet("IsVersicolor")
 model <- fit(model, iris_bin)
 pred <- predict(model, iris_bin)
 eval <- evaluate(model, adjust_class_label(iris_bin$IsVersicolor), pred)
 eval$metrics
}
```

---

cla_knn *K-Nearest Neighbors (KNN) Classification*

Description

Classification by majority vote among the k nearest neighbors. Uses `class::knn`.

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.

Details

KNN is a simple, non-parametric method. Choice of k trades bias/variance; distance metric is Euclidean by default.

Value

returns a knn object.

References

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. IEEE Trans. Info. Theory.

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 Baseline Classifier*

Description

Trivial classifier that always predicts the most frequent class observed in the training data.

Usage

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

Arguments

attribute	Name of the target attribute to predict.
slevels	Possible class labels for the target classification problem.

Details

Useful as a minimum reference point. If a stronger classifier does not outperform this learner, it is often worth revisiting the predictors, the sampling strategy, or the evaluation protocol before tuning models.

Value

A classification object of class `cla_majority`.

References

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). Data Mining: Practical Machine Learning Tools and Techniques (4th ed.). Morgan Kaufmann.

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

Multi-Layer Perceptron classifier using `nnet::nnet` (single hidden layer).

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

Details

Uses softmax output with one-hot targets from `adjust_class_label`. `size` controls hidden units and `decay` applies L2 regularization. Features should be scaled.

Value

returns a classification object

References

Rumelhart, D., Hinton, G., Williams, R. (1986). Learning representations by back-propagating errors. Bishop, C. M. (1995). Neural Networks for Pattern Recognition.

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_multinom *Multinomial logistic regression*

Description

Multiclass classification using `nnet::multinom`.

Usage

```
cla_multinom(attribute, features = NULL)
```

Arguments

`attribute`	target attribute name
`features`	optional vector of feature names (default: all except attribute)

Value

returns a `cla_multinom` object

References

Agresti, A. (2002). Categorical Data Analysis (2nd ed.). Wiley. Venables, W. N. and Ripley, B. D. (2002). Modern Applied Statistics with S (4th ed.). Springer.

Examples

```
data(iris)
model <- cla_multinom("Species")
model <- fit(model, iris)
pred <- predict(model, iris)
eval <- evaluate(model, adjust_class_label(iris$Species), pred)
eval$metrics
```

`cla_nb` *Naive Bayes Classifier*

Description

Naive Bayes classification using `e1071::naiveBayes`.

Usage

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

Arguments

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

Details

Assumes conditional independence of features given the class label, enabling fast probabilistic classification.

Value

returns a classification object.

References

Mitchell, T. (1997). Machine Learning. McGraw-Hill. (Naive Bayes)

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

Ensemble classifier of decision trees using `randomForest::randomForest`.

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

Details

Combines many decorrelated trees to reduce variance. Key hyperparameters: `ntree`, `mtry`, `nodesize`.

Value

returns a classification object

References

Breiman, L. (2001). Random Forests. Machine Learning 45(1):5–32. Liaw, A. and Wiener, M. (2002). Classification and Regression by randomForest. R News.

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_rpart *CART (rpart)*

Description

Classification tree using `rpart::rpart`.

Usage

```
cla_rpart(attribute)
```

Arguments

`attribute`	target attribute name

Value

returns a `cla_rpart` object

References

Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984). Classification and Regression Trees. Wadsworth. Therneau, T. M. and Atkinson, E. J. (1997). An Introduction to Recursive Partitioning Using the rpart Routines.

Examples

```
if (requireNamespace("rpart", quietly = TRUE)) {
 data(iris)
 model <- cla_rpart("Species")
 model <- fit(model, iris)
 pred <- predict(model, iris)
 eval <- evaluate(model, adjust_class_label(iris$Species), pred)
 eval$metrics
}
```

`cla_svm` *SVM for classification*

Description

Support Vector Machines (SVM) for classification using `e1071::svm`.

Usage

```
cla_svm(
  attribute,
  slevels,
  epsilon = 0.1,
  cost = 10,
  kernel = c("radial", "linear", "polynomial", "sigmoid")
)
```

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)

Details

SVMs find a maximum-margin hyperplane in a transformed feature space defined by a kernel (linear, radial, polynomial, sigmoid). The `cost` controls the trade-off between margin width and training error; `epsilon` affects stopping; `kernel` sets the feature map.

Value

returns a SVM classification object

References

Cortes, C. and Vapnik, V. (1995). Support-Vector Networks. Machine Learning 20(3):273–297. Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines.

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 tuning (k-fold CV)*

Description

Tune hyperparameters of a base classifier via k-fold cross-validation using a chosen metric.

Usage

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

Arguments

base_model	base model for tuning
folds	number of folds for cross-validation
ranges	a list of hyperparameter ranges to explore
metric	metric used to optimize

Value

returns a cla_tune object

References

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.

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)

# 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
```

`cla_xgboost` *XGBoost*

Description

Gradient boosting classifier using `xgboost`.

Usage

```
cla_xgboost(attribute, params = list(), nrounds = 20)
```

Arguments

`attribute`	target attribute name
`params`	list of xgboost parameters
`nrounds`	number of boosting rounds

Value

returns a `cla_xgboost` object

References

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 29(5):1189-1232. Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

Examples

```
## Not run:
data(iris)
model <- cla_xgboost("Species")
model <- fit(model, iris)
pred <- predict(model, iris)
eval <- evaluate(model, adjust_class_label(iris$Species), pred)
eval$metrics

## End(Not run)
```

`classification` *Classification base class*

Description

Ancestor class for classification models providing common fields (target attribute and levels) and evaluation helpers.

Usage

```
classification(attribute, slevels = NULL)
```

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
```

clu_tune *Clustering tuning (intrinsic metric)*

Description

Tune clustering hyperparameters by evaluating an intrinsic metric over a parameter grid and selecting the elbow (max curvature).

Usage

```
clu_tune(base_model, folds = 10, ranges = NULL)
```

Arguments

base_model	base model for tuning
folds	number of folds for cross-validation
ranges	a list of hyperparameter ranges to explore

Value

returns a clu_tune object.

References

Satopaa, V. et al. (2011). Finding a “Kneedle” in a Haystack.

Examples

```
data(iris)

# fit model
model <- clu_tune(cluster_kmeans(k = 2), ranges = list(k = 2:10))

model <- fit(model, iris[,1:4])
model$k
```

cluster *Cluster*

Description

Generic for clustering methods

Usage

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

Arguments

obj	a clusterer object
...	optional arguments

Value

clustered data

Examples

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

---

cluster_cmeans *Fuzzy c-means*

Description

Fuzzy c-means clustering.

Usage

```
cluster_cmeans(
  centers = 2,
  m = 2,
  iter = 100,
  dist = "euclidean",
  tol = 1e-06,
  nstart = 1,
  seed = NULL
)
```

Arguments

centers	number of clusters
m	fuzziness parameter (m > 1)
iter	maximum number of iterations
dist	distance method: "euclidean" or "manhattan"
tol	convergence tolerance on the membership change
nstart	number of restarts; the lowest-error run is kept (default 1)
seed	optional seed for reproducibility

Details

Produces soft membership for each cluster. The hard assignment is returned by `cluster()`. Membership degrees are returned in the `membership` attribute. The algorithm alternates between updating cluster centers as membership-weighted means and recomputing memberships from the distances to those centers, until convergence. With `nstart > 1` it runs several times from different random starts and keeps the run with the lowest within-cluster fuzzy error, reducing sensitivity to the random start.

The base `clusterer()` uses `wcss` as a generic default, but `cluster_cmeans()` specializes that choice because fuzzy clustering carries both a hard partition view and a soft membership structure.

Default evaluation in `cluster_cmeans()` is:

- main metric: `metric_silhouette()`
- internal evaluation: `silhouette`, `withinerror`
- external evaluation: `entropy`, `purity`, `adjusted_rand_index`

The generic `wcss` fallback is not used as the main metric because fuzzy c-means is better summarized by a partition quality index plus a criterion that respects the model's own within-cluster objective.

Value

returns a fuzzy clustering object.

References

Bezdek, J. C. (1981). Pattern Recognition with Fuzzy Objective Function Algorithms. Dunn, J. C. (1973). A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis.

Examples

```
data(iris)
model <- cluster_cmeans(centers = 3, m = 2)
model <- fit(model, iris[,1:4])
clu <- cluster(model, iris[,1:4])
table(clu)
```

---

cluster_dbscan *DBSCAN*

Description

Density-Based Spatial Clustering of Applications with Noise using `dbscan::dbscan`.

Usage

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

Arguments

minPts	minimum number of points
eps	distance value

Details

Discovers clusters as dense regions separated by sparse areas. Hyperparameters are eps (neighborhood radius) and minPts (minimum points). If eps is missing, it is estimated from the kNN distance curve elbow.

The base clusterer() uses wcss as a generic default, but cluster_dbscan() specializes that choice because DBSCAN does not force a centroid-based partition and can label observations as noise.

Default evaluation in cluster_dbscan() is:

- main metric: metric_noise_points()
- internal evaluation: noise_points
- external evaluation: entropy, purity, adjusted_rand_index

The general wcss fallback is not used as the main metric because it ignores the central modeling choice of DBSCAN: distinguishing dense clusters from sparse or noisy observations.

Value

returns a dbscan object

References

Ester, M., Kriegel, H.-P., Sander, J., Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Zhao, Y., & Karypis, G. (2001). Criterion functions for document clustering.

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_gmm` *Gaussian mixture model clustering (GMM)*

Description

Model-based clustering using `mclust::Mclust`.

Usage

```
cluster_gmm(G = NULL, modelNames = NULL)
```

Arguments

`G`	number of mixture components (clusters). If NULL, `Mclust` chooses.
`modelNames`	optional character vector of model names passed to `Mclust`.

Details

Fits a Gaussian mixture model and returns the MAP classification. The fitted model is stored in `obj$model`. Requires the `mclust` package.

The base `clusterer()` uses `wcss` as a generic default, but `cluster_gmm()` specializes that choice because GMM is a probabilistic model with an explicit likelihood function.

Default evaluation in `cluster_gmm()` is:

- main metric: `metric_loglik()`
- internal evaluation: `loglik`
- external evaluation: `entropy`, `purity`, `adjusted_rand_index`

The general `wcss` fallback is not used as the main metric because it does not represent the fit criterion optimized by the mixture model.

Value

returns a GMM clustering object.

References

Fraley, C., & Raftery, A. E. (2002). Model-based clustering. *JASA*. Banfield, J. D., & Raftery, A. E. (1993). Model-based Gaussian and non-Gaussian clustering.

Examples

```
if (requireNamespace("mclust", quietly = TRUE)) {
 data(iris)
 model <- cluster_gmm(G = 3)
 model <- fit(model, iris[,1:4])
 clu <- cluster(model, iris[,1:4])
 table(clu)
}
```

`cluster_hclust` *Hierarchical clustering*

Description

Agglomerative hierarchical clustering using `stats::hclust`.

Usage

```
cluster_hclust(
  k = 2,
  h = NULL,
 method = c("ward.D2", "ward.D", "single", "complete", "average", "mcquitty", "median",
    "centroid"),
 dist = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  scale = TRUE
)
```

Arguments

`k`	number of clusters to cut the tree (default 2)
`h`	height to cut the tree (optional; if provided, overrides `k`)
`method`	linkage method passed to `stats::hclust` (default "ward.D2")
`dist`	distance method passed to `stats::dist` (default "euclidean")
`scale`	logical; whether to scale data before distance (default TRUE)

Details

Computes a distance matrix (optionally after scaling) and builds a dendrogram. Clusters are obtained by cutting the tree with `k` (number of clusters) or `h` (height).

The base `clusterer()` uses `wcss` as a generic default, but `cluster_hclust()` specializes that choice because a hierarchical partition is obtained from a dendrogram and is better judged through separation/cohesion indices than through a centroid-style fallback alone.

Default evaluation in `cluster_hclust()` is:

- main metric: `metric_silhouette()`
- internal evaluation: `silhouette`, `davies_bouldin`, `calinski_harabasz`
- external evaluation: `entropy`, `purity`, `adjusted_rand_index`

The general `wcss` fallback is not kept as the main metric because the method is not fitted by directly minimizing that criterion.

Value

returns a hierarchical clustering object.

References

Johnson, S. C. (1967). Hierarchical clustering schemes. Psychometrika. Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. Calinski, T., & Harabasz, J. (1974). A dendrite method for cluster analysis.

Examples

```
data(iris)
model <- cluster_hclust(k = 3)
model <- fit(model, iris[,1:4])
clu <- cluster(model, iris[,1:4])
table(clu)
```

---

cluster_kmeans *k-means*

Description

k-means clustering using `stats::kmeans`.

Usage

```
cluster_kmeans(k = 1, nstart = 10)
```

Arguments

`k`	the number of clusters to form.
`nstart`	number of random initializations; the best (lowest within-cluster sum of squares) is kept. Default 10 for robust clustering (stats::kmeans uses 1).

Details

Partitions data into k clusters minimizing within-cluster sum of squares. The intrinsic quality metric returned is the total within-cluster SSE (lower is better).

The base `clusterer()` uses `wcss` as a generic default, but `cluster_kmeans()` specializes that choice because k-means is usually interpreted through compactness and separation between centroid-based groups.

Default evaluation in `cluster_kmeans()` is:

- main metric: `metric_silhouette()`
- internal evaluation: `silhouette`, `davies_bouldin`, `calinski_harabasz`
- external evaluation: `entropy`, `purity`, `adjusted_rand_index`

The general `wcss` fallback is not kept as the main metric here because it decreases mechanically as `k` grows and is therefore weaker as a standalone quality criterion for comparing partitions.

Value

returns a k-means object.

References

MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations. Lloyd, S. (1982). Least squares quantization in PCM. Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. Calinski, T., & Harabasz, J. (1974). A dendrite method for cluster analysis.

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 (Partitioning Around Medoids)*

Description

Clustering around representative data points (medoids) with PAM.

Usage

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

Arguments

k	the number of clusters to generate.

Details

More robust to outliers than k-means. The intrinsic metric reported is the within-cluster SSE to medoids.

PAM proceeds in two phases: BUILD, a greedy selection of the initial medoids, and SWAP, which exchanges a medoid with a non-medoid whenever that lowers the total distance to the nearest medoid. The SWAP phase uses the FastPAM1 scheme (Schubert & Rousseeuw, 2019), which evaluates the cost change of every candidate swap in a single pass and therefore reaches the same local optimum as classic PAM (identical medoids) while being markedly faster as the number of clusters grows (e.g. about 20x at k = 20 in the reference port). Inspired by the classic PAM and evolved with the FastPAM1 scheme (see references).

The base `clusterer()` uses `wcss` as a generic default, but `cluster_pam()` specializes that choice because PAM is commonly compared through partition quality indices that balance cohesion and separation.

Default evaluation in `cluster_pam()` is:

- main metric: `metric_silhouette()`
- internal evaluation: `silhouette`, `davies_bouldin`, `calinski_harabasz`
- external evaluation: `entropy`, `purity`, `adjusted_rand_index`

The generic `wcss` fallback is not used as the main metric because PAM is not optimized around centroids and is usually interpreted through medoid-based partition quality rather than only through within-cluster dispersion.

Value

returns PAM object.

References

Kaufman, L. and Rousseeuw, P. J. (1990). Finding Groups in Data: An Introduction to Cluster Analysis. Schubert, E. and Rousseeuw, P. J. (2019). Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. In Similarity Search and Applications (SISAP 2019), 171-187. Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. Calinski, T., & Harabasz, J. (1974). A dendrite method for cluster analysis.

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
```

`clusterer` *Clusterer*

Description

Base class for clustering algorithms and related evaluation utilities.

Usage

```
clusterer()
```

Details

The object stores shared state and defaults used by clustering methods. Current algorithms may still differ in how much they use this state, but the goal is to standardize future implementations around:

- `fit()` learning and storing model state
- `cluster()` producing labels from a fitted model
- configurable internal/external metrics and selection helpers via `cluutils()`

The base defaults are intentionally generic:

- `metric` = `metric_wcss()`
- `eval_internal` = `list(metric_wcss)`
- `eval_external` = `list(metric_entropy)`

These defaults provide a common contract for simple partition-based clustering, but they are not imposed on all subclasses. Concrete clustering methods can replace them when another metric is more faithful to the algorithm. For example, centroid and medoid methods often prefer separation-based indices such as silhouette, density-based methods may prefer counting noise points, mixture models naturally expose log-likelihood, and graph community detection is better described by modularity.

Therefore, users should treat the `clusterer()` defaults as a shared fallback, not as the final evaluation policy for every `cluster_*` implementation.

Value

returns a `clusterer` object

Examples

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

`cluutils` *Clustering utilities*

Description

Utility object that groups clustering metrics and model-selection helpers.

Usage

```
cluutils()
```

Details

The object organizes helpers into two semantic groups:

Metrics

- `metric_wcss()` computes the total within-cluster sum of squares.
- `metric_silhouette()` computes the mean silhouette score from pairwise distances.
- `metric_entropy()` computes external clustering entropy against a reference label.
- `metric_purity()` computes cluster purity against a reference label.
- `metric_davies_bouldin()` computes the Davies-Bouldin index.
- `metric_calinski_harabasz()` computes the Calinski-Harabasz score.
- `metric_adjusted_rand_index()` computes the adjusted Rand index.
- `metric_noise_points()` summarizes the number of noise points in density-based clustering.
- `metric_loglik()` and `metric_modularity()` expose model-specific quality summaries.

Selectors

- `selector_best()` selects the best hyperparameter value by direct optimization.
- `selector_elbow()` selects the elbow of a metric curve via maximum curvature.

Metric helpers return a standardized list with fields `metric`, `value`, `goal`, and `type`. This keeps the contract uniform even when the metrics themselves differ.

Value

returns a `cluutils` object exposing metric and selector helpers.

Examples

```
utils <- cluutils()

data(iris)
x <- iris[, 1:4]
clu <- stats::kmeans(x, centers = 3)$cluster

utils$metric_wcss(x, clu)
utils$metric_silhouette(x, clu)
utils$metric_entropy(clu, iris$Species)
utils$selector_best(c(0.31, 0.42, 0.39), goal = "maximize")
```

---

dal_base *Class dal_base*

Description

Minimal abstract base class for all DAL objects. Defines the common generics `fit()` and `action()` used by transforms and learners.

Usage

```
dal_base()
```

Value

returns a `dal_base` object

Examples

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

---

dal_learner *DAL Learner (base class)*

Description

Base ancestor for learning tasks (classification, regression, clustering, time series). Provides common behavior such as proxying `action()` to the model-specific operation (e.g., `predict()` for predictors, `cluster()` for clusterers) and an `evaluate()` generic.

An example of a learner is a decision tree (see `cla_dtree`).

Usage

```
dal_learner()
```

Value

returns a learner object

Examples

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

`dal_transform` *DAL Transform*

Description

Base class for data transformations with optional `fit()`/`inverse_transform()` support.

Usage

```
dal_transform()
```

Details

The default `transform()` calls the underlying `action.default()`; subclasses should implement `transform.className` and optionally `inverse_transform.className`.

Value

returns a `dal_transform` object

Examples

```
# See ?minmax or ?zscore for examples
```

`dal_tune` *DAL Tune (base for hyperparameter search)*

Description

Base class for hyperparameter optimization that stores a base model, a fold count, and a parameter grid. Specializations (classification/regression/clustering) implement the evaluation logic.

Usage

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

Arguments

base_model	base model for tuning
folds	number of folds for cross-validation
ranges	a list of hyperparameter ranges to explore

Details

Ranges are expanded via expand.grid, and selection is delegated to select_hyper() which can be overridden by subclasses to implement custom criteria.

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 sampling abstractions*

Description

Base class for sampling strategies that provide train/test splitting and k-fold partitioning. Two standard implementations are sample_random() and sample_stratified().

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)
```

`Diabetes` *Diabetes Data (Regression)*

Description

Ten baseline variables (age, sex, body mass index, average blood pressure, and six blood serum measurements) obtained for 442 diabetes patients, with a quantitative measure of disease progression one year after baseline as the target.

- age: age in years
- sex: sex
- bmi: body mass index
- bp: average blood pressure
- s1: tc, total serum cholesterol
- s2: ldl, low-density lipoproteins
- s3: hdl, high-density lipoproteins
- s4: tch, total cholesterol / HDL
- s5: ltg, log of serum triglycerides level
- s6: glu, blood sugar level
- target: quantitative measure of disease progression one year after baseline

Usage

```
data(Diabetes)
```

Format

Regression Dataset.

Source

The ten baseline variables and target, as distributed with scikit-learn.

References

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least Angle Regression. Annals of Statistics, 32(2), 407-499.

Examples

```
data(Diabetes)
head(Diabetes)
```

`discover` *Discover*

Description

Generic for pattern discovery.

Usage

```
discover(obj, ...)
```

Arguments

`obj`	A `pattern_miner` object.
`...`	Optional arguments passed to the concrete discovery method.

Value

Discovered patterns in the representation used by the mining engine.

`dt_pca` *PCA*

Description

Principal Component Analysis (PCA) for unsupervised dimensionality reduction. Transforms correlated variables into orthogonal principal components ordered by explained variance.

Usage

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

Arguments

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

Details

Fits PCA on (optionally) the numeric predictors only (excluding `attribute` when provided), removes constant columns, and selects the number of components by an elbow rule (minimum curvature) unless `components` is set explicitly. New data are projected with the same centering and scaling learned during `fit()`.

Value

returns an object of class `dt_pca`

References

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components.

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
```

---

feature_generation *Feature generation*

Description

Create new features from existing columns using named expressions.

Usage

```
feature_generation(...)
```

Arguments

...	named expressions that compute new features

Value

returns an object of class `feature_generation`

References

Kuhn, M. and Johnson, K. (2019). Feature Engineering and Selection: A Practical Approach for Predictive Models. CRC Press. Han, J., Kamber, M. and Pei, J. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann. (Attribute construction)

Examples

```
data(iris)
gen <- feature_generation(
 Sepal.Area = Sepal.Length * Sepal.Width,
 Petal.Area = Petal.Length * Petal.Width,
 Sepal.Ratio = Sepal.Length / Sepal.Width
)
iris_feat <- transform(gen, iris)
head(iris_feat)
```

`feature_selection_corr` *Feature selection by correlation*

Description

Remove highly correlated numeric features based on a correlation cutoff.

Usage

```
feature_selection_corr(cutoff = 0.9, features = NULL, keep = NULL)
```

Arguments

`cutoff`	correlation cutoff in [0, 1] above which one feature is removed
`features`	optional vector of feature names to consider (default: all numeric columns)
`keep`	optional vector of columns that should always be kept in `transform()`

Details

For each pair of numeric features whose absolute correlation exceeds `cutoff`, the one with the larger mean absolute correlation to the remaining features is removed, reducing redundancy among predictors.

Value

returns an object of class `feature_selection_corr`

References

Hall, M. A. (1999). Correlation-based Feature Selection for Machine Learning. PhD thesis, University of Waikato. Kuhn, M. (2008). Building Predictive Models in R Using the caret Package. Journal of Statistical Software, 28(5):1-26.

Examples

```
data(iris)
fs <- feature_selection_corr(cutoff = 0.9)
fs <- fit(fs, iris)
iris_fs <- transform(fs, iris)
fs$selected
names(iris_fs)
```

`feature_selection_fss` *Feature selection by forward stepwise search*

Description

Selects numeric predictors using forward stepwise subset search.

Usage

```
feature_selection_fss(attribute, features = NULL)
```

Arguments

`attribute`	target attribute name
`features`	optional vector of feature names (default: all columns except `attribute`)

Details

Adds one predictor at a time (greedy forward selection) and keeps the subset with the highest adjusted R-squared, which penalizes adding predictors that do not pay for their complexity. The target attribute must be numeric.

Implemented with base `stats::lm` (see references for the method).

Value

returns an object of class `feature_selection_fss`

References

Hocking, R. R. (1976). The Analysis and Selection of Variables in Linear Regression. Biometrics, 32(1):1-49. Miller, A. (2002). Subset Selection in Regression (2nd ed.). Chapman & Hall/CRC.

Examples

```
data(mtcars)
fs <- feature_selection_fss("mpg")
fs <- fit(fs, mtcars)
fs$selected
mtcars_fs <- transform(fs, mtcars)
names(mtcars_fs)
```

`feature_selection_info_gain` *Feature selection by information gain*

Description

Rank and select features using information gain with optional discretization.

Usage

```
feature_selection_info_gain(
  attribute,
  features = NULL,
  top = NULL,
  cutoff = 0,
  bins = 3
)
```

Arguments

`attribute`	target attribute name
`features`	optional vector of feature names (default: all columns except `attribute`)
`top`	optional number of top features to keep
`cutoff`	minimum information gain to keep a feature (default: 0)
`bins`	number of quantile bins for numeric features

Details

Numeric predictors are discretized by quantile bins before computing entropy-based information gain.

Value

returns an object of class `feature_selection_info_gain`

References

Quinlan, J. R. (1986). Induction of Decision Trees. Machine Learning, 1(1):81-106. Cover, T. M. and Thomas, J. A. (2006). Elements of Information Theory (2nd ed.). Wiley.

Examples

```
data(iris)
fg <- feature_generation(
 IsVersicolor = ifelse(Species == "versicolor", "versicolor", "not_versicolor")
)
iris_bin <- transform(fg, iris)
```

```
iris_bin$IsVersicolor <- factor(iris_bin$IsVersicolor)
fs <- feature_selection_info_gain("IsVersicolor", top = 2)
fs <- fit(fs, iris_bin)
fs$selected
iris_fs <- transform(fs, iris_bin)
names(iris_fs)
```

`feature_selection_lasso` *Feature selection by lasso*

Description

Selects predictors using L1-regularized regression.

Usage

```
feature_selection_lasso(attribute, features = NULL)
```

Arguments

`attribute`	target attribute name
`features`	optional vector of feature names (default: all numeric columns except `attribute`)

Details

Fits a lasso path with `glmnet` and keeps predictors with non-zero coefficients at `lambda.min`. The target attribute must be numeric.

Value

returns an object of class `feature_selection_lasso`

References

Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society, Series B, 58(1):267-288. Friedman, J., Hastie, T. and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1):1-22.

Examples

```
if (requireNamespace("glmnet", quietly = TRUE)) {
 data(Boston)
 fs <- feature_selection_lasso("medv")
 fs <- fit(fs, Boston)
 fs$selected
 boston_fs <- transform(fs, Boston)
 names(boston_fs)
}
```

`feature_selection_relief` *Feature selection by ReliefF*

Description

Rank and select features using the ReliefF weighting scheme.

Usage

```
feature_selection_relief(
  attribute,
  features = NULL,
  top = NULL,
  cutoff = NULL,
  m = 50,
  k = 10,
  seed = NULL
)
```

Arguments

`attribute`	target attribute name
`features`	optional vector of feature names (default: all columns except `attribute`)
`top`	optional number of top features to keep
`cutoff`	optional minimum ReliefF weight to keep a feature
`m`	number of sampled instances for ReliefF updates
`k`	number of nearest hits/misses per class averaged in each update (default 10)
`seed`	optional seed for the instance sampling (reproducibility)

Details

For each of `m` sampled instances the algorithm finds its `k` nearest neighbors of the same class (nearest hits) and its `k` nearest neighbors of each other class (nearest misses) in the normalized feature space, and updates every feature's weight by subtracting its average hit difference and adding its average miss difference, the latter weighted by each other class's prior probability `P(C) / (1 - P(class(R)))`. Averaging over `k` neighbors (ReliefF, Kononenko 1994) makes the estimate robust to noise, and the class-prior weighting handles multi-class targets.

Inspired by the original Relief and evolved to full ReliefF (see references).

Value

returns an object of class `feature_selection_relief`

References

Kira, K. and Rendell, L. A. (1992). A Practical Approach to Feature Selection. In Proceedings of the Ninth International Workshop on Machine Learning, 249-256. Kononenko, I. (1994). Estimating Attributes: Analysis and Extensions of RELIEF. In Proceedings of the European Conference on Machine Learning, 171-182.

Examples

```
data(iris)
fg <- feature_generation(
 IsVersicolor = ifelse(Species == "versicolor", "versicolor", "not_versicolor")
)
iris_bin <- transform(fg, iris)
iris_bin$IsVersicolor <- factor(iris_bin$IsVersicolor)
fs <- feature_selection_relief("IsVersicolor", top = 2, m = 50)
fs <- fit(fs, iris_bin)
fs$selected
transform(fs, iris_bin) |> names()
```

`feature_selection_stepwise`

Feature selection by stepwise model selection

Description

Select features using stepwise search over generalized linear models.

Usage

```
feature_selection_stepwise(
  attribute,
  features = NULL,
  direction = "forward",
  family = stats::binomial,
  trace = 0
)
```

Arguments

`attribute`	target attribute name
`features`	optional vector of feature names (default: all columns except `attribute`)
`direction`	stepwise direction: "forward", "backward", or "both"
`family`	glm family passed to `stats::glm` (default: `binomial`)
`trace`	level of tracing from `stats::step`

Details

Supports forward, backward, and both directions via `stats::step`. With the default `binomial` family, the target should represent a binary outcome.

Value

returns an object of class `feature_selection_stepwise`

References

Akaike, H. (1974). A New Look at the Statistical Model Identification. IEEE Transactions on Automatic Control, 19(6):716-723. Hastie, T., Tibshirani, R. and Friedman, J. (2009). The Elements of Statistical Learning (2nd ed.). Springer.

Examples

```
data(Boston)
fs <- feature_selection_stepwise("medv", direction = "forward", family = stats::gaussian)
fs <- fit(fs, Boston)
fs$selected
transform(fs, Boston) |> names()
```

---

fit *Fit*

Description

Generic to train/adjust an object using provided data and optional parameters.

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, ...)
```

Arguments

obj	an object containing the model and tuning configuration
data	the dataset used for training and evaluation
...	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 (elbow detection)*

Description

Computes a smoothing spline over a sequence and returns the location/value of maximum curvature, often used as an "elbow" detector.

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 (elbow detection)*

Description

Computes a smoothing spline over a sequence and returns the location/value of minimum curvature, complementary to maximum curvature and useful in elbow detection.

Usage

```
fit_curvature_min()
```

Value

Returns an object of class fit_curvature_min, 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)
```

---

hierarchy_cut *Hierarchy mapping by cut*

Description

Create a categorical hierarchy from a numeric attribute using cut points.

Usage

```
hierarchy_cut(attribute, breaks, labels = NULL, new_attribute = NULL)
```

Arguments

attribute	numeric attribute to discretize
breaks	numeric breakpoints for cut
labels	optional labels for the cut intervals
new_attribute	name of the new attribute (default: "attribute.Level")

Value

returns an object of class hierarchy_cut

References

Han, J., Kamber, M. and Pei, J. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann. (Concept hierarchy generation)

Examples

```
data(iris)
hc <- hierarchy_cut(
 "Sepal.Length",
 breaks = c(-Inf, 5.5, 6.5, Inf),
 labels = c("baixo", "medio", "alto")
)
iris_h <- transform(hc, iris)
table(iris_h$Sepal.Length.Level)
```

`imputation_predictive` *Predictive imputation base*

Description

Base class for supervised imputers that learn one target column from a set of source columns.

Usage

```
imputation_predictive(target, sources = NULL, method = c("median", "mean"))
```

Arguments

`target`	target column to impute
`sources`	optional vector of predictor column names
`method`	initial imputation method for numeric source columns: "median" or "mean"

Details

The target column is the attribute to be imputed. The source columns are the predictors used to estimate missing target values. If `sources = NULL`, all supported columns except the target are used. Missing values in source columns can be pre-imputed by a simpler method before fitting the predictive model.

Value

returns an object of class `imputation_predictive`

References

Han, J., Kamber, M. and Pei, J. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann. van Buuren, S. and Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3):1-67.

Examples

```
data(iris)
imp <- imputation_predictive(
 "Sepal.Length",
 sources = c("Sepal.Width", "Petal.Length", "Petal.Width", "Species")
)
class(imp)
```

`imputation_simple` *Simple imputation*

Description

Impute missing values in mixed datasets using simple statistics.

Usage

```
imputation_simple(method = c("median", "mean"), cols = NULL)
```

Arguments

`method`	imputation method for numeric columns: "median" or "mean"
`cols`	optional vector of column names to impute (default: all supported columns)

Details

Numeric columns are imputed with the mean or median. Factor, character, logical, and ordered columns are imputed with the mode (most frequent observed value). This class is intended as a low-complexity baseline for preprocessing workflows. The default recommendation of median for numeric variables follows standard data preprocessing guidance because it is less sensitive to outliers than the mean, while mode imputation is the usual baseline for categorical attributes.

Value

returns an object of class `imputation_simple`

References

Han, J., Kamber, M., Pei, J. (2011). Data Mining: Concepts and Techniques.

Little, R. J. A., Rubin, D. B. (2019). Statistical Analysis with Missing Data.

Examples

```
data(iris)
iris_na <- iris
iris_na$Sepal.Length[c(2, 10, 25)] <- NA
iris_na$Species[c(3, 15)] <- NA

imp <- imputation_simple(method = "median")
imp <- fit(imp, iris_na)
iris_imp <- transform(imp, iris_na)
summary(iris_imp$Sepal.Length)
table(iris_imp$Species, useNA = "ifany")
```

`imputation_tree` *Tree-based predictive imputation*

Description

Impute one target column from a set of source columns using a decision tree.

Usage

```
imputation_tree(target, sources = NULL, method = c("median", "mean"))
```

Arguments

`target`	target column to impute
`sources`	optional vector of predictor column names (default: all supported columns except `target`)
`method`	initial imputation method for numeric source columns: "median" or "mean"

Details

The method fits a tree with the observed values of the target column and uses the source columns as predictors. If source columns contain missing values, they are first completed with `imputation_simple()` so the tree can be trained and applied. The learned model imputes only the target column; source columns are preserved in the returned data.

Value

returns an object of class `imputation_tree`

References

Breiman, L., Friedman, J., Olshen, R., Stone, C. (1984). Classification and Regression Trees. Wadsworth.

van Buuren, S., Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1-67.

Examples

```
data(iris)
iris_na <- iris
iris_na$Sepal.Length[c(2, 10, 25)] <- NA

imp <- imputation_tree("Sepal.Length")
imp <- fit(imp, iris_na)
iris_imp <- transform(imp, iris_na)
summary(iris_imp$Sepal.Length)
sum(is.na(iris_imp$Sepal.Length))
```

---

inverse_transform *Inverse Transform*

Description

Optional inverse operation for a transformation; defaults to identity.

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

Split a dataset into k folds using a sampling strategy.

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

Linearly scales numeric columns to the [0,1] range per column.

Usage

```
minmax()
```

Details

For each numeric column j, computes (x - min_j) / (max_j - min_j). Constant columns map to 0.

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

Value

returns an object of class minmax

References

Han, J., Kamber, M., Pei, J. (2011). Data Mining: Concepts and Techniques. (Normalization section)

Examples

```
data(iris)
head(iris)

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

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

`na_removal` *Missing value removal*

Description

Remove rows (or elements) that contain missing values.

Usage

```
na_removal()
```

Details

For data frames or matrices, removes rows with any NA. For vectors, removes NA values.

Value

returns an object of class `na_removal`

References

Little, R. J. A. and Rubin, D. B. (2019). Statistical Analysis with Missing Data (3rd ed.). Wiley.

Examples

```
data(iris)
iris.na <- iris
iris.na$Sepal.Length[2] <- NA
obj <- na_removal()
iris.clean <- transform(obj, iris.na)
nrow(iris.clean)
```

`outliers_boxplot` *Outlier removal by boxplot (IQR rule)*

Description

Removes outliers from numeric columns using Tukey's boxplot rule: values below Q1 - alpha·IQR or above Q3 + alpha·IQR are flagged as outliers.

Usage

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

Arguments

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

Details

The default `alpha=1.5` corresponds to the standard boxplot whiskers; `alpha=3` is used for extreme outliers.

Value

returns an outlier object

References

Tukey, J. W. (1977). Exploratory Data Analysis. Addison-Wesley.

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` *Outlier removal by Gaussian 3-sigma rule*

Description

Removes outliers from numeric columns using the 3-sigma rule under a Gaussian assumption: values outside mean ± alpha·sd are flagged as outliers.

Usage

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

Arguments

`alpha`	gaussian threshold (default 3)

Value

returns an outlier object

References

Pukelsheim, F. (1994). The Three Sigma Rule. The American Statistician 48(2):88–91.

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
```

pat_apriori *Apriori rules*

Description

Frequent itemsets and association rules using `arules::apriori`.

Usage

```
pat_apriori(
  target = c("rules", "frequent itemsets"),
  supp = 0,
  conf = 0,
  support_strategy = pat_support_threshold("curvature"),
  confidence_strategy = pat_confidence_threshold(),
  minlen = 2,
  maxlen = 10,
  lhs = NULL,
  rhs = NULL,
  include = NULL,
  exclude = NULL,
  quality_filter = NULL,
  rule_filter = pat_rule_filter_none(),
  control = NULL
)
```

Arguments

`target`	mining target: `"rules"` or `"frequent itemsets"`
`supp`	minimum support threshold. If `0`, estimated during `fit()` using `support_strategy`.
`conf`	minimum confidence threshold for rules. If `0`, estimated during `fit()` using `confidence_strategy`.
`support_strategy`	support threshold strategy created with `pat_support_threshold()`
`confidence_strategy`	confidence threshold strategy created with `pat_confidence_threshold()`
`minlen`	minimum pattern length
`maxlen`	maximum pattern length
`lhs`	optional vector of items constrained to the left-hand side of rules
`rhs`	optional vector of items constrained to the right-hand side of rules
`include`	optional vector of items allowed in the discovered patterns
`exclude`	optional vector of items forbidden in the discovered patterns
`quality_filter`	optional quality filter created with `patutils()`
`rule_filter`	rule filter created with `pat_rule_filter_none()`, `pat_rule_filter_interest()`, or `pat_rule_filter_dara()`
`control`	list of control parameters

Value

returns a `pat_apriori` object

References

Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases. Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), 487-499.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
 data("AdultUCI", package = "arules")
 trans <- suppressWarnings(methods::as(as.data.frame(AdultUCI), "transactions"))
 utils <- patutils()
 pm <- pat_apriori(
   target = "rules",
   supp = 0,
   conf = 0,
   support_strategy = pat_support_threshold("curvature"),
   confidence_strategy = pat_confidence_threshold("rhs_baseline", margin = 0.1),
   minlen = 2,
   maxlen = 3,
   rhs = c("native-country=United-States"),
   quality_filter = utils$quality_min(confidence = 0.9, lift = 1.03),
   rule_filter = pat_rule_filter_interest(lift_min = 1),
   control = list(verbose = FALSE)
 )
 pm <- fit(pm, trans)
 rules <- suppressWarnings(discover(pm, trans))
 eval <- evaluate(pm, rules)
 eval$metrics
}
```

---

pat_confidence_threshold

Pattern confidence threshold strategy

Description

Configure how a minimum confidence threshold is estimated during `fit()`.

Usage

```
pat_confidence_threshold(
  method = c("rhs_baseline", "support_adaptive", "fixed"),
  value = 0.8,
  margin = 0.1,
```

```
  scale = 1,
  min = 0.5,
  max = 0.95
)
```

Arguments

method	Strategy name: "fixed", "rhs_baseline", or "support_adaptive".
value	Fixed confidence used by "fixed".
margin	Added to the RHS baseline confidence for "rhs_baseline".
scale	Multiplicative factor used by "support_adaptive".
min	Minimum allowed confidence.
max	Maximum allowed confidence.

Value

A pat_confidence_threshold object.

Examples

```
pat_confidence_threshold("rhs_baseline", margin = 0.1, min = 0.5)
pat_confidence_threshold("support_adaptive", min = 0.6, max = 0.9)
```

---

pat_dara *Divergent Association Ranking Analysis*

Description

Rank attribute values whose ordering diverges between the dataset and discovered rules.

Usage

```
pat_dara(
  data,
  rules,
  rhs_attr,
  rhs_value = NULL,
  attributes = NULL,
  rule_weight = NULL
)
```

Arguments

data	Reference data frame.
rules	Tidy rule data frame produced by pat_rules_tidy().
rhs_attr	Right-hand-side attribute used by the mined rules.
rhs_value	Optional right-hand-side value used to restrict data.
attributes	Optional attributes to compare. Defaults to common columns in data and rules.
rule_weight	Optional numeric column in rules used as rule frequency instead of one count per rule.

Details

DARA compares two value-frequency orderings for each explanatory attribute: the ordering in a reference dataset, optionally restricted to a right-hand-side value, and the ordering in a table of association rules. The right-hand-side attribute is required so it can be excluded from explanatory divergence. The returned counter c increases for values that are more prominent in rules than in the reference data and decreases for values that are less prominent in rules.

Value

A list of per-attribute data frames with columns value, dataset, rules, and c.

References

Baroni, L., Salles, R., Salles, S., Guedes, G., Porto, F., Bezerra, E., Barcellos, C., Pedroso, M., & Ogasawara, E. (2020). An analysis of malaria in the Brazilian Legal Amazon using divergent association rules. Journal of Biomedical Informatics, 108, 103512.

The canonical implementation is maintained in the Python pydaltoolbox port; this is a backport of it.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("AdultUCI", package = "arules")
  data <- as.data.frame(AdultUCI)
  data <- data[, c("workclass", "education", "marital-status", "occupation", "income")]
 pm <- fit(pat_apriori(target = "rules", supp = 0.05, conf = 0.6, rhs = "income=small"), data)
 rules <- pat_rules_tidy(discover(pm, data), transactions = data, item_columns = colnames(data))
  ranking <- pat_dara(data, rules, rhs_attr = "income", rhs_value = "small")
  ranking[["marital-status"]]
}
```

pat_filter_rules *Apply a rule filter*

Description

Filter association rules using a none, interest, or DARA rule filter.

Usage

```
pat_filter_rules(rules, filter, data = NULL, ...)
```

Arguments

rules	An arules rules object or a tidy rule data frame.
filter	A rule filter produced by pat_rule_filter_none(), pat_rule_filter_interest(), or pat_rule_filter_dara().
data	Optional transaction/reference data used to compute interest measures for arules rules.
...	Additional arguments reserved for future filters.

Value

Filtered rules in the same representation as rules.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("AdultUCI", package = "arules")
  data <- as.data.frame(AdultUCI)
  data <- data[, c("workclass", "education", "marital-status", "occupation", "income")]
 pm <- fit(pat_apriori(target = "rules", supp = 0.05, conf = 0.6, rhs = "income=small"), data)
  rules <- pat_rules_tidy(discover(pm, data), transactions = data)
  filter <- pat_rule_filter_interest(lift_min = 1)
  pat_filter_rules(rules, filter)
}
```

pat_prefix_span *PrefixSpan sequences*

Description

Frequent sequential pattern mining with PrefixSpan.

Usage

```
pat_prefix_span(
  support = 0,
  support_strategy = pat_support_threshold("min_count", min_count = 2),
  minlen = 1,
  maxlen = NULL,
  maxsize = NULL,
  mingap = NULL,
  maxgap = NULL,
  quality_filter = NULL
)
```

Arguments

support	minimum support threshold. If 0, estimated during fit() using support_strategy.
support_strategy	support threshold strategy created with pat_support_threshold()
minlen	minimum number of events per pattern
maxlen	maximum number of events per pattern (NULL = unbounded)
maxsize	maximum number of items per event (NULL = unbounded)
mingap	minimum gap between successive events (NULL = unbounded)
maxgap	maximum gap between successive events (NULL = unbounded)
quality_filter	optional quality filter created with patutils()

Details

PrefixSpan (Prefix-projected Sequential pattern mining) discovers frequent sequential patterns – ordered lists of itemsets – by growing patterns one prefix at a time and recursively mining the projected database of suffixes. A pattern grows by an i-extension (adding an item to its last itemset) or an s-extension (appending a new itemset); the support of a pattern is the fraction of input sequences in which it occurs in order. maxlen bounds the number of itemsets, maxsize the items per itemset, and mingap/maxgap constrain the time difference between consecutive matched events.

Sequences are given as a data frame with sequenceID, eventID and item columns, or as a list of sequences (each a list of itemsets, or of list(time, items) events).

Value

returns a pat_prefix_span object

References

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., & Hsu, M. (2001). PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. ICDE, 215-224.

The canonical implementation is maintained in the Python pydaltoolbox port; this is a backport of it.

Examples

```
seqs <- data.frame(
  sequenceID = c(1,1,1,2,2,3),
  eventID    = c(10,20,20,10,20,10),
  item       = c("A","B","C","A","C","A")
)
pm <- pat_prefix_span(support = 0.5)
pm <- fit(pm, seqs)
seqs_found <- discover(pm, seqs)
evaluate(pm, seqs_found)$metrics
```

---

pat_rule_filter_dara *DARA rule filter*

Description

Build a rule filter based on Divergent Association Ranking Analysis.

Usage

```
pat_rule_filter_dara(
  data,
  rhs_attr,
  rhs_value = NULL,
  attributes = NULL,
  min_score = 1,
  top = NULL,
  score = c("sum", "mean", "max"),
  use_abs = FALSE,
  rule_weight = NULL
)
```

Arguments

data	Reference data frame used by DARA.
rhs_attr	Right-hand-side attribute used by the mined rules.
rhs_value	Optional right-hand-side value used to restrict data.
attributes	Optional attributes to compare. Defaults to common columns.
min_score	Minimum DARA score required to keep a rule.
top	Optional number of highest-scored rules to keep.
score	Rule score aggregation: "sum", "mean", or "max".
use_abs	If TRUE, rank by absolute divergence counters.
rule_weight	Optional numeric rule column used to weight rule frequencies.

Details

DARA filters rules by scoring each antecedent with the divergence counters computed by `pat_dara()`. A rule with antecedent values that are more prominent in the rule set than in the exploratory data receives a higher positive score.

Value

A `pat_rule_filter_dara` object.

References

Baroni, L., Salles, R., Salles, S., Guedes, G., Porto, F., Bezerra, E., Barcellos, C., Pedroso, M., & Ogasawara, E. (2020). An analysis of malaria in the Brazilian Legal Amazon using divergent association rules. Journal of Biomedical Informatics, 108, 103512.

The canonical implementation is maintained in the Python `pydaltoolbox` port; this is a backport of it.

Examples

```
data <- data.frame(a = factor(c("x", "x", "y")), b = factor(c("z", "w", "z")))
filter <- pat_rule_filter_dara(data, rhs_attr = "b", rhs_value = "z")
class(filter)
```

---

pat_rule_filter_interest

Interesting rule filter

Description

Build a rule filter based on classical interest measures.

Usage

```
pat_rule_filter_interest(
  lift_min = 1,
  kulc_min = NULL,
  ir_max = NULL,
  support_min = NULL,
  confidence_min = NULL,
  count_min = NULL
)
```

Arguments

lift_min	Minimum lift. Use NULL to ignore lift.
kulc_min	Minimum Kulczynski measure. Use NULL to ignore Kulczynski.
ir_max	Maximum imbalance ratio. Use NULL to ignore imbalance ratio.
support_min	Minimum support. Use NULL to ignore support.
confidence_min	Minimum confidence. Use NULL to ignore confidence.
count_min	Minimum rule count. Use NULL to ignore count.

Details

This filter follows the usual post-processing step for association rules: rules are discovered first, then kept only when their interest measures pass user-defined thresholds. lift > 1, high Kulczynski, and low imbalance ratio are common criteria discussed in association-rule mining texts such as Han, Kamber, and Pei.

Value

A pat_rule_filter_interest object.

Examples

```
filter <- pat_rule_filter_interest(lift_min = 1, kulc_min = 0.6, ir_max = 0.9)
class(filter)
```

---

pat_rule_filter_none *No rule filtering*

Description

Build an explicit rule filter that leaves discovered rules unchanged.

Usage

```
pat_rule_filter_none()
```

Details

Use this filter when the miner should only discover candidate rules and skip post-mining interestingness filtering. It is the default rule filter for association-rule miners.

Value

A pat_rule_filter_none object.

Examples

```
filter <- pat_rule_filter_none()
class(filter)
```

pat_rules_tidy *Tidy association rules*

Description

Convert association rules into a tabular representation suitable for downstream analysis.

Usage

```
pat_rules_tidy(rules, transactions = NULL, item_columns = NULL)
```

Arguments

rules	An arules rules object.
transactions	Optional transaction data used to compute additional interest measures.
item_columns	Optional columns to recreate from left-hand-side items.

Value

A data frame with rule labels, quality measures, and optional item columns.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("AdultUCI", package = "arules")
  data <- as.data.frame(AdultUCI)
  data <- data[, c("workclass", "education", "marital-status", "occupation", "income")]
 pm <- fit(pat_apriori(target = "rules", supp = 0.05, conf = 0.6, rhs = "income=small"), data)
  rules <- pat_rules_tidy(discover(pm, data), transactions = data)
  head(rules)
}
```

pat_support_threshold *Pattern support threshold strategy*

Description

Configure how a minimum support threshold is estimated during fit().

Usage

```
pat_support_threshold(
  method = c("curvature", "min_count", "sqrt", "quantile", "density"),
  min_count = 10,
  q = 0.25,
  scale = 1,
  min = 0.001,
  max = 0.5
)
```

Arguments

method	Strategy name: "curvature", "min_count", "sqrt", "quantile", or "density".
min_count	Minimum absolute transaction count for "min_count".
q	Quantile of item frequencies for "quantile".
scale	Multiplicative factor used by "quantile" and "density".
min	Minimum allowed support.
max	Maximum allowed support.

Value

A pat_support_threshold object.

Examples

```
pat_support_threshold("curvature")
pat_support_threshold("min_count", min_count = 20)
pat_support_threshold("quantile", q = 0.25, scale = 0.5)
```

---

pattern_miner *Pattern Miner*

Description

Base class for frequent pattern and sequence mining.

Usage

```
pattern_miner()
```

Details

Pattern miners follow a lightweight Experiment Line:

- `fit()` validates the mining input and stores a schema signature
- `discover()` runs the mining algorithm on data compatible with that schema
- `evaluate()` summarizes pattern quality and filtering effects

Different miners may normalize their inputs differently (for example, item transactions versus sequence transactions), but the base contract remains the same.

Value

A `pattern_miner` object.

Examples

```
miner <- pattern_miner()
class(miner)
```

`patutils` *Pattern mining utilities*

Description

Utility object that groups filtering helpers and evaluation metrics for pattern mining.

Usage

```
patutils()
```

Details

The object groups two kinds of helpers:

- quality-filter builders such as `quality_min()` and `quality_max()`
- descriptive metrics for discovered patterns such as pattern count, mean support, mean confidence, mean lift, mean length, and retained ratio after filtering

Value

returns a `patutils` object

Examples

```
utils <- patutils()
utils$quality_min(confidence = 0.8, lift = 1.1)
```

plot_bar *Plot Bar Graph*

Description

Draw a simple bar chart from a two-column data frame: first column as categories (x), second as values (y).

Usage

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

Arguments

data	Two-column data frame: category in the first column, numeric values in the second.
label_x	Label for the x-axis.
label_y	Label for the y-axis.
colors	Optional fill color.
alpha	Bar transparency between 0 and 1.

Details

If colors is provided, a constant fill is used; otherwise ggplot2's default palette applies. alpha controls bar transparency. The first column is coerced to factor when needed, which makes the function convenient for quick summaries of grouped values.

Value

A ggplot2::ggplot object.

References

Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

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

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

`plot_boxplot` *Plot boxplot*

Description

Boxplots for each numeric column of a data.frame.

Usage

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

Arguments

`data`	data.frame with one or more numeric columns
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	optional fill color for boxes
`barwidth`	width of the box (numeric)

Details

The data is melted to long format and a box is drawn per original column. If `colors` is provided, a constant fill is applied to all boxes. Use `barwidth` to control box width.

Value

returns a ggplot2::ggplot graphic

References

Tukey, J. W. (1977). Exploratory Data Analysis. Addison-Wesley. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

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

`plot_boxplot_class` *Boxplot per class*

Description

Boxplots of a numeric column grouped by a class label.

Usage

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

Arguments

`data`	data.frame with a grouping column and one numeric column
`class_label`	name of the grouping (class) column
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	optional fill color for the boxes

Details

Expects a data.frame with the grouping column named in `class_label` and one numeric column. The function melts to long format and draws per-group distributions.

Value

returns a ggplot2::ggplot graphic

References

Tukey, J. W. (1977). Exploratory Data Analysis. Addison-Wesley. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

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

`plot_correlation` *Plot correlation*

Description

Correlation heatmap with optional labels and triangle filtering.

Usage

```
plot_correlation(
  df,
  vars = NULL,
  method = c("pearson", "spearman", "kendall"),
  use = "pairwise.complete.obs",
  triangle = c("full", "upper", "lower"),
  reorder = c("none", "hclust", "alphabetical"),
  digits = 2,
  label_size = 3,
  tile_color = "white",
  show_diag = TRUE,
  title = NULL
)
```

Arguments

`df`	data.frame with numeric columns
`vars`	optional vector of column names to include
`method`	correlation method: "pearson", "spearman", or "kendall"
`use`	handling of missing values for `stats::cor`
`triangle`	which triangle to show: "full", "upper", or "lower"
`reorder`	reordering strategy: "none", "hclust", or "alphabetical"
`digits`	number of digits for labels
`label_size`	size of label text
`tile_color`	border color for tiles
`show_diag`	whether to show the diagonal
`title`	optional plot title

Details

Computes a correlation matrix from numeric columns (or `vars`) and renders a ggplot2 heatmap with values annotated. Supports reordering by hierarchical clustering or alphabetically.

Value

returns a ggplot2::ggplot graphic

References

Friendly, M. (2002). Corrgrams: Exploratory Displays for Correlation Matrices. The American Statistician, 56(4):316-324. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

```
data(iris)
grf <- plot_correlation(iris[,1:4])
plot(grf)
```

`plot_dendrogram` *Plot dendrogram*

Description

Dendrogram plot for an hclust or dendrogram object using ggplot2.

Usage

```
plot_dendrogram(hc, labels = TRUE, label_size = 3, title = NULL)
```

Arguments

`hc`	an object of class `hclust` or `dendrogram`
`labels`	logical; whether to draw leaf labels
`label_size`	label text size
`title`	optional plot title

Details

Converts a dendrogram into line segments and renders it with ggplot2.

Value

returns a ggplot2::ggplot graphic

References

Ward, J. H. (1963). Hierarchical Grouping to Optimize an Objective Function. Journal of the American Statistical Association, 58(301):236-244. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

```
data(iris)
hc <- hclust(dist(scale(iris[,1:4])), method = "ward.D2")
grf <- plot_dendrogram(hc)
plot(grf)
```

plot_density *Plot density*

Description

Kernel density plot for one or multiple numeric columns.

Usage

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

Arguments

data	data.frame with one or more numeric columns
label_x	x-axis label
label_y	y-axis label
colors	optional fill color (single column) or vector for groups
bin	optional bin width passed to geom_density
alpha	fill transparency (0–1)

Details

If data has multiple numeric columns, densities are overlaid and filled by column (group). When a single column is provided, colors (if set) is used as a constant fill. The bin argument is passed to geom_density(binwidth=...).

Value

returns a ggplot2::ggplot graphic

References

Silverman, B. W. (1986). Density Estimation for Statistics and Data Analysis. Chapman & Hall.
Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

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

`plot_density_class` *Plot density per class*

Description

Kernel density plot grouped by a class label.

Usage

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

Arguments

`data`	data.frame with class label and a numeric column
`class_label`	name of the grouping (class) column
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	optional vector of fills per class
`bin`	optional bin width passed to `geom_density`
`alpha`	fill transparency (0–1)

Details

Expects `data` with a grouping column named in `class_label` and one numeric column. Each group is filled with a distinct color (if provided).

Value

returns a ggplot2::ggplot graphic

References

Silverman, B. W. (1986). Density Estimation for Statistics and Data Analysis. Chapman & Hall. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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

Grouped (side-by-side) bar chart for multiple series per category.

Usage

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

Arguments

`data`	data.frame with category in first column and series in remaining columns
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	optional vector of fill colors, one per series
`alpha`	bar transparency (0–1)

Details

Expects a data.frame where the first column is the category (x) and the remaining columns are numeric series. Bars are grouped by series. Provide `colors` with length equal to the number of series to set fills.

Value

returns a ggplot2::ggplot graphic

References

Cleveland, W. S. and McGill, R. (1984). Graphical Perception. Journal of the American Statistical Association, 79(387):531-554. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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

Histogram for a numeric column using ggplot2.

Usage

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

Arguments

`data`	data.frame with one numeric column (first column is used if multiple)
`label_x`	x-axis label
`label_y`	y-axis label
`color`	fill color
`alpha`	transparency level (0–1)

Details

If multiple columns are provided, only the first is used. Breaks are computed via `graphics::hist` to mirror base R binning. `color` controls the fill; `alpha` the transparency.

Value

returns a ggplot2::ggplot graphic

References

Sturges, H. A. (1926). The choice of a class interval. Journal of the American Statistical Association, 21(153):65-66. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

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

`plot_lollipop` *Plot lollipop*

Description

Lollipop chart (stick + circle + value label) per category.

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 with category and numeric values
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	stick/circle color
`color_text`	color of the text inside the circle
`size_text`	text size
`size_ball`	circle size
`alpha_ball`	circle transparency (0–1)
`min_value`	minimum baseline for the stick
`max_value_gap`	gap from value to stick end

Details

Expects a data.frame with category in the first column and numeric values in subsequent columns. Circles are drawn at values, with vertical segments extending from `min_value` to `value - max_value_gap`.

Value

returns a ggplot2::ggplot graphic

References

Tufte, E. R. (1983). The Visual Display of Quantitative Information. Graphics Press. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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_parallel *Plot parallel coordinates*

Description

Parallel coordinates plot for multivariate data, with lines colored by group.

Usage

```
plot_parallel(data, columns, group, colors = NULL, title = NULL)
```

Arguments

data	data.frame
columns	numeric columns to include (indices or names)
group	grouping column (index or name)
colors	optional vector of colors for groups
title	optional title

Details

Each selected variable becomes a parallel vertical axis and each observation a polyline crossing them, so class separation and multivariate patterns stand out. Variables are standardized per column (z-score) before plotting so axes on different scales are comparable.

Value

returns a ggplot2::ggplot graphic

References

Inselberg, A. (1985). The plane with parallel coordinates. The Visual Computer, 1(2), 69-91. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

```
data(iris)
grf <- plot_parallel(iris, columns = 1:4, group = 5)
plot(grf)
```

`plot_pieplot` *Plot pie*

Description

Pie chart from a two-column data.frame (category, value) using polar coordinates.

Usage

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

Arguments

`data`	two-column data.frame with category and value
`label_x`	x-axis label (unused in pie, kept for symmetry)
`label_y`	y-axis label (unused in pie)
`colors`	vector of slice fills
`textcolor`	label text color
`bordercolor`	slice border color

Details

Slices are sized by the second (numeric) column. Text and border colors can be customized.

Value

returns a ggplot2::ggplot graphic

References

Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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_pixel` *Plot pixel visualization*

Description

Pixel-oriented visualization of a numeric matrix or data.frame.

Usage

```
plot_pixel(
  data,
  colors = NULL,
  title = NULL,
  label_x = "sample",
  label_y = "Attributes"
)
```

Arguments

`data`	numeric matrix or data.frame
`colors`	optional vector of colors for the fill gradient
`title`	optional plot title
`label_x`	x-axis label
`label_y`	y-axis label

Details

Renders a heatmap-like plot where each cell is a pixel. Useful for multivariate inspection.

Value

returns a ggplot2::ggplot graphic

References

Keim, D. A. (2000). Designing Pixel-Oriented Visualization Techniques: Theory and Applications. IEEE Transactions on Visualization and Computer Graphics, 6(1):59-78. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Examples

```
data(iris)
grf <- plot_pixel(as.matrix(iris[,1:4]), title = "Iris")
plot(grf)
```

`plot_points` *Plot points*

Description

Dot chart for multiple series across categories (points only).

Usage

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

Arguments

`data`	data.frame with category + one or more numeric columns
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	optional color vector for series

Details

Expects a data.frame with category in the first column and one or more numeric series. Points are colored by series (legend shows original column names). Supply `colors` to override the palette.

Value

returns a ggplot2::ggplot graphic

References

Cleveland, W. S. (1985). The Elements of Graphing Data. Wadsworth. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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

Radar (spider) chart for a single profile of variables using radial axes.

Usage

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

Arguments

data	two-column data.frame: variable name and value
label_x	x-axis label (unused; variable names are shown around the circle)
label_y	y-axis label
colors	line/fill color for the polygon

Details

Expects a two-column data.frame with variable names in the first column and numeric values in the second. The graphic is built as an n-sided polygon, where n is the number of variables, so at least three variables are required. The function already sets the drawing limits for the full polygon; adding `ylim()` or other Cartesian clipping after the fact can hide part of the radar.

Value

returns a ggplot2::ggplot graphic

References

Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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")
plot(grf)
```

plot_scatter *Scatter graph*

Description

Scatter plot from a long data.frame with columns named `x`, `value`, and `variable`.

Usage

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

Arguments

`data`	long data.frame with columns `x`, `value`, `variable`
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	optional color(s); for numeric `variable`, supply a gradient as c(low, high)

Details

Colors are mapped to `variable`. If `variable` is numeric, a gradient color scale is used when `colors` is provided.

Value

return a ggplot2::ggplot graphic

References

Cleveland, W. S. and McGill, R. (1984). Graphical Perception. Journal of the American Statistical Association, 79(387):531-554. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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

Line plot for one or more series over a common x index.

Usage

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

Arguments

`data`	data.frame with x in the first column and series in remaining columns
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	optional vector of colors for series

Details

Expects a data.frame where the first column is the x index and remaining columns are numeric series. Points and lines are drawn per series; supply `colors` to override the palette.

Value

returns a ggplot2::ggplot graphic

References

Cleveland, W. S. (1985). The Elements of Graphing Data. Wadsworth. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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

Stacked bar chart for multiple series per category.

Usage

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

Arguments

`data`	data.frame with category in first column and series in remaining columns
`label_x`	x-axis label
`label_y`	y-axis label
`colors`	optional vector of fill colors, one per series
`alpha`	bar transparency (0–1)

Details

Expects a data.frame with category in the first column and series in remaining columns. Bars are stacked within each category. Provide `colors` (one per series) to control fills.

Value

returns a ggplot2::ggplot graphic

References

Cleveland, W. S. and McGill, R. (1984). Graphical Perception. Journal of the American Statistical Association, 79(387):531-554. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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

Simple time series plot with points and a line.

Usage

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

Arguments

x	time index (numeric vector) or NULL to use 1:length(y)
y	numeric series
label_x	x-axis label
label_y	y-axis label
color	color for the series

Details

If x is NULL, an integer index 1:n is used. The color applies to both points and line.

Value

returns a ggplot2::ggplot graphic

References

Cleveland, W. S. (1985). The Elements of Graphing Data. Wadsworth. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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 time series with predictions*

Description

Plot original series plus dashed lines for in-sample adjustment and optional out-of-sample predictions.

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 (numeric vector) or NULL to use 1:length(y)
y	numeric time series
yadj	fitted/adjusted values for the training window
ypred	optional predicted values after the training window
label_x	x-axis title
label_y	y-axis title
color	color for the original series
color_adjust	color for the adjusted values (dashed)
color_prediction	color for the predictions (dashed)

Details

yadj length defines the training segment; ypred (if provided) is appended after yadj.

Value

returns a ggplot2::ggplot graphic

References

Hyndman, R. J. and Athanasopoulos, G. (2021). Forecasting: Principles and Practice (3rd ed.). OTexts. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

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, ypred=prediction)
plot(grf)
```

---

predictor *Predictor (base for classification/regression)*

Description

Ancestor class for supervised predictors (classification and regression). Provides a default `fit()` to record feature names and proxies `action()` to `predict()`.

An example predictor is a decision tree classifier (`cla_dtree`).

Usage

```
predictor()
```

Value

returns a predictor object

Examples

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

---

predictor_prepare_fit *Prepare predictor fit data*

Description

Normalizes supervised fit input and records predictor columns by removing the target attribute from the data columns.

Usage

```
predictor_prepare_fit(obj, data)
```

Arguments

obj	predictor-like object containing attribute
data	training data

Value

returns a list with the updated object and normalized data

---

reg_dtree *Decision Tree for regression*

Description

Regression tree using recursive partitioning via the tree package.

Usage

```
reg_dtree(attribute)
```

Arguments

attribute	attribute target to model building.

Details

Splits are chosen to reduce squared error within nodes; result is an interpretable set of piecewise constants.

Value

returns a decision tree regression object

References

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). Classification and Regression Trees. Wadsworth.

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 *K-Nearest Neighbors (KNN) Regression*

Description

KNN regression using `FNN::knn.reg`, predicting by averaging the targets of the k nearest neighbors.

Usage

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

Arguments

`attribute`	attribute target to model building
`k`	number of k neighbors

Details

Non-parametric approach suitable for local smoothing. Sensitive to feature scaling; consider normalization beforehand.

Value

returns a knn regression object

References

Altman, N. (1992). An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression.

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_lm` *Linear regression (lm)*

Description

Linear regression using `stats::lm`.

Usage

```
reg_lm(formula = NULL, attribute = NULL, features = NULL)
```

Arguments

`formula`	optional regression formula (e.g., y ~ x1 + x2).
`attribute`	target attribute name (used when formula is NULL)
`features`	optional vector of feature names (used when formula is NULL)

Value

returns a `reg_lm` object

References

Draper, N. R. and Smith, H. (1998). Applied Regression Analysis (3rd ed.). Wiley. Weisberg, S. (2005). Applied Linear Regression (3rd ed.). Wiley.

Examples

```
if (requireNamespace("MASS", quietly = TRUE)) {
 data(Boston, package = "MASS")

 # Simple linear regression
 model_simple <- reg_lm(formula = medv ~ lstat)
 model_simple <- fit(model_simple, Boston)
 pred_simple <- predict(model_simple, Boston)
 eval_simple <- evaluate(model_simple, Boston$medv, pred_simple)
 eval_simple$metrics

 # Polynomial regression (degree 2)
 model_poly <- reg_lm(formula = medv ~ poly(lstat, 2, raw = TRUE))
 model_poly <- fit(model_poly, Boston)
 pred_poly <- predict(model_poly, Boston)
 eval_poly <- evaluate(model_poly, Boston$medv, pred_poly)
```

```
 eval_poly$metrics

 # Multiple regression
 model_multi <- reg_lm(formula = medv ~ lstat + rm + ptratio)
 model_multi <- fit(model_multi, Boston)
 pred_multi <- predict(model_multi, Boston)
 eval_multi <- evaluate(model_multi, Boston$medv, pred_multi)
 eval_multi$metrics
}
```

`reg_mlp` *MLP for regression*

Description

Multi-Layer Perceptron regression using `nnet::nnet` (single hidden layer).

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

Details

Feedforward neural network with `size` hidden units and L2 regularization controlled by `decay`. Data should be scaled for stable training.

Value

returns a object of class `reg_mlp`

References

Bishop, C. M. (1995). Neural Networks for Pattern Recognition. Oxford University Press.

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

Regression via Random Forests, an ensemble of decision trees trained on bootstrap samples with random feature subsetting at each split. This wrapper uses the `randomForest` package API.

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

Details

Random Forests reduce variance and are robust to overfitting on tabular data. Key hyperparameters are the number of trees (`ntree`), the number of variables tried at each split (`mtry`), and the minimum node size (`nodesize`).

Value

returns an object of class `reg_rfobj`

References

Breiman, L. (2001). Random Forests. Machine Learning 45(1):5–32. Liaw, A. and Wiener, M. (2002). Classification and Regression by randomForest. R News.

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

Support Vector Regression (SVR) using `e1071::svm`.

Usage

```
reg_svm(
  attribute,
  epsilon = 0.1,
  cost = 10,
  kernel = c("radial", "linear", "polynomial", "sigmoid")
)
```

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)

Details

SVR optimizes a margin with an epsilon-insensitive loss around the regression function. The `cost` controls regularization strength; `epsilon` sets the width of the insensitive tube; and `kernel` defines the feature map (linear, radial, polynomial, sigmoid).

Value

returns a SVM regression object

References

Drucker, H., Burges, C., Kaufman, L., Smola, A., Vapnik, V. (1997). Support Vector Regression Machines. Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines.

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 tuning (k-fold CV)*

Description

Tune hyperparameters of a base regressor via k-fold cross-validation minimizing an error metric (MSE).

Usage

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

Arguments

`base_model`	base model for tuning
`folds`	number of folds for cross-validation
`ranges`	a list of hyperparameter ranges to explore

Value

returns a reg_tune object.

References

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.

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"), folds=3, ranges = list(size=c(3), decay=c(0.1,0.5)))

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

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

---

regression *Regression base class*

Description

Ancestor class for regression models. Stores the target attribute and provides common evaluation metrics.

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
```

`sample_balance` *Class balancing (up/down sampling)*

Description

Balance class distribution using up-sampling or down-sampling.

Usage

```
sample_balance(attribute, method = c("down", "up"))
```

Arguments

`attribute`	target class attribute name
`method`	balancing method: "down" or "up"

Details

Down-sampling reduces every class to the minority count (without replacement); up-sampling grows every class to the majority count (sampling with replacement).

Value

returns an object of class `sample_balance`

References

He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering, 21(9):1263-1284.

Examples

```
data(iris)
iris_imb <- iris[iris$Species != "setosa", ]
sb <- sample_balance("Species", method = "down")
iris_bal <- transform(sb, iris_imb)
table(iris_bal$Species)
```

sample_groups *Group sampling*

Description

Sample entire groups defined by a categorical attribute. In sampling theory, this design is known as cluster sampling (also called one-stage cluster sampling or sampling by groups). The groups are assumed to be pre-defined in the data; this function does not infer groups with clustering algorithms such as k-means.

Usage

```
sample_groups(attribute, n_groups)
```

Arguments

attribute	group-defining attribute name
n_groups	number of groups to sample

Value

returns an object of class sample_groups

References

Cochran, W. G. (1977). Sampling Techniques (3rd ed.). Wiley.

Examples

```
data(iris)
sc <- sample_groups("Species", n_groups = 2)
iris_sc <- transform(sc, iris)
table(iris_sc$Species)
```

sample_random *Random Sampling*

Description

Train/test split and k-fold partitioning by simple random sampling.

Usage

```
sample_random()
```

Details

This is the simplest partitioning strategy in daltoolbox. It is a good starting point when the dataset is reasonably balanced and the user wants to understand the train/test workflow before moving to stratified or more specialized sampling schemes.

Value

An object of class sample_random.

References

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. IJCAI.

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_simple *Simple sampling*

Description

Sample rows or elements with or without replacement.

Usage

```
sample_simple(size, replace = FALSE, prob = NULL)
```

Arguments

size	number of samples to draw
replace	logical; sample with replacement if TRUE
prob	optional vector of sampling probabilities

Value

returns an object of class `sample_simple`

References

Cochran, W. G. (1977). Sampling Techniques (3rd ed.). Wiley.

Examples

```
data(iris)
srswor <- sample_simple(size = 10, replace = FALSE)
srswr <- sample_simple(size = 10, replace = TRUE)
sample_wor <- transform(srswor, iris$Sepal.Length)
sample_wr <- transform(srswr, iris$Sepal.Length)
sample_wor
sample_wr
```

`sample_stratified` *Stratified Sampling*

Description

Train/test split and k-fold partitioning that preserve the target class proportions.

Usage

```
sample_stratified(attribute)
```

Arguments

`attribute`	Name of the target attribute whose class proportions should be preserved.

Details

Use this sampler when the response distribution matters, especially in imbalanced classification problems. Compared with simple random sampling, it reduces the chance that one split will over-represent or underrepresent a class by accident.

Value

An object of class `sample_stratified`.

References

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. IJCAI.

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 of hyperparameters*

Description

Generic to select the best hyperparameters from cross-validation results; subclasses can override.

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

Assign a named list of parameters to matching fields in the object (best-effort).

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 (returns object unchanged).

Usage

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

Arguments

obj	object
params	parameters

Value

returns the object unchanged

smoothing *Smoothing (binning/quantization)*

Description

Family of smoothing methods that reduce noise by replacing values with the mean of a bin. Supported strategies include equal-interval bins, equal-frequency (quantile) bins, k-means quantization, and class-aware clustering.

Usage

```
smoothing(n)
```

Arguments

n	number of bins

Details

The smoothing level is controlled by n (number of bins/levels). The base helper tune() chooses n by locating the elbow (maximum curvature) of the MSE curve across candidates. Concrete subclasses may override that criterion when supervision is required. After fit(), values are mapped to bin means via transform().

Value

returns an object of class `smoothing`

References

Han, J., Kamber, M. and Pei, J. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann. (Discretization)

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

bins <- cut(iris$Sepal.Length, unique(obj$interval.adj), FALSE, include.lowest = TRUE)
entro <- evaluate(obj, bins, iris$Species)
entro$entropy
```

`smoothing_cluster` *Smoothing by class-aware clustering*

Description

Discretize a numeric attribute into `n` bins by clustering the attribute together with a one-hot representation of the class label, then projecting the clusters back to ordered cut points on the numeric axis.

Usage

```
smoothing_cluster(class_label, n)
```

Arguments

`class_label`	name of the class attribute
`n`	number of bins

Value

returns an object of class `smoothing_cluster`

References

Han, J., Kamber, M., Pei, J. (2011). Data Mining: Concepts and Techniques. (Discretization)

Examples

```
data(iris)
cluster_data <- iris[, c("Sepal.Length", "Species")]
obj <- smoothing_cluster("Species", n = 2)
obj <- fit(obj, cluster_data)
sl.bi <- transform(obj, iris$Sepal.Length)
table(sl.bi)
obj$interval

bins <- cut(iris$Sepal.Length, unique(obj$interval.adj), FALSE, include.lowest = TRUE)
entro <- evaluate(obj, bins, iris$Species)
entro$entropy
```

---

smoothing_freq *Smoothing by equal frequency*

Description

Discretize a numeric vector into n bins with approximately equal frequency (quantile cuts), and replace each value by the mean of its bin.

Usage

```
smoothing_freq(n)
```

Arguments

n	number of bins

Value

returns an object of class smoothing_freq

References

Han, J., Kamber, M., Pei, J. (2011). Data Mining: Concepts and Techniques. (Discretization)

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

bins <- cut(iris$Sepal.Length, unique(obj$interval.adj), FALSE, include.lowest = TRUE)
entro <- evaluate(obj, bins, iris$Species)
entro$entropy
```

smoothing_inter *Smoothing by equal interval*

Description

Discretize a numeric vector into n equal-width intervals (robust bounds via boxplot whiskers) and replace each value by the bin mean.

Usage

```
smoothing_inter(n)
```

Arguments

n	number of bins

Value

returns an object of class smoothing_inter

References

Han, J., Kamber, M., Pei, J. (2011). Data Mining: Concepts and Techniques. (Discretization)

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

bins <- cut(iris$Sepal.Length, unique(obj$interval.adj), FALSE, include.lowest = TRUE)
entro <- evaluate(obj, bins, iris$Species)
entro$entropy
```

smoothing_quantization *Smoothing by quantization (k-means)*

Description

Quantize a numeric vector into n levels using k-means on the values and replace each value by its cluster mean (vector quantization).

Usage

```
smoothing_quantization(n)
```

Arguments

n	number of bins

Value

returns an object of class smoothing_quantization

References

MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations.

Examples

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

bins <- cut(iris$Sepal.Length, unique(obj$interval.adj), FALSE, include.lowest = TRUE)
entro <- evaluate(obj, bins, iris$Species)
entro$entropy
```

---

train_test *Train-Test Partition*

Description

Partition a dataset into training and test sets using a sampling strategy.

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

Build one train/test split from a list of folds produced by cross-validation.

Usage

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

Arguments

`folds`	Data already partitioned into folds.
`k`	Fold index to be used as the test set.

Details

The k-th fold is returned as the test set and all remaining folds are concatenated into the training set. This helper makes manual cross-validation loops easier to explain and reproduce.

Value

A list with two elements:

- train: data frame containing all folds except the k-th.
- test: data frame corresponding to the k-th fold.

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

Generic to apply a transformation to data.

Usage

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

Arguments

`obj`	a `dal_transform` object.
`...`	optional arguments.

Value

returns a transformed data.

Examples

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

transform.pat_rule_filter *Transform rules with a rule filter*

Description

Experiment Line `transform()` entry point for rule filters; delegates to `pat_filter_rules()`.

Usage

```
## S3 method for class 'pat_rule_filter'
transform(obj, data, reference = NULL, ...)
```

Arguments

`obj`	A rule filter created with `pat_rule_filter_none()`, `pat_rule_filter_interest()`, or `pat_rule_filter_dara()`.
`data`	An `arules` rules object or a tidy rule data frame to filter.
`reference`	Optional transaction/reference data used to compute interest measures for `arules` rules.
`...`	Additional arguments reserved for future filters.

Value

Filtered rules in the same representation as `data`.

zscore *Z-score normalization*

Description

Standardize numeric columns to zero mean and unit variance, optionally rescaled to a target mean (`nmean`) and sd (`nsd`).

Usage

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

Arguments

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

Details

For each numeric column j, computes ((x - mean_j)/sd_j) * nsd + nmean. Constant columns become nmean.

$zscore = (x - mean(x))/sd(x)$

Value

returns the z-score transformation object

References

Han, J., Kamber, M., Pei, J. (2011). Data Mining: Concepts and Techniques. (Standardization)

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

∗ **datasets**
    Boston, 13
    Diabetes, 44

action, 5
action.dal_transform, 5
adjust_class_label, 6
adjust_data.frame, 7
adjust_factor, 7
adjust_matrix, 8
aggregation, 8
autoenc_base_e, 9
autoenc_base_ed, 10

bal_oversampling, 11
bal_subsampling, 12
Boston, 13

categ_mapping, 14
cla_bagging, 15
cla_boosting, 15
cla_dtree, 16
cla_glm, 17
cla_glmnet, 18
cla_knn, 19
cla_majority, 20
cla_mlp, 21
cla_multinom, 22
cla_nb, 23
cla_rf, 24
cla_rpart, 25
cla_svm, 26
cla_tune, 27
cla_xgboost, 28
classification, 29
clu_tune, 30
cluster, 30
cluster_cmeans, 31
cluster_dbscan, 32
cluster_gmm, 34
cluster_hclust, 35
cluster_kmeans, 36
cluster_pam, 37
clusterer, 39
cluutils, 40

dal_base, 41
dal_learner, 41
dal_transform, 42
dal_tune, 42
data_sample, 43
Diabetes, 44
discover, 45
dt_pca, 45

evaluate, 46

feature_generation, 47
feature_selection_corr, 48
feature_selection_fss, 49
feature_selection_info_gain, 50
feature_selection_lasso, 51
feature_selection_relief, 52
feature_selection_stepwise, 53
fit, 54
fit.cla_tune, 55
fit.cluster_dbscan, 55
fit_curvature_max, 56
fit_curvature_min, 56

hierarchy_cut, 57

imputation_predictive, 58
imputation_simple, 59
imputation_tree, 60
inverse_transform, 61

k_fold, 61

minmax, 62

na_removal, 63

outliers_boxplot, 64
outliers_gaussian, 65

pat_apriori, 66
pat_confidence_threshold, 67
pat_dara, 68
pat_filter_rules, 70
pat_prefix_span, 70
pat_rule_filter_dara, 72
pat_rule_filter_interest, 73
pat_rule_filter_none, 74
pat_rules_tidy, 75
pat_support_threshold, 75
pattern_miner, 76
patutils, 77
plot_bar, 78
plot_boxplot, 79
plot_boxplot_class, 80
plot_correlation, 81
plot_dendrogram, 82
plot_density, 83
plot_density_class, 84
plot_groupedbar, 85
plot_hist, 86
plot_lollipop, 87
plot_parallel, 88
plot_pieplot, 89
plot_pixel, 90
plot_points, 91
plot_radar, 92
plot_scatter, 93
plot_series, 94
plot_stackedbar, 95
plot_ts, 96
plot_ts_pred, 97
predictor, 98
predictor_prepare_fit, 98

reg_dtree, 99
reg_knn, 100
reg_lm, 101
reg_mlp, 102
reg_rf, 103
reg_svm, 104
reg_tune, 105
regression, 106

sample_balance, 107
sample_groups, 108
sample_random, 108
sample_simple, 109
sample_stratified, 110
select_hyper, 111
select_hyper.cla_tune, 112
set_params, 112
set_params.default, 113
smoothing, 113
smoothing_cluster, 114
smoothing_freq, 115
smoothing_inter, 116
smoothing_quantization, 116

train_test, 117
train_test_from_folds, 118
transform, 119
transform.pat_rule_filter, 120

zscore, 120