Package 'daltoolboxdp'

May 3, 2025

Title Python-Based Extensions for Data Analytics Workflows

Version 1.1.717

Description Provides Python-based extensions to enhance data analytics workflows, particularly for tasks involving data preprocessing and predictive modeling. Includes tools for data sampling, transformation, feature selection, balancing strategies (e.g., SMOTE), and model construction. These capabilities leverage Python libraries via the reticulate interface, enabling seamless integration with a broader machine learning ecosystem. Supports instance selection and hybrid workflows that combine R and Python functionalities for flexible and reproducible analytical pipelines. The architecture is inspired by the Experiment Lines approach, which promotes modularity, extensibility, and interoperability across tools. More information on Experiment Lines is available in Ogasawara et al. (2009) <doi:10.1007/978-3-642-02279-1_20>.

License MIT + file LICENSE

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

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

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports daltoolbox, leaps, FSelector, doBy, glmnet, smotefamily, reticulate, stats

Config/reticulate list(packages = list(list(package = ``scipy"), list(package = ``torch"), list(package = ``pandas"), list(package = ``numpy"), list(package = ``matplotlib"), list(package = ``scikit-learn")))

NeedsCompilation no

Author Eduardo Ogasawara [aut, ths, cre] (ORCID: <https://orcid.org/0000-0002-0466-0626>), Diego Salles [aut],

Eduardo Bezerra [ctb],
Federal Center for Technological Education of Rio de Janeiro (CEFET/RJ)
[cph] (organization: CEFET/RJ)

Maintainer Eduardo Ogasawara <eogasawara@ieee.org>

Repository CRAN

Date/Publication 2025-05-03 19:50:02 UTC

Contents

`bal_oversampling` *Oversampling*

Description

Oversampling balances the class distribution of a dataset by increasing the representation of the minority class in the dataset. It wraps the smotefamily library.

Usage

```
bal_oversampling(attribute)
```

Arguments

`attribute`	The class attribute to target balancing using oversampling.

Value

A `bal_oversampling` object.

Examples

```
data(iris)
mod_iris <- iris[c(1:50,51:71,101:111),]

bal <- bal_oversampling('Species')
bal <- daltoolbox::fit(bal, mod_iris)
adjust_iris <- daltoolbox::transform(bal, mod_iris)
table(adjust_iris$Species)
```

`bal_subsampling` *Subsampling*

Description

Subsampling balances the class distribution of a dataset by reducing the representation of the majority class in the dataset.

Usage

```
bal_subsampling(attribute)
```

Arguments

`attribute`	The class attribute to target balancing using subsampling

Value

A `bal_subsampling` object.

Examples

```
data(iris)
mod_iris <- iris[c(1:50,51:71,101:111),]

bal <- bal_subsampling('Species')
bal <- daltoolbox::fit(bal, mod_iris)
adjust_iris <- daltoolbox::transform(bal, mod_iris)
table(adjust_iris$Species)
```

fs *Feature Selection*

Description

Feature selection is a process of selecting a subset of relevant features from a larger set of features in a dataset for use in model training. The FeatureSelection class in R provides a framework for performing feature selection.

Usage

```
fs(attribute)
```

Arguments

attribute	The target variable.

Value

An instance of the FeatureSelection class.

Examples

```
#See ?fs_fss for an example of feature selection
```

fs_fss *Forward Stepwise Selection*

Description

Forward stepwise selection is a technique for feature selection in which attributes are added to a model one at a time based on their ability to improve the model's performance. It stops adding once the candidate addition does not significantly improve model adjustment. It wraps the leaps library.

Usage

```
fs_fss(attribute)
```

Arguments

attribute	The target variable.

Value

A fs_fss object.

Examples

```
data(iris)
myfeature <- daltoolbox::fit(fs_fss("Species"), iris)
data <- daltoolbox::transform(myfeature, iris)
head(data)
```

---

fs_ig *Information Gain*

Description

Information Gain is a feature selection technique based on information theory. It measures the information obtained for the target variable by knowing the presence or absence of a feature. It wraps the FSelector library.

Usage

```
fs_ig(attribute)
```

Arguments

attribute	The target variable.

Value

A fs_ig object.

Examples

```
data(iris)
myfeature <- daltoolbox::fit(fs_ig("Species"), iris)
data <- daltoolbox::transform(myfeature, iris)
head(data)
```

---

fs_lasso *Feature Selection using Lasso*

Description

Feature selection using Lasso regression is a technique for selecting a subset of relevant features. It wraps the glmnet library.

Usage

```
fs_lasso(attribute)
```

Arguments

attribute	The target variable.

Value

A fs_lasso object.

Examples

```
data(iris)
myfeature <- daltoolbox::fit(fs_lasso("Species"), iris)
data <- daltoolbox::transform(myfeature, iris)
head(data)
```

---

fs_relief *Relief*

Description

Feature selection using Relief is a technique for selecting a subset of relevant features. It calculates the relevance of a feature by considering the difference in feature values between nearest neighbors of the same and different classes. It wraps the FSelector library.

Usage

```
fs_relief(attribute)
```

Arguments

attribute	The target variable.

Value

A fs_relief object.

Examples

```
data(iris)
myfeature <- daltoolbox::fit(fs_relief("Species"), iris)
data <- daltoolbox::transform(myfeature, iris)
head(data)
```

skcla_gb *Gradient Boosting Classifier*

Description

Implements a classifier using the Gradient Boosting algorithm. This function wraps the GradientBoostingClassifier from Python's scikit-learn library.

Usage

```
skcla_gb(
  attribute,
  slevels,
  loss = "log_loss",
  learning_rate = 0.1,
  n_estimators = 100,
  subsample = 1,
  criterion = "friedman_mse",
  min_samples_split = 2,
  min_samples_leaf = 1,
  min_weight_fraction_leaf = 0,
  max_depth = 3,
  min_impurity_decrease = 0,
  init = NULL,
  random_state = NULL,
  max_features = NULL,
  verbose = 0,
  max_leaf_nodes = NULL,
  warm_start = FALSE,
  validation_fraction = 0.1,
  n_iter_no_change = NULL,
  tol = 1e-04,
  ccp_alpha = 0
)
```

Arguments

attribute	Target attribute name for model building
slevels	Possible values for the target classification
loss	Loss function to be optimized ('log_loss', 'exponential')
learning_rate	Learning rate that shrinks the contribution of each tree
n_estimators	Number of boosting stages to perform
subsample	Fraction of samples to be used for fitting the individual base learners
criterion	Function to measure the quality of a split

`min_samples_split`
: Minimum number of samples required to split an internal node

`min_samples_leaf`
: Minimum number of samples required to be at a leaf node

`min_weight_fraction_leaf`
: Minimum weighted fraction of the sum total of weights

`max_depth`
: Maximum depth of the individual regression estimators

`min_impurity_decrease`
: Minimum impurity decrease required for split

`init`
: Estimator object to initialize the model

`random_state`
: Random number generator seed

`max_features`
: Number of features to consider for best split

`verbose`
: Controls verbosity of the output

`max_leaf_nodes`
: Maximum number of leaf nodes

`warm_start`
: Whether to reuse solution of previous call

`validation_fraction`
: Proportion of training data to set aside for validation

`n_iter_no_change`
: Used to decide if early stopping will be used

`tol`
: Tolerance for early stopping

`ccp_alpha`
: Complexity parameter for cost-complexity pruning

Details

Tree Boosting

Value

A Gradient Boosting classifier object

`skcla_gb` object

Examples

```
#See an example of using `skcla_gb` at this
#https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/skcla_gb.md
```

skcla_knn *K-Nearest Neighbors Classifier*

Description

Implements classification using the K-Nearest Neighbors algorithm. This function wraps the KNeighborsClassifier from Python's scikit-learn library.

Usage

```
skcla_knn(
  attribute,
  slevels,
  n_neighbors = 5,
  weights = "uniform",
  algorithm = "auto",
  leaf_size = 30,
  p = 2,
  metric = "minkowski",
  metric_params = NULL,
  n_jobs = NULL
)
```

Arguments

attribute	Target attribute name for model building
slevels	List of possible values for classification target
n_neighbors	Number of neighbors to use for queries
weights	Weight function used in prediction ('uniform', 'distance')
algorithm	Algorithm used to compute nearest neighbors ('auto', 'ball_tree', 'kd_tree', 'brute')
leaf_size	Leaf size passed to BallTree or KDTree
p	Power parameter for the Minkowski metric
metric	Distance metric for the tree ('euclidean', 'manhattan', 'chebyshev', 'minkowski', etc.)
metric_params	Additional parameters for the metric function
n_jobs	Number of parallel jobs for neighbor searches

Details

K-Nearest Neighbors Classifier

Value

A K-Nearest Neighbors classifier object

skcla_knn object

Examples

```
#See an example of using `skcla_knn` at this
#https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/skcla_knn.md
```

---

skcla_mlp *Multi-layer Perceptron Classifier*

Description

Implements classification using Multi-layer Perceptron algorithm. This function wraps the MLP-Classifier from Python's scikit-learn library.

Usage

```
skcla_mlp(
  attribute,
  slevels,
  hidden_layer_sizes = c(100),
  activation = "relu",
  solver = "adam",
  alpha = 1e-04,
  batch_size = "auto",
  learning_rate = "constant",
  learning_rate_init = 0.001,
  power_t = 0.5,
  max_iter = 200,
  shuffle = TRUE,
  random_state = NULL,
  tol = 1e-04,
  verbose = FALSE,
  warm_start = FALSE,
  momentum = 0.9,
  nesterovs_momentum = TRUE,
  early_stopping = FALSE,
  validation_fraction = 0.1,
  beta_1 = 0.9,
  beta_2 = 0.999,
  epsilon = 1e-08,
  n_iter_no_change = 10,
  max_fun = 15000
)
```

Arguments

attribute	Target attribute name for model building
slevels	List of possible values for classification target

`hidden_layer_sizes`
: Number of neurons in each hidden layer

`activation`
: Activation function for hidden layer ('identity', 'logistic', 'tanh', 'relu')

`solver`
: The solver for weight optimization ('lbfgs', 'sgd', 'adam')

`alpha`
: L2 penalty (regularization term) parameter

`batch_size`
: Size of minibatches for stochastic optimizers

`learning_rate`
: Learning rate schedule for weight updates

`learning_rate_init`
: Initial learning rate used

`power_t`
: Exponent for inverse scaling learning rate

`max_iter`
: Maximum number of iterations

`shuffle`
: Whether to shuffle samples in each iteration

`random_state`
: Seed for random number generation

`tol`
: Tolerance for optimization

`verbose`
: Whether to print progress messages to stdout

`warm_start`
: Whether to reuse previous solution

`momentum`
: Momentum for gradient descent update

`nesterovs_momentum`
: Whether to use Nesterov's momentum

`early_stopping`
: Whether to use early stopping

`validation_fraction`
: Proportion of training data for validation

`beta_1`
: Exponential decay rate for estimates of first moment vector

`beta_2`
: Exponential decay rate for estimates of second moment vector

`epsilon`
: Value for numerical stability in adam

`n_iter_no_change`
: Maximum number of epochs to not meet tol improvement

`max_fun`
: Maximum number of loss function calls

Details

Neural Network Classifier

Value

A Multi-layer Perceptron classifier object

`skcla_mlp` object

Examples

```
#See an example of using `skcla_mlp` at this
#https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/skcla_mlp.md
```

skcla_nb *Gaussian Naive Bayes Classifier*

Description

Implements classification using the Gaussian Naive Bayes algorithm. This function wraps the GaussianNB from Python's scikit-learn library.

Usage

```
skcla_nb(attribute, slevels, var_smoothing = 1e-09, priors = NULL)
```

Arguments

attribute	Target attribute name for model building
slevels	List of possible values for classification target
var_smoothing	Portion of the largest variance of all features that is added to variances
priors	Prior probabilities of the classes. If specified must be a list of length n_classes

Details

Naive Bayes Classifier

Value

A Naive Bayes classifier object

skcla_nb object

Examples

```
#See an example of using `skcla_nb` at this
#https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/skcla_nb.md
```

skcla_rf *Random Forest Classifier*

Description

Implements classification using Random Forest algorithm. This function wraps the RandomForestClassifier from Python's scikit-learn library.

Usage

```
skcla_rf(
  attribute,
  slevels,
  n_estimators = 100,
  criterion = "gini",
  max_depth = NULL,
  min_samples_split = 2,
  min_samples_leaf = 1,
  min_weight_fraction_leaf = 0,
  max_features = "sqrt",
  max_leaf_nodes = NULL,
  min_impurity_decrease = 0,
  bootstrap = TRUE,
  oob_score = FALSE,
  n_jobs = NULL,
  random_state = NULL,
  verbose = 0,
  warm_start = FALSE,
  class_weight = NULL,
  ccp_alpha = 0,
  max_samples = NULL,
  monotonic_cst = NULL
)
```

Arguments

attribute	Target attribute name for model building
slevels	List of possible values for classification target
n_estimators	Number of trees in random forest
criterion	Function name for measuring split quality
max_depth	Maximum tree depth value
min_samples_split	Minimum samples needed for internal node split
min_samples_leaf	Minimum samples needed at leaf node
min_weight_fraction_leaf	Minimum weighted fraction value
max_features	Number of features to consider for best split
max_leaf_nodes	Maximum number of leaf nodes
min_impurity_decrease	Minimum impurity decrease needed for split
bootstrap	Whether to use bootstrap samples
oob_score	Whether to use out-of-bag samples
n_jobs	Number of parallel jobs

random_state	Seed for random number generation
verbose	Whether to enable verbose output
warm_start	Whether to reuse previous solution
class_weight	Weights associated with classes
ccp_alpha	Complexity parameter value for pruning
max_samples	Number of samples for training estimators
monotonic_cst	Monotonicity constraints for features

Details

Tree Ensemble

Value

A Random Forest classifier object

skcla_rf object

Examples

```
#See an example of using `skcla_rf` at this
#https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/skcla_rf.md
```

---

skcla_svc *Support Vector Machine Classification*

Description

Implements classification using Support Vector Machine (SVM) algorithm. This function wraps the SVC classifier from Python's scikit-learn library.

Usage

```
skcla_svc(
  attribute,
  slevels,
  kernel = "rbf",
  degree = 3,
  gamma = "scale",
  coef0 = 0,
  tol = 0.001,
  C = 1,
  shrinking = TRUE,
  probability = FALSE,
  cache_size = 200,
  class_weight = NULL,
  verbose = FALSE,
```

```
  max_iter = -1,
  decision_function_shape = "ovr",
  break_ties = FALSE,
  random_state = NULL
)
```

Arguments

attribute	Target attribute name for model building
slevels	List of possible values for classification target
kernel	Kernel function type ('linear', 'poly', 'rbf', 'sigmoid')
degree	Polynomial degree when using 'poly' kernel
gamma	Kernel coefficient value
coef0	Independent term value in kernel function
tol	Tolerance value for stopping criterion
C	Regularization strength parameter
shrinking	Whether to use shrinking heuristic
probability	Whether to enable probability estimates
cache_size	Kernel cache size value in MB
class_weight	Weights associated with classes
verbose	Whether to enable verbose output
max_iter	Maximum number of iterations
decision_function_shape	Shape of decision function ('ovo', 'ovr')
break_ties	Whether to break tie decisions
random_state	Seed for random number generation

Details

SVM Classifier

Value

An SVM classifier object

skcla_svc object

Examples

```
#See an example of using `skcla_svc` at this
#https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/cla_svm.md
```

Index