Package 'daltoolboxdp'

March 11, 2026

Title Deep Python Extensions for 'daltoolbox'

Version 1.2.747

Description Extends 'daltoolbox' with Python-backed components for deep learning, scikit-learn classification, and time-series forecasting through 'reticulate'. The package provides objects that follow the 'daltoolbox' architecture while delegating model creation, fitting, encoding, and prediction to Python libraries such as 'torch' and 'scikit-learn'. In the package name, 'dp' stands for 'Deep Python'. The overall workflow is inspired by the Experiment Lines approach described 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 7.3.3

Depends R (>= 4.1.0)

Imports tspredit, daltoolbox, reticulate

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], Janio Lima [aut], Lucas Tavares [aut], Eduardo Bezerra [ctb], CEFET/RJ [cph]

Maintainer Eduardo Ogasawara <eogasawara@ieee.org>

Repository CRAN

Date/Publication 2026-03-11 00:20:22 UTC

Contents

`autoenc_adv_e` *Adversarial Autoencoder - Encode*

Description

Creates a deep learning adversarial autoencoder (AAE) to encode sequences of observations. Wraps a PyTorch implementation.

Usage

```
autoenc_adv_e(
  input_size,
  encoding_size,
  batch_size = 350,
  num_epochs = 1000,
  learning_rate = 0.001
)
```

Arguments

input_size	input size
encoding_size	encoding size
batch_size	size for batch learning
num_epochs	number of epochs for training
learning_rate	learning rate

Details

Adversarial autoencoders regularize the latent space using an adversarial training objective, encouraging the aggregated posterior to match a prior distribution. This can lead to more structured latent representations.

Value

A autoenc_adv_e object.

References

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., & Frey, B. (2016). Adversarial Autoencoders.

Examples

```
## Not run:
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)
ae <- autoenc_adv_e(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)       # adversarially-regularized encoder
Z  <- daltoolbox::transform(ae, X) # encodings
dim(Z)

## End(Not run)

# See a complete example:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_adv_e.md
```

---

autoenc_adv_ed *Adversarial Autoencoder - Encode-Decode*

Description

Creates a deep learning adversarial autoencoder (AAE) that encodes and decodes sequences of observations. Wraps a PyTorch implementation.

Usage

```
autoenc_adv_ed(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001
)
```

Arguments

input_size	input size
encoding_size	encoding size
batch_size	size for batch learning
num_epochs	number of epochs for training
learning_rate	learning rate

Details

The adversarial loss constrains the latent distribution, improving sampling and reconstruction quality in some setups compared to a vanilla AE.

Value

A autoenc_adv_ed object.

References

Makhzani, A. et al. (2016). Adversarial Autoencoders.

Examples

```
## Not run:
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)
ae <- autoenc_adv_ed(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)
X_hat <- daltoolbox::transform(ae, X)  # reconstructions
mean((X - X_hat)^2)

## End(Not run)

# More details:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_adv_ed.md
```

autoenc_conv_e *Convolutional Autoencoder - Encode*

Description

Creates a deep learning convolutional autoencoder (ConvAE) to encode sequences of observations. Wraps a PyTorch implementation.

Usage

```
autoenc_conv_e(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001
)
```

Arguments

input_size	input size
encoding_size	encoding size
batch_size	size for batch learning
num_epochs	number of epochs for training
learning_rate	learning rate

Value

A autoenc_conv_e object.

References

Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011). Stacked Convolutional Auto-Encoders.

Examples

```
## Not run:
# Conv1D-based encoder expects data reshaped internally to (n, input_size, 1)
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)
ae <- autoenc_conv_e(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)
Z  <- daltoolbox::transform(ae, X)   # 50 x 5 encodings

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/transf/autoenc_conv_e.md
```

autoenc_conv_ed *Convolutional Autoencoder - Encode-Decode*

Description

Creates a deep learning convolutional autoencoder (ConvAE) to encode and decode sequences of observations. Wraps a PyTorch implementation.

Usage

```
autoenc_conv_ed(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001
)
```

Arguments

input_size	input size
encoding_size	encoding size
batch_size	size for batch learning
num_epochs	number of epochs for training
learning_rate	learning rate

Value

A autoenc_conv_ed object.

Examples

```
## Not run:
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)
ae <- autoenc_conv_ed(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)
X_hat <- daltoolbox::transform(ae, X)  # same dims as X
mean((X - X_hat)^2)

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/transf/autoenc_conv_ed.md
```

autoenc_denoise_e *Denoising Autoencoder - Encode*

Description

Creates a deep learning denoising autoencoder (DAE) to encode sequences while learning robustness to noise. Wraps a PyTorch implementation.

Usage

```
autoenc_denoise_e(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001,
  noise_factor = 0.3
)
```

Arguments

input_size	input size
encoding_size	encoding size
batch_size	size for batch learning
num_epochs	number of epochs for training
learning_rate	learning rate
noise_factor	Numeric. Standard deviation (scale) of the noise added during training.

Value

A autoenc_denoise_e object.

References

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders.

Examples

```
## Not run:
# 1) Prepare data
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)

# 2) Fit denoising encoder (higher noise_factor = stronger noise during training)
ae <- autoenc_denoise_e(input_size = 20, encoding_size = 5, noise_factor = 0.2, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)
```

```
# 3) Obtain latent encodings
Z <- daltoolbox::transform(ae, X)
dim(Z)

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_denoise_e.md
```

`autoenc_denoise_ed` *Denoising Autoencoder - Encode-Decode*

Description

Creates a deep learning denoising autoencoder (DAE) that encodes and decodes sequences, learning robustness to input noise. Wraps a PyTorch implementation.

Usage

```
autoenc_denoise_ed(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001,
  noise_factor = 0.3
)
```

Arguments

`input_size`	input size
`encoding_size`	encoding size
`batch_size`	size for batch learning
`num_epochs`	number of epochs for training
`learning_rate`	learning rate
`noise_factor`	Numeric. Standard deviation (scale) of the noise added during training.

Value

A `autoenc_denoise_ed` object.

References

Vincent, P. et al. (2008). Extracting and Composing Robust Features with Denoising Autoencoders.

Examples

```
## Not run:
# 1) Prepare data
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)

# 2) Fit denoising autoencoder (encode-decode)
ae <- autoenc_denoise_ed(input_size = 20, encoding_size = 5, noise_factor = 0.2, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)

# 3) Reconstruct inputs and compute error
X_hat <- daltoolbox::transform(ae, X)
mean((X - X_hat)^2)

## End(Not run)

# More examples:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_denoise_ed.md
```

---

autoenc_e *Autoencoder - Encode*

Description

Creates a deep learning autoencoder that learns a latent representation (encoding) for a sequence of observations. Wraps a PyTorch implementation via the `reticulate` bridge.

Usage

```
autoenc_e(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001
)
```

Arguments

`input_size`	Integer. Number of input features per observation.
`encoding_size`	Integer. Size of the latent (bottleneck) representation.
`batch_size`	Integer. Mini-batch size used during training. Default is 32.
`num_epochs`	Integer. Maximum number of training epochs. Default is 1000.
`learning_rate`	Numeric. Optimizer learning rate. Default is 0.001.

Details

This encoder provides dimensionality reduction by training a neural network to compress inputs into a lower-dimensional bottleneck. The learned encoding can be used for downstream tasks such as clustering, visualization, or as features for predictive models.

Value

A autoenc_e object.

References

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.

Examples

```
## Not run:
# Requirements: Python with torch installed and reticulate configured.
set.seed(123)

# 1) Create a toy dataset with 100 samples and 20 features
X <- matrix(rnorm(2000), nrow = 100, ncol = 20)

# 2) Create and fit an encoder (5-D bottleneck)
ae <- autoenc_e(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)

# 3) Transform data to latent space
Z <- daltoolbox::transform(ae, X)   # matrix with dimensions 100 x 5
dim(Z)                              # c(100, 5)

## End(Not run)

# See a more complete example at:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_e.md
```

---

autoenc_ed *Autoencoder - Encode-Decode*

Description

Creates a deep learning autoencoder that encodes and decodes sequences of observations. Wraps a PyTorch implementation via reticulate.

Usage

```
autoenc_ed(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001
)
```

Arguments

input_size	Integer. Number of input features per observation.
encoding_size	Integer. Size of the latent (bottleneck) representation.
batch_size	Integer. Mini-batch size used during training. Default is 32.
num_epochs	Integer. Maximum number of training epochs. Default is 1000.
learning_rate	Numeric. Optimizer learning rate. Default is 0.001.

Details

This variant both compresses inputs into a latent representation and reconstructs them back to input space, allowing the reconstruction error to be used as a quality metric or for anomaly detection.

Value

A autoenc_ed object.

References

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.

Examples

```
## Not run:
# Requirements: Python with torch installed and reticulate configured.

# 1) Create sample data (50 x 20)
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)

# 2) Fit encode-decode autoencoder (5-D bottleneck)
ae <- autoenc_ed(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)

# 3) Reconstruct inputs and inspect reconstruction error
X_hat <- daltoolbox::transform(ae, X)  # same dimensions as X
mean((X - X_hat)^2)                    # simple MSE across all entries

## End(Not run)

# More examples:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_ed.md
```

`autoenc_lstm_e` *LSTM Autoencoder - Encode*

Description

Creates a deep learning LSTM-based autoencoder to encode sequences of observations. Wraps a PyTorch implementation.

Usage

```
autoenc_lstm_e(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 50,
  learning_rate = 0.001
)
```

Arguments

`input_size`	input size
`encoding_size`	encoding size
`batch_size`	size for batch learning
`num_epochs`	number of epochs for training
`learning_rate`	learning rate

Value

A `autoenc_lstm_e` object.

References

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory.

Examples

```
## Not run:
# LSTM-based encoder over sequences stored as rows
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)
ae <- autoenc_lstm_e(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)
Z  <- daltoolbox::transform(ae, X)  # 50 x 5
dim(Z)

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_lstm_e.md
```

autoenc_lstm_ed *LSTM Autoencoder - Encode-Decode*

Description

Creates a deep learning LSTM-based autoencoder that encodes and decodes sequences of observations. Wraps a PyTorch implementation via `reticulate`.

Usage

```
autoenc_lstm_ed(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 50,
  learning_rate = 0.001
)
```

Arguments

`input_size`	Integer. Number of input features per observation.
`encoding_size`	Integer. Size of the latent (bottleneck) representation.
`batch_size`	Integer. Mini-batch size used during training. Default is 32.
`num_epochs`	Integer. Maximum number of training epochs. Default is 50.
`learning_rate`	Numeric. Optimizer learning rate. Default is 0.001.

Value

A `autoenc_lstm_ed` object.

References

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory.

Examples

```
## Not run:
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)
ae <- autoenc_lstm_ed(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)
X_hat <- daltoolbox::transform(ae, X)  # reconstructions
mean((X - X_hat)^2)

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_lstm_ed.md
```

autoenc_stacked_e *Stacked Autoencoder - Encode*

Description

Creates a deep learning stacked autoencoder to encode sequences of observations. The autoencoder layers are based on DAL Toolbox vanilla autoencoder and wrap a PyTorch implementation.

Usage

```
autoenc_stacked_e(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001,
  k = 3
)
```

Arguments

input_size	input size
encoding_size	encoding size
batch_size	size for batch learning
num_epochs	number of epochs for training
learning_rate	learning rate
k	Integer. Number of autoencoder layers in the stack.

Value

A autoenc_stacked_e object.

References

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion.

Examples

```
## Not run:
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)
ae <- autoenc_stacked_e(input_size = 20, encoding_size = 5, k = 3, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)
Z  <- daltoolbox::transform(ae, X)

## End(Not run)
```

```
# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_stacked_e.md
```

`autoenc_stacked_ed` *Stacked Autoencoder - Encode-Decode*

Description

Creates a deep learning stacked autoencoder to encode and decode sequences of observations. The layers are based on DAL Toolbox vanilla autoencoder and wrap a PyTorch implementation.

Usage

```
autoenc_stacked_ed(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001,
  k = 3
)
```

Arguments

`input_size`	input size
`encoding_size`	encoding size
`batch_size`	size for batch learning
`num_epochs`	number of epochs for training
`learning_rate`	learning rate
`k`	Integer. Number of autoencoder layers in the stack.

Value

A `autoenc_stacked_ed` object.

References

Vincent, P. et al. (2010). Stacked Denoising Autoencoders.

Examples

```
## Not run:
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)
ae <- autoenc_stacked_ed(input_size = 20, encoding_size = 5, k = 3, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)
X_hat <- daltoolbox::transform(ae, X)

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_stacked_e.md
```

---

autoenc_variational_e *Variational Autoencoder - Encode*

Description

Creates a deep learning variational autoencoder (VAE) to encode sequences of observations. Wraps a PyTorch implementation.

Usage

```
autoenc_variational_e(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001
)
```

Arguments

input_size	input size
encoding_size	encoding size
batch_size	size for batch learning
num_epochs	number of epochs for training
learning_rate	learning rate

Value

A autoenc_variational_e object.

References

Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes.

Examples

```
## Not run:
# Requirements: Python with torch installed and reticulate configured.

# 1) Create sample data
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)

# 2) Fit VAE encoder
ae <- autoenc_variational_e(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)

# 3) Transform to latent encodings
#    Note: the underlying Python returns [mean | var] concatenated; depending on
#    the implementation, you may receive 2*encoding_size columns.
Z <- daltoolbox::transform(ae, X)
dim(Z)

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_variational_e.md
```

---

autoenc_variational_ed

Variational Autoencoder - Encode-Decode

Description

Creates a deep learning variational autoencoder (VAE) that encodes and decodes sequences of observations. Wraps a PyTorch implementation.

Usage

```
autoenc_variational_ed(
  input_size,
  encoding_size,
  batch_size = 32,
  num_epochs = 1000,
  learning_rate = 0.001
)
```

Arguments

input_size	input size
encoding_size	encoding size
batch_size	size for batch learning
num_epochs	number of epochs for training
learning_rate	learning rate

Value

A autoenc_variational_ed object.

References

Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes.

Examples

```
## Not run:
# Requirements: Python with torch installed and reticulate configured.

# 1) Sample data
X <- matrix(rnorm(1000), nrow = 50, ncol = 20)

# 2) Fit VAE encode-decode
ae <- autoenc_variational_ed(input_size = 20, encoding_size = 5, num_epochs = 50)
ae <- daltoolbox::fit(ae, X)

# 3) Reconstruct inputs
X_hat <- daltoolbox::transform(ae, X)
mean((X - X_hat)^2)

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/autoencoder/autoenc_variational_ed.md
```

---

skcla_gb *Gradient Boosting Classifier*

Description

Implements a classifier using the Gradient Boosting algorithm. Wraps scikit-learn's GradientBoostingClassifier through reticulate.

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 skcla_gb classifier object.

References

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine.

Examples

```
## Not run:
data(iris)
clf <- skcla_gb(attribute = 'Species', slevels = levels(iris$Species), n_estimators = 150)
clf <- daltoolbox::fit(clf, iris)
pred <- predict(clf, iris)
table(pred, iris$Species)

## End(Not run)

# More examples:
# 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. Wraps scikit-learn's KNeighborsClassifier through reticulate.

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 skcla_knn classifier object.

References

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification.

Examples

```
## Not run:
data(iris)

# 1) Initialize KNN (k=7) with target attribute + levels
clf <- skcla_knn(attribute = 'Species', slevels = levels(iris$Species), n_neighbors = 7)

# 2) Fit and predict; factors are handled internally
clf <- daltoolbox::fit(clf, iris)
pred <- predict(clf, iris)
table(pred, iris$Species)

## End(Not run)

# More examples:
# https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/skcla_knn.md
```

skcla_mlp *Multi-layer Perceptron Classifier*

Description

Implements classification using a multi-layer perceptron (MLP). Wraps scikit-learn's `MLPClassifier` through `reticulate`.

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 `skcla_mlp` classifier object.

References

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

Examples

```
## Not run:
data(iris)

# 1) Define MLP architecture (two hidden layers)
clf <- skcla_mlp(attribute = 'Species', slevels = levels(iris$Species),
```

```
                hidden_layer_sizes = c(32, 16))

# 2) Fit and predict
clf <- daltoolbox::fit(clf, iris)
pred <- predict(clf, iris)
table(pred, iris$Species)

## End(Not run)

# More examples:
# https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/skcla_mlp.md
```

---

skcla_nb *Gaussian Naive Bayes Classifier*

Description

Implements classification using Gaussian Naive Bayes. Wraps scikit-learn's `GaussianNB` through `reticulate`.

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 `skcla_nb` classifier object.

References

Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. (Gaussian Naive Bayes)

Examples

```
## Not run:
data(iris)

# Gaussian Naive Bayes for multi-class iris
clf <- skcla_nb(attribute = 'Species', slevels = levels(iris$Species))
clf <- daltoolbox::fit(clf, iris)
pred <- predict(clf, iris)
table(pred, iris$Species)

## End(Not run)

# More examples:
# https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/skcla_nb.md
```

---

skcla_rf *Random Forest Classifier*

Description

Implements classification using the Random Forest algorithm. Wraps scikit-learn's `RandomForestClassifier` through `reticulate`.

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 `skcla_rf` classifier object.

References

Breiman, L. (2001). Random Forests. Machine Learning.

Examples

```
## Not run:
data(iris)

# 1) Define classifier with target attribute and its levels
clf <- skcla_rf(attribute = 'Species', slevels = levels(iris$Species), n_estimators = 200)

# 2) Fit and predict
clf <- daltoolbox::fit(clf, iris)
pred <- predict(clf, iris)   # wrapper drops target column internally
table(pred, iris$Species)

## End(Not run)

# More examples:
# 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 machines. Wraps scikit-learn's SVC through `reticulate`.

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

A skcla_svc classifier object.

References

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks.

Examples

```
## Not run:
data(iris)

# 1) Create SVM classifier (RBF kernel)
clf <- skcla_svc(attribute = 'Species', slevels = levels(iris$Species), kernel = 'rbf', C = 1)

# 2) Fit and predict
clf <- daltoolbox::fit(clf, iris)
pred <- predict(clf, iris)
table(pred, iris$Species)
```

```
## End(Not run)

# More examples:
# https://github.com/cefet-rj-dal/daltoolboxdp/blob/main/examples/cla_svm.md
```

`ts_conv1d` *Conv1D*

Description

Time series forecaster using a 1D convolutional neural network. Wraps a PyTorch implementation via `reticulate`.

Usage

```
ts_conv1d(preprocess = NA, input_size = NA, epochs = 10000L)
```

Arguments

`preprocess`	Optional preprocessing/normalization object.
`input_size`	Integer. Number of lagged inputs per training example.
`epochs`	Integer. Maximum number of training epochs.

Value

A `ts_conv1d` object.

References

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. Bai, S., Kolter, J. Z., & Koltun, V. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.

Examples

```
## Not run:
# Conv1D forecaster expects features + 't0' target internally; the R wrapper
# builds the required data frame when you call do_fit/do_predict via tspredit.

tsf <- ts_conv1d(input_size = 12, epochs = 1000L)
# model <- daltoolbox::fit(tsf, your_data)

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/timeseries/ts_conv1d.md
```

ts_lstm *LSTM*

Description

Time series forecaster using an LSTM neural network. Wraps a PyTorch implementation via `reticulate`.

Usage

```
ts_lstm(preprocess = NA, input_size = NA, epochs = 10000L)
```

Arguments

`preprocess`	Optional preprocessing/normalization object.
`input_size`	Integer. Number of lagged inputs per training example.
`epochs`	Integer. Maximum number of training epochs.

Value

A `ts_lstm` object.

References

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory.

Examples

```
## Not run:
# LSTM forecaster expects a frame where 't0' is the target during fitting.
# The R wrapper constructs it from (x, y), so you usually call do_fit via tspredit.

# Minimal construction (see vignette for full workflow)
tsf <- ts_lstm(input_size = 12, epochs = 1000L)
# model <- daltoolbox::fit(tsf, your_data)  # delegated to tspredit

## End(Not run)

# See:
# https://github.com/cefet-rj-dal/daltoolbox/blob/main/timeseries/ts_lstm.md
```

Index