

Package: mlsauce (via r-universe)

September 27, 2026

Title Miscellaneous Statistical/Machine Learning stuff

Version 0.39.1

Date 2026-06-29

Author T. Moudiki

Maintainer T. Moudiki <thierry.moudiki@gmail.com>

Description Miscellaneous Statistical/Machine Learning stuff.

License BSD_3_clause Clear + file LICENSE

Imports reticulate

Suggests Rcpp, misc, MASS

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/reticulate required-modules: scikit-learn, numpy, scipy, pandas

Remotes thierrymoudiki/misc

Config/pak/sysreqs libpng-dev python3

Repository <https://techtonique.r-universe.dev>

Date/Publication 2026-06-29 19:37:48 UTC

RemoteUrl https://github.com/Techtonique/mlsauce_r

RemoteRef HEAD

RemoteSha 6e51d91e4bc13f63b43caa952b743e9bdf5166f1

Contents

AdaOpt	2
GenericBoostingClassifier	4
GenericBoostingRegressor	6
LassoRegressor	8
LazyBoostingClassifier	9

LazyBoostingRegressor	11
LSBoostClassifier	12
LSBoostRegressor	14
RidgeRegressor	16
RVFLJackknifePlus	17
StumpClassifier	19
Index	20

AdaOpt	<i>AdaOpt classifier</i>
--------	--------------------------

Description

AdaOpt classifier

Usage

```
AdaOpt(  
  n_iterations = 50L,  
  learning_rate = 0.3,  
  reg_lambda = 0.1,  
  reg_alpha = 0.5,  
  eta = 0.01,  
  gamma = 0.01,  
  k = 3L,  
  tolerance = 0,  
  n_clusters = 0,  
  batch_size = 100L,  
  row_sample = 1,  
  type_dist = "euclidean-f",  
  cache = TRUE,  
  n_clusters_input = 0,  
  clustering_method = "kmeans",  
  cluster_scaling = "standard",  
  seed = 123L,  
  venv_path = "./venv",  
  ...  
)
```

Arguments

- | | |
|---------------|---|
| n_iterations | number of iterations of the optimizer at training time |
| learning_rate | controls the speed of the optimizer at training time |
| reg_lambda | L2 regularization parameter for successive errors in the optimizer (at training time) |
| reg_alpha | L1 regularization parameter for successive errors in the optimizer (at training time) |

eta	controls the slope in gradient descent (at training time)
gamma	controls the step size in gradient descent (at training time)
k	number of nearest neighbors selected at test time for classification
tolerance	controls early stopping in gradient descent (at training time)
n_clusters	number of clusters, if MiniBatch k-means is used at test time (for faster prediction)
batch_size	size of the batch, if MiniBatch k-means is used at test time (for faster prediction)
row_sample	percentage of rows chosen from training set (by stratified subsampling, for faster prediction)
type_dist	distance used for finding the nearest neighbors; currently euclidean-f (euclidean distances calculated as whole), euclidean (euclidean distances calculated row by row), cosine (cosine distance)
cache	if the nearest neighbors are cached or not, for faster retrieval in subsequent calls
n_clusters_input	number of clusters a priori on input data
clustering_method	either "kmeans" or "gmm" (Gaussian mixture)
cluster_scaling	either 'standard', 'minmax', 'robust'
seed	reproducibility seed for initial weak learner and clustering

Value

An object of class AdaOpt

Examples

```
## Not run:
library(datasets)

X <- as.matrix(iris[, 1:4])
y <- as.integer(iris[, 5]) - 1L

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(iris[train_index, 1:4])
y_train <- as.integer(iris[train_index, 5]) - 1L
X_test <- as.matrix(iris[test_index, 1:4])
y_test <- as.integer(iris[test_index, 5]) - 1L

obj <- mlsauce::AdaOpt()

print(obj$get_params())
```

```
obj$fit(X_train, y_train)

print(obj$score(X_test, y_test))

## End(Not run)
```

GenericBoostingClassifier

GenericBoosting classifier

Description

GenericBoosting classifier

Usage

```
GenericBoostingClassifier(
  base_model = NULL,
  n_estimators = 100L,
  learning_rate = 0.1,
  n_hidden_features = 5L,
  row_sample = 1,
  col_sample = 1,
  dropout = 0,
  tolerance = 1e-04,
  direct_link = 1L,
  verbose = 1L,
  seed = 123L,
  activation = "relu",
  n_clusters = 0,
  clustering_method = "kmeans",
  cluster_scaling = "standard",
  degree = NULL,
  weights_distr = "uniform",
  venv_path = "./venv",
  ...
)
```

Arguments

base_model: object, base model to be boosted.

n_estimators: int, number of boosting iterations.

learning_rate: float, controls the learning speed at training time.

n_hidden_features:
int

number of nodes in successive hidden layers.
 row_sample: float, percentage of rows chosen from the training set.
 col_sample: float, percentage of columns chosen from the training set.
 dropout: float, percentage of nodes dropped from the training set.
 tolerance: float, controls early stopping in gradient descent (at training time).
 direct_link: bool, indicates whether the original features are included (True) in model's fitting or not (False).
 verbose: int, progress bar (yes = 1) or not (no = 0) (currently).
 seed: int, reproducibility seed for nodes_sim=='uniform', clustering and dropout.
 activation: str, activation function: currently 'relu', 'relu6', 'sigmoid', 'tanh'
 n_clusters: int, number of clusters for clustering.
 clustering_method: str, clustering method: currently 'kmeans', 'gmm' (Gaussian Mixture Model)
 cluster_scaling: str, scaling method for clustering: currently 'standard', 'minmax', 'robust'
 degree: int, degree of polynomial interactions features.
 weights_distr: str, distribution of weights for the hidden layer currently 'uniform', 'gaussian'

Value

An object of class GenericBoostingClassifier

Examples

```

# See https://thierrymoudiki.github.io/blog/2024/10/14/r/genericboosting-r for advanced examples
## Not run:
library(datasets)

X <- as.matrix(iris[, 1:4])
y <- as.integer(iris[, 5]) - 1L

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(X[train_index, ])
y_train <- as.integer(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- as.integer(y[test_index])

sklearn <- nnetsauce::get_sklearn()

obj2 <- sklearn$tree$DecisionTreeRegressor()

obj <- mlsauce::GenericBoostingClassifier(obj2)

```

```
print(obj$get_params())

obj$fit(X_train, y_train)

print(obj$score(X_test, y_test))

sklearn <- nnetsauce::get_sklearn()

obj2 <- sklearn$linear_model$LinearRegression()

obj <- mlsauce::GenericBoostingClassifier(obj2)

print(obj$get_params())

obj$fit(X_train, y_train)

print(obj$score(X_test, y_test))

## End(Not run)
```

GenericBoostingRegressor

GenericBoosting Regressor

Description

GenericBoosting Regressor

Usage

```
GenericBoostingRegressor(  
  base_model = NULL,  
  n_estimators = 100L,  
  learning_rate = 0.1,  
  n_hidden_features = 5L,  
  row_sample = 1,  
  col_sample = 1,  
  dropout = 0,  
  tolerance = 1e-04,  
  direct_link = 1L,  
  verbose = 1L,  
  seed = 123L,  
  activation = "relu",  
  n_clusters = 0,  
  clustering_method = "kmeans",  
  cluster_scaling = "standard",  
  degree = NULL,
```

```

    weights_distr = "uniform",
    venv_path = "./venv",
    ...
)

```

Arguments

base_model: object, base model to be boosted.
n_estimators: int, number of boosting iterations.
learning_rate: float, controls the learning speed at training time.
n_hidden_features: int
 number of nodes in successive hidden layers.
row_sample: float, percentage of rows chosen from the training set.
col_sample: float, percentage of columns chosen from the training set.
dropout: float, percentage of nodes dropped from the training set.
tolerance: float, controls early stopping in gradient descent (at training time).
direct_link: bool, indicates whether the original features are included (True) in model's fitting or not (False).
verbose: int, progress bar (yes = 1) or not (no = 0) (currently).
seed: int, reproducibility seed for nodes_sim=='uniform', clustering and dropout.
activation: str, activation function: currently 'relu', 'relu6', 'sigmoid', 'tanh'
n_clusters: int, number of clusters for clustering.
clustering_method: str, clustering method: currently 'kmeans', 'gmm' (Gaussian Mixture Model)
cluster_scaling: str, scaling method for clustering: currently 'standard', 'minmax', 'robust'
degree: int, degree of polynomial interactions features.
weights_distr: str, distribution of weights for the hidden layer currently 'uniform', 'gaussian'

Value

An object of class GenericBoostingRegressor

Examples

```

# See https://thierrymoudiki.github.io/blog/2024/10/14/r/genericboosting-r for advanced examples

## Not run:
library(datasets)

X <- as.matrix(datasets::mtcars[, -1])
y <- as.integer(datasets::mtcars[, 1])

n <- dim(X)[1]

```

```

p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(X[train_index, ])
y_train <- as.double(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- as.double(y[test_index])

sklearn <- nnetsauce::get_sklearn()

obj2 <- sklearn$linear_model$LinearRegression()

obj <- mlsauce::GenericBoostingRegressor(obj2)

print(obj$get_params())

obj$fit(X_train, y_train)

print(sqrt(mean((obj$predict(X_test) - y_test)**2)))

## End(Not run)

```

LassoRegressor

Lasso regressor

Description

Lasso regressor

Usage

```

LassoRegressor(
  reg_lambda = 0.1,
  max_iter = 10L,
  tol = 0.001,
  venv_path = "./venv",
  ...
)

```

Arguments

<code>reg_lambda</code>	L1 regularization parameter
<code>max_iter</code>	number of iterations of lasso shooting algorithm.
<code>tol</code>	tolerance for convergence of lasso shooting algorithm.

Value

An object of class Lasso

Examples

```
## Not run:
library(datasets)

X <- as.matrix(datasets::mtcars[, -1])
y <- as.integer(datasets::mtcars[, 1])

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(X[train_index, ])
y_train <- as.double(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- as.double(y[test_index])

obj <- mlsauce::LassoRegressor()

print(obj$get_params())

obj$fit(X_train, y_train)

print(obj$score(X_test, y_test))

## End(Not run)
```

LazyBoostingClassifier

LazyBoostingClassifier

Description

Lazy Generic Boosting Classifier (AutoML Hold-out set validation)

Usage

```
LazyBoostingClassifier(
  verbose = 0,
  ignore_warnings = TRUE,
  custom_metric = NULL,
  predictions = FALSE,
  sort_by = "Accuracy",
  random_state = 42L,
```

```

    estimators = "all",
    preprocess = FALSE,
    n_jobs = NULL,
    venv_path = "./venv",
    ...
)

```

Arguments

verbose: int, progress bar (yes = 1) or not (no = 0) (currently).
ignore_warnings: bool, ignore warnings.
custom_metric: function, custom metric.
predictions: bool, return predictions.
sort_by: str, sort by metric.
random_state: int, random state.
estimators: str, estimators to use. List of names for custom, or just 'all'.
preprocess: bool, preprocess data or not.
n_jobs: int, number of jobs.

Value

LazyBoostingClassifier object

Examples

```

# See https://thierrymoudiki.github.io/blog/2024/10/14/r/genericboosting-r for advanced examples

## Not run:
library(mlsauce)
library(datasets)

data(iris)

X <- as.matrix(iris[, 1:4])
y <- as.integer(iris[, 5]) - 1L

n <- dim(X)[1]
p <- dim(X)[2]

set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index

X_train <- as.matrix(X[train_index, ])
y_train <- as.integer(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- as.integer(y[test_index])

```

```
obj <- LazyBoostingClassifier(verbose=0, ignore_warnings=TRUE,
                             custom_metric=NULL, preprocess=FALSE)

obj$fit(X_train, X_test, y_train, y_test)

## End(Not run)
```

LazyBoostingRegressor *LazyBoostingRegressor*

Description

Lazy Generic Boosting Regressor (AutoML Hold-out set validation)

Usage

```
LazyBoostingRegressor(
  verbose = 0,
  ignore_warnings = TRUE,
  custom_metric = NULL,
  predictions = FALSE,
  sort_by = "RMSE",
  random_state = 42L,
  estimators = "all",
  preprocess = FALSE,
  n_jobs = NULL,
  venv_path = "./venv",
  ...
)
```

Arguments

verbose: int, progress bar (yes = 1) or not (no = 0) (currently).

ignore_warnings: bool, ignore warnings.

custom_metric: function, custom metric.

predictions: bool, return predictions.

sort_by: str, sort by metric.

random_state: int, random state.

estimators: str, estimators to use. List of names for custom, or just 'all'.

preprocess: bool, preprocess data or not.

n_jobs: int, number of jobs.

Value

LazyBoostingRegressor object

Examples

```
# See https://thierrymoudiki.github.io/blog/2024/10/14/r/genericboosting-r for advanced examples
## Not run:
library(mlsauce)
library(datasets)

data(mtcars)

X <- as.matrix(mtcars[, -1])
y <- as.integer(mtcars[, 1])

n <- dim(X)[1]
p <- dim(X)[2]

set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index

X_train <- as.matrix(X[train_index, ])
y_train <- as.integer(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- as.integer(y[test_index])

obj <- LazyBoostingRegressor(verbose=0, ignore_warnings=TRUE,
                             custom_metric=NULL, preprocess=FALSE)

obj$fit(X_train, X_test, y_train, y_test)

## End(Not run)
```

LSBoostClassifier

LSBoost classifier

Description

LSBoost classifier

Usage

```
LSBoostClassifier(
  n_estimators = 100L,
  learning_rate = 0.1,
  n_hidden_features = 5L,
  reg_lambda = 0.1,
  row_sample = 1,
  col_sample = 1,
  dropout = 0,
  tolerance = 1e-04,
  direct_link = 1L,
```

```

    verbose = 1L,
    seed = 123L,
    solver = c("ridge", "lasso"),
    activation = "relu",
    n_clusters = 0L,
    clustering_method = "kmeans",
    cluster_scaling = "standard",
    degree = 1L,
    weights_distr = "uniform",
    venv_path = "./venv",
    ...
)

```

Arguments

n_estimators: int, number of boosting iterations.
learning_rate: float, controls the learning speed at training time.
n_hidden_features:
 int
 number of nodes in successive hidden layers.
reg_lambda: float, L2 regularization parameter for successive errors in the optimizer (at training time).
row_sample: float, percentage of rows chosen from the training set.
col_sample: float, percentage of columns chosen from the training set.
dropout: float, percentage of nodes dropped from the training set.
tolerance: float, controls early stopping in gradient descent (at training time).
direct_link: bool, indicates whether the original features are included (True) in model's fitting or not (False).
verbose: int, progress bar (yes = 1) or not (no = 0) (currently).
seed: int, reproducibility seed for nodes_sim=='uniform', clustering and dropout.
solver: str, type of 'weak' learner; currently in ('ridge', 'lasso')
activation: str, activation function: currently 'relu', 'relu6', 'sigmoid', 'tanh'
n_clusters: int, number of clusters for clustering.
clustering_method:
 str, clustering method: currently 'kmeans', 'gmm' (Gaussian Mixture Model)
cluster_scaling:
 str, scaling method for clustering: currently 'standard', 'minmax', 'robust'
degree: int, degree of polynomial interactions features.
weights_distr: str, distribution of weights for the hidden layer currently 'uniform', 'gaussian'

Value

An object of class LSBoostClassifier

Examples

```

library(datasets)

X <- as.matrix(iris[, 1:4])
y <- as.integer(iris[, 5]) - 1L

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(X[train_index, ])
y_train <- floor(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- floor(y[test_index])

## Not run:
obj <- mlsauce::LSBoostClassifier()

print(obj$get_params())

obj$fit(X_train, y_train)

print(obj$score(X_test, y_test))

## End(Not run)

```

LSBoostRegressor

LSBoost Regressor

Description

LSBoost Regressor

Usage

```

LSBoostRegressor(
  n_estimators = 100L,
  learning_rate = 0.1,
  n_hidden_features = 5L,
  reg_lambda = 0.1,
  row_sample = 1,
  col_sample = 1,
  dropout = 0,
  tolerance = 1e-04,
  direct_link = 1L,
  verbose = 1L,
  seed = 123L,

```

```

    solver = c("ridge", "lasso"),
    activation = "relu",
    n_clusters = 0L,
    clustering_method = "kmeans",
    cluster_scaling = "standard",
    degree = 1L,
    weights_distr = "uniform",
    venv_path = "./venv",
    ...
)

```

Arguments

n_estimators: int, number of boosting iterations.
learning_rate: float, controls the learning speed at training time.
n_hidden_features:
 int
 number of nodes in successive hidden layers.
reg_lambda: float, L2 regularization parameter for successive errors in the optimizer (at training time).
row_sample: float, percentage of rows chosen from the training set.
col_sample: float, percentage of columns chosen from the training set.
dropout: float, percentage of nodes dropped from the training set.
tolerance: float, controls early stopping in gradient descent (at training time).
direct_link: bool, indicates whether the original features are included (True) in model's fitting or not (False).
verbose: int, progress bar (yes = 1) or not (no = 0) (currently).
seed: int, reproducibility seed for nodes_sim=='uniform', clustering and dropout.
solver: str, type of 'weak' learner; currently in ('ridge', 'lasso')
activation: str, activation function: currently 'relu', 'relu6', 'sigmoid', 'tanh'
n_clusters: int, number of clusters for clustering.
clustering_method:
 str, clustering method: currently 'kmeans', 'gmm' (Gaussian Mixture Model)
cluster_scaling:
 str, scaling method for clustering: currently 'standard', 'minmax', 'robust'
degree: int, degree of polynomial interactions features.
weights_distr: str, distribution of weights for the hidden layer currently 'uniform', 'gaussian'

Value

An object of class LSBoostRegressor

Examples

```
## Not run:
library(datasets)

X <- as.matrix(datasets::mtcars[, -1])
y <- as.integer(datasets::mtcars[, 1])

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(X[train_index, ])
y_train <- as.double(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- as.double(y[test_index])

obj <- mlsauce::LSBoostRegressor()

print(obj$get_params())

obj$fit(X_train, y_train)

print(obj$score(X_test, y_test))

## End(Not run)
```

RidgeRegressor

Ridge regressor

Description

Ridge regressor

Usage

```
RidgeRegressor(reg_lambda = 0.1, venv_path = "./venv", ...)
```

Arguments

reg_lambda L2 regularization parameter

Value

An object of class Ridge

Examples

```
## Not run:
library(datasets)

X <- as.matrix(datasets::mtcars[, -1])
y <- as.integer(datasets::mtcars[, 1])

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(X[train_index, ])
y_train <- as.double(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- as.double(y[test_index])

obj <- mlsauce::RidgeRegressor()

print(obj$get_params())

obj$fit(X_train, y_train)

print(obj$score(X_test, y_test))

## End(Not run)
```

RVFLJackknifePlus

RVFL Jackknife+ Regressor

Description

Random Vector Functional Link (RVFL) regressor with closed-form Jackknife+ prediction intervals.

Usage

```
RVFLJackknifePlus(
  n_hidden = 200L,
  lambda = 1,
  activation = c("tanh", "relu", "sigmoid"),
  random_state = 0L,
  symmetric = FALSE,
  venv_path = "./venv",
  ...
)
```

Arguments

<code>n_hidden</code>	int. Number of random hidden features. Set to 0 for ridge regression.
<code>lambda</code>	numeric. Ridge regularization parameter.
<code>activation</code>	character. Activation function for the random hidden layer. One of "tanh", "relu", or "sigmoid".
<code>random_state</code>	integer. Random seed used to generate the hidden layer.
<code>symmetric</code>	logical. If TRUE, symmetric Jackknife+ intervals are returned. Otherwise, asymmetric Jackknife+ intervals are used.
<code>venv_path</code>	character. Path to the Python virtual environment.

Value

An object of class `RVFLJackknifePlus`.

Examples

```
## Not run:
library(datasets)

X <- as.matrix(mtcars[, -1])
y <- mtcars[, 1]

n <- nrow(X)
set.seed(123)
train_index <- sample(seq_len(n), floor(0.8 * n))

X_train <- X[train_index, ]
y_train <- y[train_index]

X_test <- X[-train_index, ]

obj <- mlsauce::RVFLJackknifePlus()

print(obj$get_params())

obj$fit(X_train, y_train)

pred <- obj$predict(X_test)

pred_pi <- obj$predict(
  X_test,
  alpha = 0.05,
  return_pi = TRUE
)

print(pred)
print(pred_pi$lower)
print(pred_pi$upper)

## End(Not run)
```

StumpClassifier	<i>Stump classifier</i>
-----------------	-------------------------

Description

Stump classifier

Usage

```
StumpClassifier(bins = "auto", venv_path = "./venv", ...)
```

Arguments

bins: int, number of histogram bins.

Value

An object of class StumpClassifier

Examples

```
## Not run:
library(datasets)

X <- as.matrix(iris[, 1:4])
y <- as.integer(iris[, 5]) - 1L

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(iris[train_index, 1:4])
y_train <- as.integer(iris[train_index, 5]) - 1L
X_test <- as.matrix(iris[test_index, 1:4])
y_test <- as.integer(iris[test_index, 5]) - 1L

obj <- mlsauce::StumpClassifier()

print(obj$get_params())

obj$fit(X_train, y_train)

print(obj$score(X_test, y_test))

## End(Not run)
```

Index

AdaOpt, [2](#)

GenericBoostingClassifier, [4](#)

GenericBoostingRegressor, [6](#)

LassoRegressor, [8](#)

LazyBoostingClassifier, [9](#)

LazyBoostingRegressor, [11](#)

LSBoostClassifier, [12](#)

LSBoostRegressor, [14](#)

RidgeRegressor, [16](#)

RVFLJackknifePlus, [17](#)

StumpClassifier, [19](#)