Package: nnetsauce (via r-universe)

October 6, 2026

Title Randomized and Quasi-Randomized networks for Statistical/Machine Learning

Version 0.50.1

Date 2026-02-04

Author T. Moudiki

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

Description Randomized and Quasi-Randomized networks for Statistical/Machine Learning

Imports reticulate

Depends R (>= 3.3.3), forecast, memoise

License BSD_3_clause Clear + file LICENSE

Suggests roxygen2, reticulate

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Config/pak/sysreqs libpng-dev python3

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

Date/Publication 2026-04-24 17:39:48 UTC

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

RemoteRef HEAD

RemoteSha 698dae1cb0ebc4bb991fc642820758b6cf7d833f

Contents

AdaBoostClassifier 2
BaseRegressor 3
BayesianRVFL2Regressor 5
BayesianRVFLRegressor 6
CustomClassifier 7
CustomRegressor 8

DeepClassifier 9
DeepMTS 10
DeepRegressor 11
GLMClassifier 12
GLMRegressor 13
LazyClassifier 14
LazyDeepClassifier 15
LazyDeepMTS 17
LazyDeepRegressor 18
LazyMTS 19
LazyRegressor 21
MTS 22
MultitaskClassifier 23
plot.MTS 24
QuantileRegressor 25
RandomBagClassifier 26
RandomBagRegressor 27
Ridge2Classifier 28
Ridge2MultitaskClassifier 29
Ridge2Regressor 30
to_forecast 31

Index **33**

`AdaBoostClassifier` *Adaboost classifier with quasi-randomized hidden layer*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
AdaBoostClassifier(
  obj,
  n_estimators = 10L,
  learning_rate = 0.1,
  n_hidden_features = 1L,
  reg_lambda = 0,
  reg_alpha = 0.5,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = FALSE,
  n_clusters = 2L,
  cluster_encode = TRUE,
```

```
  type_clust = "kmeans",
  col_sample = 1,
  row_sample = 1,
  seed = 123L,
  verbose = 1,
  method = "SAMME",
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

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(213)
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

# ValueError: Sample weights must be 1D array or scalar
# sklearn <- nnetsauce::get_sklearn()
# obj <- sklearn$tree$DecisionTreeClassifier()
# obj2 <- AdaBoostClassifier(obj)
# obj2$fit(X_train, y_train)
# print(obj2$score(X_test, y_test))
# print(obj2$predict_proba(X_test))

## End(Not run)
```

`BaseRegressor` *Linear regressor with a quasi-randomized layer*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
BaseRegressor(
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  col_sample = 1,
  row_sample = 1,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

```
## Not run:
set.seed(123)
n <- 50 ; p <- 3
X <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)

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

set.seed(213)
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 <- y[train_index]
X_test <- as.matrix(X[test_index, ])
y_test <- y[test_index]

obj <- BaseRegressor(n_hidden_features=10L, dropout=0.9)
print(obj$fit(X_train, y_train))
print(obj$score(X_test, y_test))

## End(Not run)
```

BayesianRVFL2Regressor *Bayesian Random Vector Functional link network with 2 shrinkage parameters*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/ #' @return

Usage

```
BayesianRVFL2Regressor(
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  s1 = 0.1,
  s2 = 0.1,
  sigma = 0.05,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

```
## Not run:
set.seed(123)
n <- 50 ; p <- 3
X <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]
```

```
obj <- BayesianRVFL2Regressor(n_hidden_features = 5L, s1=0.01)
print(obj$fit(X_train, y_train))
print(obj$score(X_test, y_test))

## End(Not run)
```

BayesianRVFLRegressor *Bayesian Random Vector Functional link network with 1 shrinkage parameter*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
BayesianRVFLRegressor(
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  s = 0.1,
  sigma = 0.05,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

```
## Not run:
set.seed(123)
n <- 50 ; p <- 3
X <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
```

```
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- BayesianRVFLRegressor(n_hidden_features = 5L)
print(obj$fit(X_train, y_train))
print(obj$score(X_test, y_test))

## End(Not run)
```

CustomClassifier *Custom classifier with quasi-randomized layer*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
CustomClassifier(
  obj,
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  col_sample = 1,
  row_sample = 1,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

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

set.seed(123)
X <- as.matrix(iris[, 1:4])
y <- as.integer(iris$Species) - 1L

(index_train <- base::sample.int(n = nrow(X),
```

```
                                size = floor(0.8*nrow(X)),
                                replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

sklearn <- nnetsauce::get_sklearn()
obj <- sklearn$tree$DecisionTreeClassifier()
obj2 <- CustomClassifier(obj)
obj2$fit(X_train, y_train)
print(obj2$score(X_test, y_test))

## End(Not run)
```

`CustomRegressor` *Custom regressor with quasi-randomized layer*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
CustomRegressor(
  obj,
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  col_sample = 1,
  row_sample = 1,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

```
## Not run:
set.seed(123)
```

```
n <- 50 ; p <- 3
X <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

sklearn <- nnetsauce::get_sklearn()
obj <- sklearn$linear_model$ElasticNet()
obj2 <- CustomRegressor(obj)
obj2$fit(X_train, y_train)
print(obj2$score(X_test, y_test))

## End(Not run)
```

`DeepClassifier` *Deep classification models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

```
DeepClassifier(obj, n_layers = 3L, venv_path = "./venv", ...)
```

Arguments

`obj`	a model object
`n_layers`	number of hidden layers
`...`	additional parameters to be passed to `nnetsauce::CustomClassifier`

Examples

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

set.seed(123)
X <- as.matrix(iris[, 1:4])
y <- as.integer(iris$Species) - 1L

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
```

```
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

sklearn <- nnetsauce::get_sklearn()
obj2 <- sklearn$linear_model$LogisticRegressionCV()

obj <- DeepClassifier(obj2, n_layers = 3L)
res <- obj$fit(X_train, y_train)

# accuracy
print(mean(obj$predict(X_test)==y_test))

## End(Not run)
```

DeepMTS *Deep MTS models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

```
DeepMTS(
  obj,
  n_layers = 3L,
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  lags = 1L,
  replications = NULL,
  kernel = NULL,
  agg = "mean",
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  verbose = 0,
  venv_path = "./venv",
  ...
)
```

Arguments

obj	a model object
n_layers	number of hidden layers
...	additional parameters to be passed to nnetsauce::CustomRegressor

Examples

```
## Not run:
set.seed(123)
X <- matrix(rnorm(300), 100, 3)

sklearn <- nnetsauce::get_sklearn()
obj <- sklearn$linear_model$ElasticNet()
obj2 <- DeepMTS(obj)

obj2$fit(X)
obj2$predict()

## End(Not run)
```

DeepRegressor *Deep regression models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

```
DeepRegressor(obj, n_layers = 3L, venv_path = "./venv", ...)
```

Arguments

obj	a model object
n_layers	number of hidden layers
...	additional parameters to be passed to nnetsauce::CustomRegressor

Examples

```
## Not run:
X <- MASS::Boston[,-14] # dataset has an ethical problem
y <- MASS::Boston$medv

set.seed(13)
(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
```

```
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

sklearn <- nnetsauce::get_sklearn()
obj2 <- sklearn$linear_model$ElasticNet()

obj <- DeepRegressor(obj2, n_layers = 3L, n_clusters=2L)
res <- obj$fit(X_train, y_train)
print(obj$predict(X_test))

## End(Not run)
```

GLMClassifier *Generalized nonlinear models for Classification*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
GLMClassifier(
  n_hidden_features = 5L,
  lambda1 = 0.01,
  alpha1 = 0.5,
  lambda2 = 0.01,
  alpha2 = 0.5,
  family = "expit",
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  type_scaling = c("std", "std", "std"),
  optimizer = ns$Optimizer(),
  seed = 123L,
  venv_path = "./venv",
  ...
)
```

Examples

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

set.seed(123)
X <- as.matrix(iris[, 1:4])
y <- as.integer(iris$Species) - 1L

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- GLMClassifier()
obj$fit(X_train, y_train)
print(obj$score(X_test, y_test))

## End(Not run)
```

GLMRegressor *Generalized nonlinear models for continuous output (regression)*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
GLMRegressor(
  n_hidden_features = 5L,
  lambda1 = 0.01,
  alpha1 = 0.5,
  lambda2 = 0.01,
  alpha2 = 0.5,
  family = "gaussian",
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  type_scaling = c("std", "std", "std"),
```

```
  optimizer = ns$Optimizer(),
  seed = 123L,
  venv_path = "./venv",
  ...
)
```

Examples

```
## Not run:
set.seed(123)
n <- 50 ; p <- 3
X <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- GLMRegressor()
obj$fit(X_train, y_train)
print(obj$score(X_test, y_test))

## End(Not run)
```

LazyClassifier *Automated Machine Learning for classification models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

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

Arguments

verbose	monitor progress (0, default, is false and 1 is true)
ignore_warnings	print trace when model fitting failed
custom_metric	defining a custom metric (default is NULL)
predictions	obtain predictions (default is FALSE)
random_state	reproducibility seed
estimators	specify classifiers to be adjusted (default is 'all')
preprocess	preprocessing input covariates (default is FALSE FALSE)
...	additional parameters to be passed to nnetsauce::CustomClassifier

Value

a list that you can $fit

Examples

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

set.seed(123)
X <- as.matrix(iris[, 1:4])
y <- as.integer(iris$Species) - 1L

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- LazyClassifier()
res <- obj$fit(X_train, X_test, y_train, y_test)
print(res[[1]])

## End(Not run)
```

LazyDeepClassifier *Automated Machine Learning for deep classification models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

```
LazyDeepClassifier(
  verbose = 0,
  ignore_warnings = TRUE,
  custom_metric = NULL,
  predictions = FALSE,
  random_state = 42L,
  estimators = "all",
  preprocess = FALSE,
  n_layers = 3L,
  venv_path = "./venv",
  ...
)
```

Arguments

verbose	monitor progress (0, default, is false and 1 is true)
ignore_warnings	print trace when model fitting failed
custom_metric	defining a custom metric (default is NULL)
predictions	obtain predictions (default is FALSE)
random_state	reproducibility seed
estimators	specify classifiers to be adjusted (default is 'all')
preprocess	preprocessing input covariates (default is FALSE FALSE)
n_layers	number of layers for the deep model
...	additional parameters to be passed to nnetsauce::CustomClassifier

Value

a list that you can $fit

Examples

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

set.seed(123)
X <- as.matrix(iris[, 1:4])
y <- as.integer(iris$Species) - 1L

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]
```

```
obj <- LazyDeepClassifier()
res <- obj$fit(X_train, X_test, y_train, y_test)
print(res[[1]])

## End(Not run)
```

LazyDeepMTS *Automated Machine Learning for deep time series models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

```
LazyDeepMTS(
  verbose = 0,
  ignore_warnings = TRUE,
  custom_metric = NULL,
  predictions = FALSE,
  random_state = 42L,
  estimators = "all",
  preprocess = FALSE,
  show_progress = TRUE,
  n_layers = 3L,
  venv_path = "./venv",
  ...
)
```

Arguments

verbose	monitor progress (0, default, is false and 1 is true)
ignore_warnings	print trace when model fitting failed
custom_metric	defining a custom metric (default is NULL)
predictions	obtain predictions (default is FALSE)
random_state	reproducibility seed
estimators	specify regressors to be adjusted (default is 'all')
preprocess	preprocessing input covariates (default is FALSE FALSE)
n_layers	number of layers for the deep model
...	additional parameters to be passed to nnetsauce::CustomRegressor

Value

a list that you can $fit

Examples

```
## Not run:
set.seed(123)
X <- matrix(rnorm(300), 100, 3)

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- data.frame(X[index_train, ])
X_test <- data.frame(X[-index_train, ])

obj <- LazyDeepMTS()

res <- obj$fit(X_train, X_test)
print(res[[1]])

## End(Not run)
```

`LazyDeepRegressor` *Automated Machine Learning for deep regression models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

```
LazyDeepRegressor(
  verbose = 0,
  ignore_warnings = TRUE,
  custom_metric = NULL,
  predictions = FALSE,
  random_state = 42L,
  estimators = "all",
  preprocess = FALSE,
  n_layers = 3L,
  venv_path = "./venv",
  ...
)
```

Arguments

`verbose`	monitor progress (`0`, default, is false and `1` is true)
`ignore_warnings`	print trace when model fitting failed
`custom_metric`	defining a custom metric (default is `NULL`)
`predictions`	obtain predictions (default is `FALSE`)

random_state	reproducibility seed
estimators	specify regressors to be adjusted (default is 'all')
preprocess	preprocessing input covariates (default is FALSE FALSE)
n_layers	number of layers for the deep model
...	additional parameters to be passed to nnetsauce::CustomRegressor

Value

a list that you can $fit

Examples

```
## Not run:
X <- MASS::Boston[,-14] # dataset has an ethical problem
y <- MASS::Boston$medv

set.seed(13)
(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- LazyDeepRegressor()
res <- obj$fit(X_train, X_test, y_train, y_test)
print(res[[1]])

## End(Not run)
```

LazyMTS *Automated Machine Learning for time series models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

```
LazyMTS(
  verbose = 0,
  ignore_warnings = TRUE,
  custom_metric = NULL,
  predictions = FALSE,
  random_state = 42L,
  estimators = "all",
```

```
  preprocess = FALSE,
  show_progress = TRUE,
  venv_path = "./venv",
  ...
)
```

Arguments

verbose	monitor progress (0, default, is false and 1 is true)
ignore_warnings	print trace when model fitting failed
custom_metric	defining a custom metric (default is NULL)
predictions	obtain predictions (default is FALSE)
random_state	reproducibility seed
estimators	specify regressors to be adjusted (default is 'all')
preprocess	preprocessing input covariates (default is FALSE FALSE)
...	additional parameters to be passed to nnetsauce::CustomRegressor

Value

a list that you can $fit

Examples

```
## Not run:
set.seed(123)
X <- matrix(rnorm(300), 100, 3)

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- data.frame(X[index_train, ])
X_test <- data.frame(X[-index_train, ])

obj <- LazyMTS()

res <- obj$fit(X_train, X_test)
print(res[[1]])

## End(Not run)
```

LazyRegressor *Automated Machine Learning for regression models*

Description

See also https://techtonique.github.io/nnetsauce/

Usage

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

Arguments

verbose	monitor progress (0, default, is false and 1 is true)
ignore_warnings	print trace when model fitting failed
custom_metric	defining a custom metric (default is NULL)
predictions	obtain predictions (default is FALSE)
random_state	reproducibility seed
estimators	specify regressors to be adjusted (default is 'all')
preprocess	preprocessing input covariates (default is FALSE FALSE)
...	additional parameters to be passed to nnetsauce::CustomRegressor

Value

a list that you can $fit

Examples

```
## Not run:
X <- MASS::Boston[,-14] # dataset has an ethical problem
y <- MASS::Boston$medv

set.seed(13)
(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
```

```
                                  replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- LazyRegressor()
res <- obj$fit(X_train, X_test, y_train, y_test)
print(res[[1]])

## End(Not run)
```

MTS *Multivariate Time Series*

Description

Parameters description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
MTS(
  obj,
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  lags = 1L,
  replications = NULL,
  kernel = NULL,
  agg = "mean",
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  verbose = 0,
  venv_path = "./venv",
  ...
)
```

Examples

```
## Not run:
# Example 1 -----

set.seed(123)
X <- matrix(rnorm(300), 100, 3)

obj <- sklearn$linear_model$ElasticNet()
obj2 <- MTS(obj)

obj2$fit(X)
obj2$predict()


# Example 2 -----

set.seed(123)
X <- matrix(rnorm(300), 100, 3)

obj <- sklearn$linear_model$BayesianRidge()
obj2 <- MTS(obj)

obj2$fit(X)
obj2$predict(return_std = TRUE)

## End(Not run)
```

`MultitaskClassifier` *Multitask Classification model based on regression models, with shared covariates*

Description

Parameters description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
MultitaskClassifier(
  obj,
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
```

```
  col_sample = 1,
  row_sample = 1,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

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

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

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- sklearn$linear_model$LinearRegression()
obj2 <- MultitaskClassifier(obj)
obj2$fit(X_train, y_train)
print(obj2$score(X_test, y_test))
print(obj2$predict_proba(X_test))

## End(Not run)
```

`plot.MTS` *Plot multivariate time series forecast or residuals*

Description

Plot multivariate time series forecast or residuals

Usage

```
## S3 method for class 'MTS'
plot(x, selected_series, level = 95, ...)
```

Arguments

`x`	result from `basicf`, `ridge2f` or `varf` (multivariate time series forecast)

selected_series	name of the time series selected for plotting
level	confidence levels for prediction intervals
...	additional parameters to be passed to `plot` or `matplot`
type	"pi": basic prediction intervals; "dist": a distribution of predictions; "sims": the simulations

---

`QuantileRegressor` *Model-agnostic quantile regressor*

Description

Model-agnostic quantile regressor

Usage

```
QuantileRegressor(
  obj,
  level = 95,
  scoring = "predictions",
  venv_path = "./venv",
  ...
)
```

Arguments

obj	A model object
level	The quantile level
scoring	The scoring function ("predictions", "residuals", "conformal", "studentized", "conformal-studentized")
venv_path	The path to the virtual environment
...	Additional arguments

Examples

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

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

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

set.seed(213)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
```

```
test_index <- -train_index

X_train <- as.matrix(mtcars[train_index, -1])
y_train <- mtcars[train_index, 1]
X_test <- as.matrix(mtcars[test_index, -1])
y_test <- mtcars[test_index, 1]

sklearn <- nnetsauce::get_sklearn()
obj <- sklearn$ensemble$RandomForestRegressor(random_state=42L)
obj2 <- QuantileRegressor(obj)
obj2$fit(X_train, y_train)
print(obj2$score(X_test, y_test))

## End(Not run)
```

RandomBagClassifier *Bootstrap aggregating with quasi-randomized layer (classification)*

Description

Parameters description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
RandomBagClassifier(
  obj,
  n_estimators = 50L,
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = FALSE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  col_sample = 1,
  row_sample = 1,
  n_jobs = NULL,
  seed = 123L,
  verbose = 1L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

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

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

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

sklearn <- nnetsauce::get_sklearn()
obj <- sklearn$tree$DecisionTreeClassifier()
obj2 <- RandomBagClassifier(obj, n_estimators=50L,
n_hidden_features=5L)
obj2$fit(X_train, y_train)
print(obj2$score(X_test, y_test))
print(obj2$predict_proba(X_test))

## End(Not run)
```

RandomBagRegressor *Bootstrap aggregating with quasi-randomized layer (regression)*

Description

Parameters description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
RandomBagRegressor(
  obj,
  n_estimators = 10L,
  n_hidden_features = 1L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = FALSE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  col_sample = 1,
```

```
  row_sample = 1,
  n_jobs = NULL,
  seed = 123L,
  verbose = 1L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

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

n <- 20 ; p <- 5
X <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)

sklearn <- nnetsauce::get_sklearn()
obj <- sklearn$tree$DecisionTreeRegressor()
obj2 <- RandomBagRegressor(obj)
obj2$fit(X[1:12,], y[1:12])
print(obj2$score(X[13:20, ], y[13:20]))

## End(Not run)
```

`Ridge2Classifier` *Multinomial logit, quasi-randomized classification model with 2 shrinkage parameters*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
Ridge2Classifier(
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  direct_link = TRUE,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  lambda1 = 0.1,
```

```
  lambda2 = 0.1,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

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

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

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- Ridge2Classifier()
obj$fit(X_train, y_train)
print(obj$score(X_test, y_test))
print(obj$predict_proba(X_train))

## End(Not run)
```

`Ridge2MultitaskClassifier` *Multitask quasi-randomized classification model with 2 shrinkage parameters*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
Ridge2MultitaskClassifier(
  n_hidden_features = 5L,
  activation_name = "relu",
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
```

```
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  lambda1 = 0.1,
  lambda2 = 0.1,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

```
## Not run:
# Example 1 -----

library(datasets)

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

(index_train <- base::sample.int(n = nrow(X),
                                 size = floor(0.8*nrow(X)),
                                 replace = FALSE))
X_train <- X[index_train, ]
y_train <- y[index_train]
X_test <- X[-index_train, ]
y_test <- y[-index_train]

obj <- Ridge2MultitaskClassifier()
obj$fit(X_train, y_train)
print(obj$score(X_test, y_test))
print(obj$predict_proba(X_train))

## End(Not run)
```

`Ridge2Regressor` *Quasi-randomized regression model with 2 shrinkage parameters*

Description

Parameters' description can be found at https://techtonique.github.io/nnetsauce/

Usage

```
Ridge2Regressor(
  n_hidden_features = 5L,
  activation_name = "relu",
```

```
  a = 0.01,
  nodes_sim = "sobol",
  bias = TRUE,
  dropout = 0,
  n_clusters = 2L,
  cluster_encode = TRUE,
  type_clust = "kmeans",
  lambda1 = 0.1,
  lambda2 = 0.1,
  seed = 123L,
  backend = c("cpu", "gpu", "tpu"),
  venv_path = "./venv",
  ...
)
```

Examples

```
## Not run:
set.seed(123)
n <- 50 ; p <- 3
X <- matrix(rnorm(n * p), n, p) # no intercept!
y <- rnorm(n)

obj <- nnetsauce::Ridge2Regressor(n_hidden_features = 5L)
print(obj$fit(X, y))
print(obj$score(X, y))

## End(Not run)
```

`to_forecast` *Transform list to forecast or mforecast object*

Description

Transform list to forecast or mforecast object

Usage

```
to_forecast(out, start_input, frequency_input)
```

Arguments

`out`	a list, an object constructed by `MTS` or `DeepMTS`
`start_input`	start of the input time series (as in `stats::start`)
`frequency_input`	frequency of the input time series (as in `stats::frequency`)

Value

an object of class `forecast` or `mforecast`

Index

AdaBoostClassifier, 2

BaseRegressor, 3
BayesianRVFL2Regressor, 5
BayesianRVFLRegressor, 6

CustomClassifier, 7
CustomRegressor, 8

DeepClassifier, 9
DeepMTS, 10
DeepRegressor, 11

GLMClassifier, 12
GLMRegressor, 13

LazyClassifier, 14
LazyDeepClassifier, 15
LazyDeepMTS, 17
LazyDeepRegressor, 18
LazyMTS, 19
LazyRegressor, 21

MTS, 22
MultitaskClassifier, 23

nnetsauce::CustomClassifier, *9*, *15, 16*
nnetsauce::CustomRegressor, *11*, *17*, *19–21*

plot.MTS, 24

QuantileRegressor, 25

RandomBagClassifier, 26
RandomBagRegressor, 27
Ridge2Classifier, 28
Ridge2MultitaskClassifier, 29
Ridge2Regressor, 30

to_forecast, 31