Package ‘cuml4r’

July 26, 2021

Type Package

Title R Interface for the RAPIDS cuML Suite of Libraries

Version 0.1.0

Maintainer Yitao Li <yitao@rstudio.com>

Description The purpose of 'cuml4r' is to provide a simple and intuitive R
interface for cuML (<https://github.com/rapidsai/cuml>).
CuML is a suite of GPU-accelerated machine learning libraries powered by
CUDA (<https://en.wikipedia.org/wiki/CUDA>).

License Apache License (>= 2.0)

Imports magrittr, Rcpp (>= 1.0.6), rlang (>= 0.1.4), zeallot (>=
0.1.0)

Suggests MASS, purrr

LinkingTo Rcpp

OS_type unix

SystemRequirements RAPIDS cuML (see https://rapids.ai/start.html)

RoxygenNote 7.1.1

NeedsCompilation yes

Author Yitao Li [aut, cre] (<https://orcid.org/0000-0002-1261-905X>)

Repository CRAN

Date/Publication 2021-07-26 06:40:02 UTC

R topics documented:

cuml4r 2
cuml_dbscan 2
cuml_kmeans 3
cuml_rand_forest 4
cuml_svm 6

Index **9**

`cuml4r` *cuml4r*

Description

The purpose of 'cuml4r' is to provide a simple and intuitive R interface for 'cuML' (<https://github.com/rapidsai/cuml>). 'cuML' is a suite of GPU- accelerated machine learning libraries powered by 'CUDA' (<https://en.wikipedia.org/wiki/CUDA

Author(s)

Yitao Li <yitao@rstudio.com>

`cuml_dbscan` *Run the DBSCAN clustering algorithm.*

Description

Run the DBSCAN (Density-based spatial clustering of applications with noise) clustering algorithm.

Usage

```
cuml_dbscan(x, min_pts, eps)
```

Arguments

`x`	The input matrix or dataframe. Each data point should be a row and should consist of numeric values only.
`min_pts, eps`	A point 'p' is a core point if at least 'min_pts' are within distance 'eps' from it.

Value

A list containing the cluster assignments of all data points. A data point not belonging to any cluster (i.e., "noise") will have NA its cluster assignment.

Examples

```
library(cuml4r)
library(magrittr)

gen_pts <- function() {
  centroids <- list(c(1000, 1000), c(-1000, -1000), c(-1000, 1000))

  pts <- centroids %>%
    purrr::map(
      ~ MASS::mvrnorm(10, mu = .x, Sigma = matrix(c(1, 0, 0, 1), nrow = 2))
    )

  rlang::exec(rbind, !!!pts)
}

m <- gen_pts()
clusters <- cuml_dbscan(m, min_pts = 5, eps = 3)

print(clusters)
```

`cuml_kmeans` *Run the K means clustering algorithm.*

Description

Run the K means clustering algorithm.

Usage

```
cuml_kmeans(x, k, max_iters = 300)
```

Arguments

`x`	The input matrix or dataframe. Each data point should be a row and should consist of numeric values only.
`k`	The number of clusters.
`max_iters`	Maximum number of iterations (default: 300).

Value

A list containing the cluster assignments and the centroid of each cluster. Each centroid will be a column within the ‘centroids‘ matrix.

Examples

```
library(cuml4r)

kclust <- cuml_kmeans(
  iris[, which(names(iris) != "Species")],
  k = 3,
  max_iters = 100
)

print(kclust)
```

`cuml_rand_forest` *Train a random forest model.*

Description

Train a random forest model for classification or regression tasks.

Usage

```
cuml_rand_forest(
  x,
  y = NULL,
  formula = NULL,
  mode = c("classification", "regression"),
  mtry = NULL,
  trees = NULL,
  min_n = NULL,
  bootstrap = TRUE,
  max_depth = 16,
  max_leaves = -1,
  max_predictors_per_note_split = NULL,
  n_bins = 128,
  min_samples_leaf = 1,
  split_criterion = NULL,
  min_impurity_decrease = 0,
  max_batch_size = 128,
  n_streams = 8,
  cuml_log_level = c("off", "critical", "error", "warn", "info", "debug", "trace")
)
```

Arguments

`x`	The input matrix or dataframe. Each data point should be a row and should consist of numeric values only.
`y`	A numeric vector of desired responses.
`formula`	If 'x' is a dataframe, then a R formula syntax of the form '<response col> ~ .' or '<response col> ~ <predictor 1> + <predictor 2> + ...' may be used to specify the response column and the predictor column(s).
`mode`	Type of task to perform. Should be either "classification" or "regression".
`mtry`	The number of predictors that will be randomly sampled at each split when creating the tree models. Default: the square root of the total number of predictors.
`trees`	An integer for the number of trees contained in the ensemble. Default: 100.
`min_n`	An integer for the minimum number of data points in a node that are required for the node to be split further. Default: 2.

`bootstrap`	Whether to perform bootstrap. If TRUE, each tree in the forest is built on a bootstrapped sample with replacement. If FALSE, the whole dataset is used to build each tree.
`max_depth`	Maximum tree depth. Default: 16.
`max_leaves`	Maximum leaf nodes per tree. Soft constraint. Default: -1 (unlimited).
`max_predictors_per_note_split`	Number of predictor to consider per node split. Default: square root of the total number predictors.
`n_bins`	Number of bins used by the split algorithm. Default: 128.
`min_samples_leaf`	The minimum number of data points in each leaf node. Default: 1.
`split_criterion`	The criterion used to split nodes, can be "gini" or "entropy" for classifications, and "mse" or "mae" for regressions. Default: "gini" for classification; "mse" for regression.
`min_impurity_decrease`	Minimum decrease in impurity requried for node to be spilt. Default: 0.
`max_batch_size`	Maximum number of nodes that can be processed in a given batch. Default: 128.
`n_streams`	Number of CUDA streams to use for building trees. Default: 8.
`cuml_log_level`	Log level within cuML library functions. Must be one of "off", "critical", "error", "warn", "info", "debug", "trace". Default: off.

Value

A random forest classifier / regressor object that can be used with the 'predict' S3 generic to make predictions on new data points.

Examples

```
library(cuml4r)

# Classification

model <- cuml_rand_forest(
  iris,
  formula = Species ~ .,
  mode = "classification",
  trees = 100
)

predictions <- predict(model, iris)

print(predictions)

cat(
  "Number of correct predictions: ",
  sum(predictions == iris[, "Species"]),
  "\n"
)

# Regression

model <- cuml_rand_forest(
  iris,
  formula = Species ~ .,
  mode = "regression",
  trees = 100
)

predictions <- predict(model, iris)

print(predictions)
print(round(predictions))

cat(
  "Number of correct predictions: ",
  sum(as.integer(round(predictions)) == as.integer(iris[, "Species"])),
  "\n"
)
```

---

cuml_svm *Train a SVM model.*

Description

Train a Support Vector Machine model for classification or regression tasks.

Usage

```
cuml_svm(
  x,
  y = NULL,
  formula = NULL,
  mode = c("classification", "regression"),
  cost = 1,
  kernel = c("rbf", "tanh", "polynomial", "linear"),
  gamma = 1/ncol(x),
  coef0 = 0,
  degree = 3L,
  tol = 0.001,
  max_iter = 100L * nrow(x),
  nochange_steps = 1000L,
  cache_size = 1024,
  epsilon = 0.1,
  sample_weights = NULL,
 cuml_log_level = c("off", "critical", "error", "warn", "info", "debug", "trace")
)
```

Arguments

`x`	The input matrix or dataframe. Each data point should be a row and should consist of numeric values only.
`y`	A numeric vector of desired responses.
`formula`	If 'x' is a dataframe, then a R formula syntax of the form '<response col> ~ .' or '<response col> ~ <predictor 1> + <predictor 2> + ...' may be used to specify the response column and the predictor column(s).
`mode`	Type of task to perform. Should be either "classification" or "regression".
`cost`	A positive number for the cost of predicting a sample within or on the wrong side of the margin. Default: 1.
`kernel`	Type of the SVM kernel function (must be one of "rbf", "tanh", "polynomial", or "linear"). Default: "rbf".
`gamma`	The gamma coefficient (only relevant to polynomial, RBF, and tanh kernel functions, see explanations below). Default: 1 / (num features). The following kernels are implemented: - RBF K(x_1, x_2) = exp(-gamma |x_1-x_2|^2) - TANH K(x_1, x_2) = tanh(gamma <x_1,x_2> + coef0) - POLYNOMIAL K(x_1, x_2) = (gamma <x_1,x_2> + coef0)^degree - LINEAR K(x_1,x_2) = <x_1,x_2>, where < , > denotes the dot product.
`coef0`	The 0th coefficient (only applicable to polynomial and tanh kernel functions, see explanations below). Default: 0. The following kernels are implemented: - RBF K(x_1, x_2) = exp(-gamma |x_1-x_2|^2) - TANH K(x_1, x_2) = tanh(gamma <x_1,x_2> + coef0) - POLYNOMIAL K(x_1, x_2) = (gamma <x_1,x_2> + coef0)^degree - LINEAR K(x_1,x_2) = <x_1,x_2>, where < , > denotes the dot product.
`degree`	Degree of the polynomial kernel function (note: not applicable to other kernel types, see explanations below). Default: 3. The following kernels are implemented: - RBF K(x_1, x_2) = exp(-gamma |x_1-x_2|^2) - TANH K(x_1, x_2) = tanh(gamma <x_1,x_2> + coef0) - POLYNOMIAL K(x_1, x_2) = (gamma <x_1,x_2> + coef0)^degree - LINEAR K(x_1,x_2) = <x_1,x_2>, where < , > denotes the dot product.
`tol`	Tolerance to stop fitting. Default: 1e-3.
`max_iter`	Maximum number of outer iterations in SmoSolver. Default: 100 * (num samples).
`nochange_steps`	Number of steps with no change w.r.t convergence. Default: 1000.
`cache_size`	Size of kernel cache (MiB) in device memory. Default: 1024.
`epsilon`	Espsilon parameter of the epsilon-SVR model. There is no penalty for points that are predicted within the epsilon-tube around the target values. Please note this parameter is only relevant for regression tasks. Default: 0.1.
`sample_weights`	Optional weight assigned to each input data point.
`cuml_log_level`	Log level within cuML library functions. Must be one of "off", "critical", "error", "warn", "info", "debug", "trace". Default: off.

Value

A Support Vector Machine classifier / regressor object that can be used with the 'predict' S3 generic to make predictions on new data points.

Examples

```
library(cuml4r)

model <- cuml_svm(
  iris[1:100,],
  formula = Species ~ .,
  mode = "classification",
  kernel = "rbf"
)

predictions <- predict(model, iris[1:100,])

cat("Iris species predictions: ", predictions, "\n")

model <- cuml_svm(
  mtcars,
  formula = mpg ~ .,
  mode = "regression",
  kernel = "rbf"
)

predictions <- predict(model, mtcars)

cat("MPG predictions:", predictions, "\n")
```

Index

cuml4r, 2
cuml_dbscan, 2
cuml_kmeans, 3
cuml_rand_forest, 4
cuml_svm, 6