

Package ‘cuda.ml’

August 21, 2026

Type Package

Title R Interface for the RAPIDS cuML Suite of Libraries

Version 0.4.0

Description R interface for RAPIDS cuML (<<https://github.com/NVIDIA/cuml>>),
a suite of GPU-accelerated machine learning libraries powered by CUDA
(<<https://en.wikipedia.org/wiki/CUDA>>).

License MIT + file LICENSE

Copyright file inst/COPYRIGHTS

URL <https://mlverse.github.io/cuda.ml/>,
<https://github.com/mlverse/cuda.ml>

BugReports <https://github.com/mlverse/cuda.ml/issues>

Depends R (>= 4.1)

Imports bundle, digest, ellipsis, filelock, hardhat, jsonlite, Rcpp
(>= 1.0.6), rlang (>= 0.3.0)

Suggests callr, glmnet, knitr, MASS, modeldata, palmerpenguins,
parsnip, purrr, recipes, reticulate, rmarkdown, testthat (>= 3.1.7), workflows, xgboost

VignetteBuilder knitr

Encoding UTF-8

OS_type unix

SystemRequirements Native operations require Linux x86_64 with glibc
2.28 or newer. GPU-backed operations require a supported NVIDIA
GPU and driver 580 or newer.

NeedsCompilation no

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.1.0

Author Yitao Li [aut, cph] (ORCID: <<https://orcid.org/0000-0002-1261-905X>>),
Tomasz Kalinowski [aut, cre, cph],
Daniel Falbel [aut, cph],
RStudio [cph, fnd]

Maintainer Tomasz Kalinowski <tomasz@posit.co>

Repository CRAN

Date/Publication 2026-08-21 05:44:12 UTC

Contents

cuda.ml-package	3
bundle.cuda_ml_model	4
cuda_ml_agglomerative_clustering	5
cuda_ml_backend_info	6
cuda_ml_cache_clean	7
cuda_ml_dbscan	7
cuda_ml_elastic_net	8
cuda_ml_install	11
cuda_ml_kmeans	13
cuda_ml_knn	14
cuda_ml_knn_algo	17
cuda_ml_lasso	18
cuda_ml_linear_reg	21
cuda_ml_logistic_reg	22
cuda_ml_nvforest_export	24
cuda_ml_nvforest_info	26
cuda_ml_nvforest_leaf_ids	27
cuda_ml_nvforest_load_model	27
cuda_ml_nvforest_predict_per_tree	29
cuda_ml_ols	30
cuda_ml_pca	31
cuda_ml_rand_forest	33
cuda_ml_ridge	36
cuda_ml_runtime_audit	38
cuda_ml_serialize	39
cuda_ml_sgd	40
cuda_ml_svm	44
cuda_ml_transform	47
cuda_ml_tsne	48
cuda_ml_tsvd	50
cuda_ml_umap	52
predict.cuda_ml_knn	54
predict.cuda_ml_linear_model	55
predict.cuda_ml_logistic_reg	55
predict.cuda_ml_nvforest	56
predict.cuda_ml_svm	57

Index

58

cuda.ml-package	<i>cuda.ml</i>
-----------------	----------------

Description

This package provides an R interface for the RAPIDS cuML library.

Installation

Install the portable R package from CRAN, then explicitly download its locked native backend and runtime:

```
install.packages("cuda.ml")
cuda.ml::cuda_ml_install()
```

The CRAN package contains no compiled code. Prebuilt backends support Linux x86_64 with glibc 2.28 or newer and are selected for the current R minor version.

Loading **cuda.ml** does not require a GPU, load native code, create a cache, or contact the network. Call `cuda_ml_install()` to install the complete backend used for GPU training and inference. It uses the pinned CUDA 13.2.2 and RAPIDS cuML and nvForest 26.06 runtime and is roughly 1.6 GiB. GPU operations then require a supported NVIDIA GPU and driver 580 or newer.

For CPU-only nvForest inference, call `cuda_ml_install(device = "cpu")` instead. This installs a separate backend that is roughly 1 MiB to download and 3 MiB when installed. It does not install cuML or the complete managed CUDA and RAPIDS runtime and requires neither an NVIDIA GPU nor an NVIDIA driver. It contains no CUDA runtime libraries. Treelite 4.7.0 is linked statically into both backends.

Random forests trained on a GPU by `cuda_ml_rand_forest()` can be persisted with `cuda_ml_serialize()` and restored for CPU inference with `cuda_ml_unserialize(state, device = "cpu")`. Current nvForest model states do not encode their deployment device. Alternatively, `cuda_ml_nvforest_export()` writes a standard Treelite checkpoint and cuda.ml JSON metadata that can be restored with `cuda_ml_nvforest_import()`. Native operations fail with an installation instruction until their corresponding backend is installed. Set `CUDA_ML_CACHE_DIR` to override the default cache and `CUDA_ML_BACKEND_MIRROR` to use an exact backend mirror.

To compile cuda.ml itself on the host without a prebuilt cuda.ml backend, call `cuda_ml_install(source = TRUE)`. The default managed source build downloads the locked CUDA, RAPIDS, CMake, Ninja, and Treelite build inputs. It detects CUDA-visible GPU architectures when available and otherwise uses the package's portable architecture list. Only Linux x86_64 with glibc 2.28 or newer and GNU C++ 14 or newer are required on the host. Use `dependencies = "host"` with explicit `CUDA_HOME`, `CUML_PREFIX`, `CUML_CUDA_ARCHITECTURES`, and `CUDA_ML_CXX` inputs for a fully native, network-free source build.

Author(s)

Yitao Li yitaoli1990@gmail.com

Tomasz Kalinowski tomasz@posit.co

Daniel Falbel daniel@posit.co

See Also

Useful links:

- <https://mlverse.github.io/cuda.ml/>
- <https://github.com/mlverse/cuda.ml>
- Report bugs at <https://github.com/mlverse/cuda.ml/issues>

bundle.cuda_ml_model *Bundle a cuda.ml model*

Description

Converts a model with explicit state into a `bundle::bundle()` object. KNN and TSVD fits do not currently implement explicit state.

Usage

```
## S3 method for class 'cuda_ml_model'
bundle(x, ...)

## S3 method for class 'cuda_ml_nvforest'
bundle(x, device = NULL, ...)
```

Arguments

<code>x</code>	A fitted cuda.ml model.
<code>...</code>	Unused.
<code>device</code>	For an nvForest-backed model, the device on which the bundle will restore. NULL preserves the model's current device. Use "cpu" when bundling a GPU-trained random forest for CPU-only deployment. Other cuda.ml model types do not accept this argument.

Deployment

cuda.ml validates the model state and required backend before loading it. Prepare the backend in the target process with `cuda_ml_install()` for GPU operation or `cuda_ml_install(device = "cpu")` for CPU-only nvForest inference.

Random-forest and nvForest states contain device-neutral Treelite model bytes. They retain prediction precision, class labels, preprocessing, and model semantics, but not the inference device, device identifier, tree layout, chunk size, or memory alignment. Select those settings while restoring; omitting device selects GPU inference.

`bundle::bundle()` stores the same explicit state. For an nvForest-backed model, the bundle also stores its chosen deployment device separately from the device-neutral state. A bundle is not required for deployment; `cuda_ml_serialize()` returns the complete state artifact directly.

See Also

[cuda_ml_serialize](#), [cuda_ml_unserialize](#)

cuda_ml_agglomerative_clustering

Perform single-linkage agglomerative clustering.

Description

Recursively merge the pair of clusters that minimally increases a given linkage distance.

Usage

```
cuda_ml_agglomerative_clustering(
    x,
    n_clusters = 2L,
    metric = c("euclidean", "l1", "l2", "manhattan", "cosine"),
    connectivity = c("pairwise", "knn"),
    n_neighbors = 15L
)
```

Arguments

x	The input matrix or data frame. Each data point should be a row and should consist of numeric values only.
n_clusters	The number of clusters to find. Default: 2L.
metric	Metric used for linkage computation. Must be one of {"euclidean", "l1", "l2", "manhattan", "cosine"}. If connectivity is "knn" then only "euclidean" is accepted. Default: "euclidean".
connectivity	The type of connectivity matrix to compute. Must be one of {"pairwise", "knn"}. Default: "pairwise". 'pairwise' will compute the entire fully-connected graph of pairwise distances between each set of points. This is the fastest to compute and can be very fast for smaller datasets but requires $O(n^2)$ space. 'knn' will sparsify the fully-connected connectivity matrix to save memory and enable much larger inputs. "n_neighbors" will control the amount of memory used and the graph will be connected automatically in the event "n_neighbors" was not large enough to connect it.
n_neighbors	The number of neighbors to compute when connectivity is "knn". Default: 15L.

Value

A clustering object with the following attributes: "n_clusters": The number of clusters found by the algorithm. "children": The children of each non-leaf node. Values less than nrow(x) correspond to leaves of the tree which are the original samples. children[i + 1][1] and children[i + 1][2] were merged to form node (nrow(x) + i) in the i-th iteration. "labels": cluster label of each data point.

Examples

```
library(cuda.ml)
if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  library(MASS)
  library(purrr)

  set.seed(0)

  gen_pts <- function() {
    centers <- list(c(1000, 1000), c(-1000, -1000), c(-1000, 1000))
    pts <- centers |>
      map(\(center) mvrnorm(50, mu = center, Sigma = diag(2)))

    do.call(rbind, pts)
  }

  clust <- cuda_ml_agglomerative_clustering(
    x = gen_pts(),
    metric = "euclidean",
    n_clusters = 3
  )

  print(clust$labels)
}
```

cuda_ml_backend_info *Report native-backend metadata*

Description

Performs cheap, read-only cache and inventory checks for routine inspection. It reports whether the selected backend cache is complete, but does not verify every recorded hash or native registration. It does not create or modify the cache, access the network, inspect an NVIDIA GPU or driver, or load native code.

Usage

```
cuda_ml_backend_info()
```

Details

Use `cuda_ml_runtime_audit()` when an explicit deep integrity check is needed. The audit re-computes recorded hashes, validates native registration, and, for the complete downloaded backend, validates the managed-runtime dependency closure.

Value

A named list describing the selected backend, pinned library versions, cache status, and CPU-only nvForest backend status.

See Also

[cuda_ml_runtime_audit\(\)](#)

<code>cuda_ml_cache_clean</code>	<i>Remove cuda.ml native-backend caches</i>
----------------------------------	---

Description

Removes downloaded and source-built runtime and backend cache generations, including the selected-backend record. Restart R before calling this function if the native backend has been loaded in this process.

Usage

```
cuda_ml_cache_clean()
```

Value

Invisibly returns TRUE.

<code>cuda_ml_dbscan</code>	<i>Run the DBSCAN clustering algorithm.</i>
-----------------------------	---

Description

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

Usage

```
cuda_ml_dbscan(x, min_pts, eps)
```

Arguments

- x** The input matrix or data frame. 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 as its cluster assignment.

Examples

```
library(cuda.ml)
if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  gen_pts <- function() {
    centroids <- list(c(1000, 1000), c(-1000, -1000), c(-1000, 1000))

    pts <- centroids |>
      purrr::map\(centroid) {
        MASS::mvrnorm(10, mu = centroid, Sigma = diag(2))
      })

    do.call(rbind, pts)
  }

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

  print(clusters)
}
```

cuda_ml_elastic_net	<i>Train a linear model using elastic net regression.</i>
---------------------	---

Description

Train a linear model with combined L1 and L2 priors as the regularizer.

Usage

```
cuda_ml_elastic_net(x, ...)

## Default S3 method:
cuda_ml_elastic_net(x, ...)

## S3 method for class 'data.frame'
cuda_ml_elastic_net(
  x,
```

```
    y,
    alpha = 1,
    l1_ratio = 0.5,
    max_iter = 1000L,
    tol = 0.001,
    fit_intercept = TRUE,
    selection = c("cyclic", "random"),
    ...
)

## S3 method for class 'matrix'
cuda_ml_elastic_net(
  x,
  y,
  alpha = 1,
  l1_ratio = 0.5,
  max_iter = 1000L,
  tol = 0.001,
  fit_intercept = TRUE,
  selection = c("cyclic", "random"),
  ...
)

## S3 method for class 'formula'
cuda_ml_elastic_net(
  formula,
  data,
  alpha = 1,
  l1_ratio = 0.5,
  max_iter = 1000L,
  tol = 0.001,
  fit_intercept = TRUE,
  selection = c("cyclic", "random"),
  ...
)

## S3 method for class 'recipe'
cuda_ml_elastic_net(
  x,
  data,
  alpha = 1,
  l1_ratio = 0.5,
  max_iter = 1000L,
  tol = 0.001,
  fit_intercept = TRUE,
  selection = c("cyclic", "random"),
  ...
)
```

Arguments

x	Depending on the context: • A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.
alpha	Positive multiplier of the penalty term. Use <code>cuda_ml_ols()</code> for an unpenalized linear model. Default: 1.
l1_ratio	The ElasticNet mixing parameter, with $0 \leq \text{l1\_ratio} \leq 1$. For <code>l1_ratio = 0</code> the penalty is an L2 penalty. For <code>l1_ratio = 1</code> it is an L1 penalty. For $0 < \text{l1\_ratio} < 1$, the penalty is a combination of L1 and L2. The penalty term is computed using the following formula: $\text{penalty} = \alpha * \text{l1\_ratio} * \text{ w }_1 + 0.5 * \alpha * (1 - \text{l1\_ratio}) * \text{ w }_2^2$ where w _1 is the L1 norm of the coefficients, and w _2 is the L2 norm of the coefficients.
max_iter	The maximum number of coordinate descent iterations. Default: 1000L.
tol	Stop the coordinate descent when the duality gap is below this threshold. Default: $1e-3$.
fit_intercept	If TRUE, then the model tries to correct for the global mean of the response variable. If FALSE, then the model expects data to be centered. Default: TRUE.
selection	If "random", then instead of updating coefficients in cyclic order, a random coefficient is updated in each iteration. Default: "cyclic".
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

An elastic net regressor that can be used with the 'predict' S3 generic to make predictions on new data points.

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  model <- cuda_ml_elastic_net(
    formula = mpg ~ ., data = mtcars, alpha = 1e-3, l1_ratio = 0.6
  )
  predictors <- subset(mtcars, select = -mpg)
  cuda_ml_predictions <- predict(model, predictors)
```

```

# predictions will be comparable to those from a `glmnet` model with
# `lambda` set to 1e-3 and `alpha` set to 0.6
# (in `glmnet`, `lambda` is the weight of the penalty term, and `alpha` is
# the elastic mixing parameter between L1 and L2 penalties.

if (requireNamespace("glmnet", quietly = TRUE)) {
  glmnet_model <- glmnet::glmnet(
    x = as.matrix(predictors), y = mtcars$mpg,
    alpha = 0.6, lambda = 1e-3, nlambdas = 1, standardize = FALSE
  )

  glm_predictions <- predict(
    glmnet_model, as.matrix(predictors),
    s = 0
  )

  print(
    all.equal(
      as.numeric(glm_predictions),
      cuda_ml_predictions$.pred,
      tolerance = 1e-2
    )
  )
}
}

```

cuda_ml_install

Install a cuda.ml native backend

Description

By default, downloads, verifies, extracts, and caches the precompiled backend and its runtime libraries. Alternatively, bootstraps a locked CUDA and RAPIDS build toolchain and compiles the native backend on the host. Calling it again with the same inputs is a no-op. Use device = "cpu" to install the separate nvForest CPU inference backend, roughly 1 MiB to download and 3 MiB when installed, without the complete CUDA and RAPIDS runtime. Installation never occurs implicitly during model loading or prediction.

Usage

```

cuda_ml_install(
  source = FALSE,
  dependencies = "managed",
  architectures = NULL,
  device = c("gpu", "cpu")
)

```

Arguments

source	A logical value. If FALSE, install the requested prebuilt backend. If TRUE, compile the complete GPU backend from the native sources included in the R package. Source installation is not supported for device = "cpu".
dependencies	For a source installation, either "managed" to download and cache the exact locked build dependencies, or "host" to use explicit host installations.
architectures	For a source installation, NULL, "native", "portable", or an explicit semicolon-separated CMake CUDA architecture list. Managed source builds detect CUDA-visible GPUs by default and otherwise use the package's portable architecture list. "native" requires detection, and "portable" forces the package list. Host source builds use CUML_CUDA_ARCHITECTURES when this argument is NULL.
device	Backend to install: "gpu" installs the complete CUDA and RAPIDS backend used for training and GPU inference; "cpu" installs only the CPU nvForest inference backend. CPU installation does not provision the complete CUDA and RAPIDS runtime or any cuML algorithms.

Details

The default cache is `tools::R_user_dir("cuda.ml", "cache")`. Set `CUDA_ML_CACHE_DIR` to use a different cache root. Set `CUDA_ML_BACKEND_MIRROR` to an `https://` or `file://` directory containing the exact locked backend archive.

The CPU-only backend supports nvForest model loading, restoration, and inference. It does not provide cuML training or GPU inference and requires neither an NVIDIA GPU nor an NVIDIA driver. It contains no CUDA runtime libraries. Install the complete backend separately with `cuda_ml_install()` when training or GPU inference is needed.

A managed source installation downloads no precompiled cuda.ml backend. It downloads and verifies the locked CUDA 13.2.2 and RAPIDS 26.06 development artifacts, CMake, and Ninja; builds Treelite 4.7.0 statically; and caches that toolchain. Only Linux x86_64 with glibc 2.28 or newer and GNU C++ 14 or newer are required on the host. When `CUDA_ML_CXX` is unset, the installer prefers g++-14, then g++, on PATH. Set `CUDA_ML_CXX` to override this discovery.

By default, a managed source build uses `nvidia-smi` to detect distinct CUDA-visible GPU compute capabilities and compiles their real targets. It honors `CUDA_VISIBLE_DEVICES`. If detection is unavailable, it uses the package's portable list, so GPU-free build hosts remain supported. Set `architectures = "native"` to require detection or `architectures = "portable"` to force the package list. Native targets usually reduce build time and backend size, but the resulting backend supports only those GPU architectures.

A host source installation makes no downloads. It requires CUDA Toolkit 13.2.2 in `CUDA_HOME`; a `CUML_PREFIX` containing cuML and nvForest 26.06, Treelite 4.7.0 headers, and `lib/libtreelite_static.a`; an explicit CMake CUDA architecture list in `CUML_CUDA_ARCHITECTURES`; and GNU C++ 14 or newer in `CUDA_ML_CXX`. CMake 3.21.1 or newer must be on PATH.

Value

Invisibly returns TRUE.

Examples

```
## Not run:
cuda_ml_install()

cuda_ml_install(device = "cpu")

cuda_ml_install(source = TRUE)

cuda_ml_install(source = TRUE, architectures = "native")

cuda_ml_install(source = TRUE, architectures = "portable")

Sys.setenv(
  CUDA_HOME = "/usr/local/cuda-13.2",
  CUML_PREFIX = "/opt/rapids-26.06",
  CUML_CUDA_ARCHITECTURES = "86-real",
  CUDA_ML_CXX = "/usr/bin/g++-14"
)
cuda_ml_install(source = TRUE, dependencies = "host")

## End(Not run)
```

cuda_ml_kmeans	<i>Run the k-means clustering algorithm.</i>
----------------	--

Description

Run the k-means clustering algorithm.

Usage

```
cuda_ml_kmeans(
  x,
  k,
  max_iters = 300,
  tol = 0,
  init_method = c("kmeans++", "random"),
  seed = 0L
)
```

Arguments

x	The input matrix or data frame. 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.
tol	Relative tolerance with regards to inertia to declare convergence. Default: 0 (i.e., do not use inertia-based stopping criterion).

init_method	Method for initializing the centroids. Valid methods include "kmeans++", "random", or a matrix of k rows, each row specifying the initial value of a centroid. Default: "kmeans++".
seed	Seed to the random number generator. Default: 0.

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(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  oils <- modeldata::oils
  oil_predictors <- oils |>
    subset(select = -class) |>
    scale()

  kclust <- cuda_ml_kmeans(
    oil_predictors,
    k = 7, max_iters = 100
  )

  print(kclust)
}
```

cuda_ml_knn

Build a KNN model.

Description

Build a k-nearest-neighbor model for classification or regression tasks.

Usage

```
cuda_ml_knn(x, ...)
```

Default S3 method:

```
cuda_ml_knn(x, ...)
```

S3 method for class 'data.frame'

```
cuda_ml_knn(
  x,
  y,
  algo = c("brute", "ivfflat", "ivfpq"),
  metric = c("euclidean", "l2", "l1", "cityblock", "taxicab", "manhattan", "braycurtis"),
```

```

    "canberra", "minkowski", "lp", "chebyshev", "linf", "jensenshannon", "cosine",
    "correlation"),
  p = 2,
  neighbors = 5L,
  ...
)

## S3 method for class 'matrix'
cuda_ml_knn(
  x,
  y,
  algo = c("brute", "ivfflat", "ivfpq"),
  metric = c("euclidean", "l2", "l1", "cityblock", "taxicab", "manhattan", "braycurtis",
    "canberra", "minkowski", "lp", "chebyshev", "linf", "jensenshannon", "cosine",
    "correlation"),
  p = 2,
  neighbors = 5L,
  ...
)

## S3 method for class 'formula'
cuda_ml_knn(
  formula,
  data,
  algo = c("brute", "ivfflat", "ivfpq"),
  metric = c("euclidean", "l2", "l1", "cityblock", "taxicab", "manhattan", "braycurtis",
    "canberra", "minkowski", "lp", "chebyshev", "linf", "jensenshannon", "cosine",
    "correlation"),
  p = 2,
  neighbors = 5L,
  ...
)

## S3 method for class 'recipe'
cuda_ml_knn(
  x,
  data,
  algo = c("brute", "ivfflat", "ivfpq"),
  metric = c("euclidean", "l2", "l1", "cityblock", "taxicab", "manhattan", "braycurtis",
    "canberra", "minkowski", "lp", "chebyshev", "linf", "jensenshannon", "cosine",
    "correlation"),
  p = 2,
  neighbors = 5L,
  ...
)

```

Arguments

x Depending on the context:

	• A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.
algo	The query algorithm to use. For most workflows, pass one of {"brute", "ivfflat", "ivfpq"} or a KNN algorithm specification constructed using the <code>cuda_ml_knn_algo_*</code> family of functions. If the algorithm is specified by one of the <code>cuda_ml_knn_algo_*</code> functions, then values of all required parameters of the algorithm will need to be specified explicitly. If the algorithm is specified by a character vector, then parameters for the algorithm are generated automatically. Descriptions of supported algorithms: • "brute": for brute-force, slow but produces exact results. • "ivfflat": for inverted file, divide the dataset in partitions and perform search on relevant partitions only. • "ivfpq": for inverted file and product quantization (vectors are divided into sub-vectors, and each sub-vector is encoded using intermediary k-means clusterings to provide partial information). Default: "brute".
metric	Distance metric to use. Must be one of {"euclidean", "l2", "l1", "cityblock", "taxicab", "manhattan", "braycurtis", "canberra", "minkowski", "lp", "chebyshev", "linf", "jensenshannon", "cosine", "correlation"}. The approximate algorithms support only "euclidean", "l2", "cosine", and "correlation". Default: "euclidean".
p	Parameter for the Minkowski metric. If $p = 1$, then the metric is equivalent to manhattan distance (l1). If $p = 2$, the metric is equivalent to euclidean distance (l2).
neighbors	Number of nearest neighbors to query. Default: 5L.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

A KNN model that can be used with the 'predict' S3 generic to make predictions on new data points. The model object contains the following:

- "knn_index": a GPU pointer to the KNN index.
- "algo": enum value of the algorithm being used for the KNN query.
- "metric": enum value of the distance metric used in KNN computations.
- "p": parameter for the Minkowski metric.
- "n_samples": number of input data points.
- "n_dims": dimension of each input data point.

Examples

```

library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  library(MASS)
  library(purrr)

  set.seed(0)

  centers <- list(c(3, 3), c(-3, -3), c(-3, 3))

  gen_pts <- function(cluster_sz) {
    pts <- centers |>
      map(\(center) mvrnorm(cluster_sz, mu = center, Sigma = diag(2)))

    do.call(rbind, pts)
  }

  gen_labels <- function(cluster_sz) {
    seq_along(centers) |>
      supply(\(x) rep(x, cluster_sz)) |>
      factor()
  }

  sample_cluster_sz <- 1000
  sample_pts <- cbind(
    gen_pts(sample_cluster_sz) |> as.data.frame(),
    label = gen_labels(sample_cluster_sz)
  )

  model <- cuda_ml_knn(
    label ~ ., sample_pts, algo = "ivfflat", metric = "euclidean"
  )

  test_cluster_sz <- 10
  test_pts <- gen_pts(test_cluster_sz) |> as.data.frame()

  predictions <- predict(model, test_pts)
  print(predictions, n = 30)
}

```

 cuda_ml_knn_algo

Configure an approximate KNN query algorithm

Description

For the main path, pass "ivfflat" or "ivfpq" directly to the algo argument of cuda_ml_knn(); cuda.ml then lets the backend choose the index parameters. Use these constructors only when those parameters need to be set explicitly.

Usage

```

cuda_ml_knn_algo_ivfflat(nlist, nprobe)

cuda_ml_knn_algo_ivfpq(nlist, nprobe, m, n_bits)

```

Arguments

<code>nlist</code>	Number of cells to partition dataset into.
<code>nprobe</code>	At query time, the number of cells used for approximate nearest neighbor search.
<code>m</code>	Number of subquantizers.
<code>n_bits</code>	Bits allocated per subquantizer, from 4 to 8. The product of <code>m</code> and <code>n_bits</code> must be divisible by 8.

Details

Both algorithms partition the training data into `nlist` cells and search `nprobe` cells for each query. IVFFlat stores the original vectors and therefore needs only those two parameters. IVFPQ also compresses vectors using product quantization, so it additionally requires the number of subquantizers (`m`) and the bits allocated to each subquantizer (`n_bits`). The distinct constructors keep the required parameters for each algorithm explicit.

Value

A KNN algorithm specification to pass to the `algo` argument of `cuda_ml_knn()`.

See Also

[cuda_ml_knn\(\)](#)

<code>cuda_ml_lasso</code>	<i>Train a linear model using LASSO regression.</i>
----------------------------	---

Description

Train a linear model using LASSO (Least Absolute Shrinkage and Selection Operator) regression.

Usage

```

cuda_ml_lasso(x, ...)

## Default S3 method:
cuda_ml_lasso(x, ...)

## S3 method for class 'data.frame'
cuda_ml_lasso(
  x,

```

```

    y,
    alpha = 1,
    max_iter = 1000L,
    tol = 0.001,
    fit_intercept = TRUE,
    selection = c("cyclic", "random"),
    ...
)

## S3 method for class 'matrix'
cuda_ml_lasso(
  x,
  y,
  alpha = 1,
  max_iter = 1000L,
  tol = 0.001,
  fit_intercept = TRUE,
  selection = c("cyclic", "random"),
  ...
)

## S3 method for class 'formula'
cuda_ml_lasso(
  formula,
  data,
  alpha = 1,
  max_iter = 1000L,
  tol = 0.001,
  fit_intercept = TRUE,
  selection = c("cyclic", "random"),
  ...
)

## S3 method for class 'recipe'
cuda_ml_lasso(
  x,
  data,
  alpha = 1,
  max_iter = 1000L,
  tol = 0.001,
  fit_intercept = TRUE,
  selection = c("cyclic", "random"),
  ...
)

```

Arguments

x Depending on the context:

	• A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.
alpha	Positive multiplier of the L1 penalty term. Use <code>cuda_ml_ols()</code> for an unpenalized linear model. Default: 1.
max_iter	The maximum number of coordinate descent iterations. Default: 1000L.
tol	Stop the coordinate descent when the duality gap is below this threshold. Default: 1e-3.
fit_intercept	If TRUE, then the model tries to correct for the global mean of the response variable. If FALSE, then the model expects data to be centered. Default: TRUE.
selection	If "random", then instead of updating coefficients in cyclic order, a random coefficient is updated in each iteration. Default: "cyclic".
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

A LASSO regressor that can be used with the 'predict' S3 generic to make predictions on new data points.

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  model <- cuda_ml_lasso(formula = mpg ~ ., data = mtcars, alpha = 1e-3)
  predictors <- subset(mtcars, select = -mpg)
  cuda_ml_predictions <- predict(model, predictors)

  # predictions will be comparable to those from a `glmnet` model with
  # `lambda` set to 1e-3 and `alpha` set to 1
  # (in `glmnet`, `lambda` is the weight of the penalty term, and `alpha` is
  # the elastic mixing parameter between L1 and L2 penalties.

  if (requireNamespace("glmnet", quietly = TRUE)) {
    glmnet_model <- glmnet::glmnet(
      x = as.matrix(predictors), y = mtcars$mpg,
      alpha = 1, lambda = 1e-3, nlambda = 1, standardize = FALSE
    )

    glm_predictions <- predict(
```

```

      glmnet_model, as.matrix(predictors),
      s = 0
    )

    print(
      all.equal(
        as.numeric(glm_predictions),
        cuda_ml_predictions$.pred,
        tolerance = 1e-2
      )
    )
  }
}

```

cuda_ml_linear_reg	<i>Train a regularized linear regression model</i>
--------------------	--

Description

This is the tidymodels-style linear regression interface. It dispatches to `cuda_ml_ols()`, `cuda_ml_ridge()`, `cuda_ml_lasso()`, or `cuda_ml_elastic_net()` according to `penalty` and `mixture`. The named model functions remain available when direct control over their solver arguments is needed.

Usage

```
cuda_ml_linear_reg(formula, data, penalty = NULL, mixture = NULL, ...)
```

Arguments

<code>formula</code>	A model formula.
<code>data</code>	A data frame containing predictors and outcome.
<code>penalty</code>	A non-negative regularization strength, or <code>NULL</code> for no regularization.
<code>mixture</code>	The proportion of regularization assigned to the L1 penalty, between 0 and 1. When <code>NULL</code> , a lasso penalty is used.
<code>...</code>	Arguments passed to the selected named model function.

Value

A fitted `cuda.ml` linear model.

`cuda_ml_logistic_reg` *Train a logistic or multinomial regression model*

Description

Fits a factor outcome with cuML's quasi-Newton solver. Regularization follows tidymodels conventions: `penalty` is the total regularization strength and `mixture` is the proportion assigned to the L1 penalty. Set `mixture = 0` for ridge, `mixture = 1` for lasso, or use an intermediate value for an elastic-net penalty. Normalize predictors with a recipe before fitting when scaling is required.

Usage

```
cuda_ml_logistic_reg(x, ...)

## Default S3 method:
cuda_ml_logistic_reg(x, ...)

## S3 method for class 'data.frame'
cuda_ml_logistic_reg(
  x,
  y,
  fit_intercept = TRUE,
  penalty = NULL,
  mixture = 0,
  tol = 1e-04,
  class_weight = NULL,
  sample_weight = NULL,
  max_iter = 1000L,
  linesearch_max_iter = 50L,
  lbfgs_memory = 5L,
  penalty_normalized = TRUE,
  ...
)

## S3 method for class 'matrix'
cuda_ml_logistic_reg(
  x,
  y,
  fit_intercept = TRUE,
  penalty = NULL,
  mixture = 0,
  tol = 1e-04,
  class_weight = NULL,
  sample_weight = NULL,
  max_iter = 1000L,
  linesearch_max_iter = 50L,
  lbfgs_memory = 5L,
```

```

    penalty_normalized = TRUE,
    ...
)

## S3 method for class 'formula'
cuda_ml_logistic_reg(
  formula,
  data,
  fit_intercept = TRUE,
  penalty = NULL,
  mixture = 0,
  tol = 1e-04,
  class_weight = NULL,
  sample_weight = NULL,
  max_iter = 1000L,
  linesearch_max_iter = 50L,
  lbfgs_memory = 5L,
  penalty_normalized = TRUE,
  ...
)

## S3 method for class 'recipe'
cuda_ml_logistic_reg(
  x,
  data,
  fit_intercept = TRUE,
  penalty = NULL,
  mixture = 0,
  tol = 1e-04,
  class_weight = NULL,
  sample_weight = NULL,
  max_iter = 1000L,
  linesearch_max_iter = 50L,
  lbfgs_memory = 5L,
  penalty_normalized = TRUE,
  ...
)

```

Arguments

x	Depending on the context: • A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.

fit_intercept	If TRUE, then the model tries to correct for the global mean of the response variable. If FALSE, then the model expects data to be centered. Default: TRUE.
penalty	A non-negative regularization strength, or NULL for no regularization. Default: NULL.
mixture	The proportion of regularization assigned to the L1 penalty, between 0 and 1. Default: 0.
tol	Stopping tolerance. Default: 1e-4.
class_weight	NULL, "balanced", or a named numeric vector with one non-negative weight per outcome level.
sample_weight	A numeric vector with one non-negative weight per training observation, or NULL.
max_iter	Maximum solver iterations. Default: 1000L.
linesearch_max_iter	Maximum line-search iterations per solver iteration. Default: 50L.
lbfgs_memory	Number of vectors retained by the L-BFGS approximation. Default: 5L.
penalty_normalized	Whether to normalize regularization by the number of observations. Default: TRUE.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

A classification model for use with `predict()`.

cuda_ml_nvforest_export

Export and import an nvForest checkpoint pair

Description

`cuda_ml_nvforest_export()` writes a standard Treelite checkpoint and a cuda.ml JSON sidecar. The checkpoint contains the device-neutral tree ensemble. The sidecar retains cuda.ml metadata, class labels, prediction precision, random-forest probability semantics, and the R preprocessing blueprint needed for a complete cuda.ml round-trip. `cuda_ml_nvforest_import()` restores the pair on a caller-selected inference device.

Usage

```

cuda_ml_nvforest_export(object, directory, prefix, overwrite = FALSE)

cuda_ml_nvforest_import(
  directory,
  prefix,
  device = c("gpu", "cpu"),
  device_id = NULL,
  layout = c("depth_first", "breadth_first", "layered"),
  precision = NULL,
  default_chunk_size = NULL,
  align_bytes = NULL
)

```

Arguments

<code>object</code>	An nvForest-backed model.
<code>directory</code>	An existing output directory.
<code>prefix</code>	A non-empty filename prefix without directory components.
<code>overwrite</code>	Whether to replace both existing output files. The default is FALSE.
<code>device</code>	Inference device: "gpu" or "cpu". The default is "gpu".
<code>device_id</code>	GPU device identifier, or NULL for the current device.
<code>layout</code>	Tree layout.
<code>precision</code>	Native, single, or double precision. NULL retains the exported model's prediction precision.
<code>default_chunk_size</code>	Default prediction chunk size, or NULL to use nvForest's heuristic.
<code>align_bytes</code>	Memory alignment, or NULL for the device default.

Value

`cuda_ml_nvforest_export()` invisibly returns a named character vector containing the absolute checkpoint and metadata paths. `cuda_ml_nvforest_import()` returns the restored nvForest-backed model.

Files

The function writes exactly '`<prefix>.treelite.checkpoint`' and '`<prefix>.cuda-ml.json`'. The JSON records the checkpoint's relative filename, size, and SHA-256 digest. It does not record inference device, layout, chunk size, memory alignment, or GPU device identifier.

Other Treelite consumers can load the checkpoint without the JSON. They must supply numeric predictors in the recorded processed feature order when feature names are available, or in the checkpoint's original positional order otherwise. They must also implement any class-label and postprocessing behavior described by the sidecar.

Loading the bare checkpoint with `cuda_ml_nvforest_load_model(model_type = "treelite_checkpoint")` likewise omits the sidecar's preprocessing, original class labels, cuda.ml model class, and random-forest probability semantics. Use `cuda_ml_nvforest_import()` for an exact cuda.ml round-trip.

Persistence choices

Use `cuda_ml_serialize()` and `cuda_ml_unserialize()` for one R-native state value. The checkpoint pair is useful when the Treelite model must also be independently available. A bundle is optional wrapping around the R-native state and is not required for either workflow.

`cuda.ml` validates the sidecar and selected backend before import. Prepare the backend first with `cuda_ml_install()` for GPU operation or `cuda_ml_install(device = "cpu")` for CPU-only inference. Import never downloads a backend.

Trust

The JSON embeds an R-serialized hardhat blueprint so that formula and recipe preprocessing round-trip. Import only artifacts from trusted sources, as with `readRDS()` and `cuda_ml_unserialize()`. The recorded SHA-256 digest checks integrity, not authenticity.

See Also

`cuda_ml_nvforest_load_model()` and `cuda_ml_serialize()`

`cuda_ml_nvforest_info` *Inspect an nvForest model*

Description

Inspect an nvForest model

Usage

```
cuda_ml_nvforest_info(object)
```

Arguments

`object` An nvForest-backed model.

Value

A named list with:

`task_type` One of "binary_classification", "multiclass_classification", or "regression".

`num_classes`, `num_features`, `num_outputs`, **and** `num_trees` Model dimensions.

`has_vector_leaves`, `average_tree_output`, **and** `has_probability_output` Logical model properties.

`device`, `device_id`, `layout`, **and** `precision` Resolved inference configuration.

`default_chunk_size` **and** `align_bytes` Native chunk and memory-alignment settings.

`treelite_postprocessor` The model's Treelite postprocessor.

cuda_ml_nvforest_leaf_ids
Return terminal leaf identifiers

Description

Return terminal leaf identifiers

Usage

```
cuda_ml_nvforest_leaf_ids(object, new_data, chunk_size = NULL)
```

Arguments

object	An nvForest-backed model.
new_data	Numeric predictor data.
chunk_size	Native prediction chunk size, or NULL for the model default. It controls native batching and does not limit the size of the returned R object.

Value

An integer matrix with one row per observation and one column per tree.

cuda_ml_nvforest_load_model
Load a tree ensemble with nvForest

Description

Loads an XGBoost, LightGBM, or Treelite model with the current nvForest API. The model's classification or regression task is read from Treelite metadata rather than supplied separately.

Usage

```
cuda_ml_nvforest_load_model(  
  model_file,  
  model_type = NULL,  
  class_levels = NULL,  
  device = c("gpu", "cpu"),  
  device_id = NULL,  
  layout = c("depth_first", "breadth_first", "layered"),  
  precision = c("native", "single", "double"),  
  default_chunk_size = NULL,  
  align_bytes = NULL  
)
```

Arguments

model_file	Path to a model file.
model_type	File format, or NULL to infer it from a recognized filename suffix. See Model formats .
class_levels	Optional class labels in model-output order. When omitted, classifiers use "0", "1", and so on.
device	Inference device: "gpu" or "cpu". The default is "gpu".
device_id	GPU device identifier, or NULL for the current device.
layout	Tree layout.
precision	Native, single, or double precision.
default_chunk_size	Default prediction chunk size, or NULL to use nvForest's heuristic.
align_bytes	Memory alignment, or NULL for the device default.

Value

An nvForest model for use with `predict()`.

Model formats

The supported `model_type` values are:

- "xgboost_ubj" for XGBoost UBJSON;
- "xgboost_json" for XGBoost JSON;
- "xgboost_legacy" for the legacy XGBoost binary format;
- "lightgbm" for LightGBM text models; and
- "treelite_checkpoint" for Treelite checkpoints.

When `model_type = NULL`, the format is inferred only from the case-insensitive filename suffix: `'.ubj'`, `'.json'`, `'.model'`, and `'.txt'` map to "xgboost_ubj", "xgboost_json", "xgboost_legacy", and "lightgbm", respectively. Treelite checkpoints have no inferred suffix and require `model_type = "treelite_checkpoint"`. Inference does not inspect file contents; use an explicit type when the suffix does not identify the format.

Runtime requirements

GPU inference requires the complete, roughly 1.6 GiB runtime installed by `cuda_ml_install()` and a supported NVIDIA GPU and driver. For CPU-only deployment, install the separate, roughly 3 MiB backend with `cuda_ml_install(device = "cpu")`. It does not install cuML or the complete managed CUDA and RAPIDS runtime, and it requires neither an NVIDIA GPU nor an NVIDIA driver. An existing complete backend installation can also execute nvForest models on CPU; the separate backend avoids that runtime in CPU-only environments.

Persistence

Persist nvForest models with `cuda_ml_serialize()` and restore them with `cuda_ml_unserialize()`. Current states do not record CPU or GPU placement. Select the deployment device when restoring, for example `cuda_ml_unserialize(state, device = "cpu")`; GPU is the default. Tree layout, chunk size, memory alignment, and GPU device identifier are likewise restore-time settings. Prediction precision is retained unless explicitly overridden. `cuda.ml` validates the saved state and selected backend before restoration.

To create a standard Treelite checkpoint together with the metadata needed for a complete `cuda.ml` round-trip, use `cuda_ml_nvforest_export()` and restore the pair with `cuda_ml_nvforest_import()`.

See Also

`cuda_ml_nvforest_info()`, `cuda_ml_nvforest_leaf_ids()`, `cuda_ml_nvforest_predict_per_tree()`, and `vignette("nvforest")`

`cuda_ml_nvforest_predict_per_tree`

Return individual-tree predictions

Description

Return individual-tree predictions

Usage

```
cuda_ml_nvforest_predict_per_tree(object, new_data, chunk_size = NULL)
```

Arguments

<code>object</code>	An nvForest-backed model.
<code>new_data</code>	Numeric predictor data.
<code>chunk_size</code>	Native prediction chunk size, or NULL for the model default. It controls native batching and does not limit the size of the returned R object.

Value

For scalar-leaf models, a numeric matrix with one column per tree. For vector-leaf models, a numeric array indexed by observation, tree, and output.

Memory use

The complete result is materialized in R: rows by trees for scalar-leaf models and rows by trees by outputs for vector-leaf models. The `chunk_size` argument controls native prediction work but does not bound the memory required by the R result.

See Also

`cuda_ml_nvforest_leaf_ids()`

 cuda_ml_ols

Train an OLS model.

Description

Train an ordinary least squares (OLS) model for regression tasks.

Usage

```

cuda_ml_ols(x, ...)

## Default S3 method:
cuda_ml_ols(x, ...)

## S3 method for class 'data.frame'
cuda_ml_ols(x, y, method = c("svd", "eig", "qr"), fit_intercept = TRUE, ...)

## S3 method for class 'matrix'
cuda_ml_ols(x, y, method = c("svd", "eig", "qr"), fit_intercept = TRUE, ...)

## S3 method for class 'formula'
cuda_ml_ols(
  formula,
  data,
  method = c("svd", "eig", "qr"),
  fit_intercept = TRUE,
  ...
)

## S3 method for class 'recipe'
cuda_ml_ols(x, data, method = c("svd", "eig", "qr"), fit_intercept = TRUE, ...)
```

Arguments

x	Depending on the context: • A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.
method	Must be one of {"svd", "eig", "qr"}. • "svd": compute SVD decomposition using Jacobi iterations. • "eig": use an eigendecomposition of the covariance matrix.

- "qr": use the QR decomposition algorithm and solve $Rx = Q^T y$.

If the number of features is larger than the sample size, then the "svd" algorithm will be force-selected because it is the only algorithm that can support this type of scenario.

Default: "svd".

fit_intercept	If TRUE, then the model tries to correct for the global mean of the response variable. If FALSE, then the model expects data to be centered. Default: TRUE.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

An OLS regressor that can be used with the 'predict' S3 generic to make predictions on new data points.

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  model <- cuda_ml_ols(formula = mpg ~ ., data = mtcars, method = "qr")
  predictors <- subset(mtcars, select = -mpg)
  predictions <- predict(model, predictors)

  # predictions will be comparable to those from a `stats::lm` model
  lm_model <- stats::lm(formula = mpg ~ ., data = mtcars, method = "qr")
  lm_predictions <- predict(lm_model, predictors)

  print(
    all.equal(
      as.numeric(lm_predictions),
      predictions$.pred,
      tolerance = 1e-3
    )
  )
}
```

cuda_ml_pca

Perform principal component analysis.

Description

Compute principal component(s) of the input data. Each feature from the input will be mean-centered (but not scaled) before the SVD computation takes place.

Usage

```

cuda_ml_pca(
  x,
  n_components = NULL,
  eig_algo = c("dq", "jacobi"),
  tol = 1e-07,
  n_iters = 15L,
  whiten = FALSE,
  transform_input = TRUE
)

```

Arguments

<code>x</code>	The input matrix or data frame. Each data point should be a row and should consist of numeric values only.
<code>n_components</code>	Number of principal component(s) to keep. Default: <code>min(nrow(x), ncol(x))</code> .
<code>eig_algo</code>	Eigen decomposition algorithm to be applied to the covariance matrix. Valid choices are "dq" (divid-and-conquer method for symmetric matrices) and "jacobi" (the Jacobi method for symmetric matrices). Default: "dq".
<code>tol</code>	Tolerance for singular values computed by the Jacobi method. Default: <code>1e-7</code> .
<code>n_iters</code>	Maximum number of iterations for the Jacobi method. Default: <code>15</code> .
<code>whiten</code>	If <code>TRUE</code> , then de-correlate all components, making each component have unit variance and removing multi-collinearity. Default: <code>FALSE</code> .
<code>transform_input</code>	If <code>TRUE</code> , then compute an approximate representation of the input data. Default: <code>TRUE</code> .

Value

A PCA model object with the following attributes:

- `"components"`: a matrix of `n_components` rows containing the top principal components.
- `"explained_variance"`: amount of variance within the input data explained by each component.
- `"explained_variance_ratio"`: fraction of variance within the input data explained by each component.
- `"singular_values"`: singular values (non-negative) corresponding to the top principal components.
- `"mean"`: the column-wise mean of `x` which was used to mean-center `x` first.
- `"transformed_data"`: (only present if `"transform_input"` is set to `TRUE`) an approximate representation of input data based on principal components.
- `"pca_params"`: opaque pointer to PCA parameters which will be used for performing inverse transforms.

The model object can be used as input to the `cuda_ml_inverse_transform()` function to map a representation based on principal components back to the original feature space.

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  oils <- modeldata::oils
  oil_predictors <- oils |>
    subset(select = -class) |>
    scale()

  oil_pca <- cuda_ml_pca(oil_predictors, n_components = 3)
  print(oil_pca)
}
```

cuda_ml_rand_forest	<i>Train a random forest model</i>
---------------------	------------------------------------

Description

Trains a cuML random forest for classification or regression and returns an nvForest-backed model for inference.

Usage

```
cuda_ml_rand_forest(x, ...)

## Default S3 method:
cuda_ml_rand_forest(x, ...)

## S3 method for class 'data.frame'
cuda_ml_rand_forest(
  x,
  y,
  mtry = NULL,
  trees = 100L,
  min_n = 2L,
  bootstrap = TRUE,
  sample_fraction = 1,
  max_depth = 16L,
  max_leaves = Inf,
  n_bins = 128L,
  min_samples_leaf = 1L,
  split_criterion = NULL,
  min_impurity_decrease = 0,
  max_batch_size = 4096L,
  n_streams = 4L,
  seed = NULL,
  ...
)
```

```
)

## S3 method for class 'matrix'
cuda_ml_rand_forest(
  x,
  y,
  mtry = NULL,
  trees = 100L,
  min_n = 2L,
  bootstrap = TRUE,
  sample_fraction = 1,
  max_depth = 16L,
  max_leaves = Inf,
  n_bins = 128L,
  min_samples_leaf = 1L,
  split_criterion = NULL,
  min_impurity_decrease = 0,
  max_batch_size = 4096L,
  n_streams = 4L,
  seed = NULL,
  ...
)

## S3 method for class 'formula'
cuda_ml_rand_forest(
  formula,
  data,
  mtry = NULL,
  trees = 100L,
  min_n = 2L,
  bootstrap = TRUE,
  sample_fraction = 1,
  max_depth = 16L,
  max_leaves = Inf,
  n_bins = 128L,
  min_samples_leaf = 1L,
  split_criterion = NULL,
  min_impurity_decrease = 0,
  max_batch_size = 4096L,
  n_streams = 4L,
  seed = NULL,
  ...
)

## S3 method for class 'recipe'
cuda_ml_rand_forest(
  x,
  data,
```

```

    mtry = NULL,
    trees = 100L,
    min_n = 2L,
    bootstrap = TRUE,
    sample_fraction = 1,
    max_depth = 16L,
    max_leaves = Inf,
    n_bins = 128L,
    min_samples_leaf = 1L,
    split_criterion = NULL,
    min_impurity_decrease = 0,
    max_batch_size = 4096L,
    n_streams = 4L,
    seed = NULL,
    ...
)

```

Arguments

x	Depending on the context: • A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.
mtry	Number of predictors sampled at each split. When NULL, classification uses the square root of the predictor count and regression uses all predictors.
trees	Number of trees. Default: 100L.
min_n	Minimum observations required to split a node. Default: 2L.
bootstrap	Whether to sample observations with replacement.
sample_fraction	Proportion of rows used for each tree, between 0 and 1. This is separate from mtry, which controls predictor sampling.
max_depth	Maximum tree depth. Default: 16L.
max_leaves	Maximum leaves per tree, or Inf for no limit.
n_bins	Number of candidate split bins. Default: 128L.
min_samples_leaf	Minimum observations in a leaf. Default: 1L.
split_criterion	Split criterion, or NULL for the mode default. Classification supports "gini" and "entropy"; regression supports "mse", "poisson", "gamma", and "inverse_gaussian".
min_impurity_decrease	Minimum impurity decrease required for a split.

max_batch_size	Maximum nodes processed in one batch. Default: 4096L.
n_streams	Number of CUDA streams used while fitting. Default: 4L.
seed	Random seed forwarded to cuML. When NULL, a seed is drawn from R's random-number generator, so <code>set.seed()</code> controls the fit.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

A random forest model for use with `predict()`.

Deployment

Training uses cuML and requires the complete GPU backend installed by `cuda_ml_install()`. Persist the fitted model with `cuda_ml_serialize()`; the current state is device neutral. A host without a GPU can install the separate CPU inference backend with `cuda_ml_install(device = "cpu")` and restore the state with `cuda_ml_unserialize(state, device = "cpu")`. The CPU backend is roughly 3 MiB installed and does not include cuML or the complete managed CUDA and RAPIDS runtime. To create an independently usable Treelite checkpoint and a cuda.ml JSON sidecar instead, use `cuda_ml_nvforest_export()` and `cuda_ml_nvforest_import()`.

cuda_ml_ridge	<i>Train a linear model using ridge regression.</i>
---------------	---

Description

Train a linear model with L2 regularization.

Usage

```

cuda_ml_ridge(x, ...)

## Default S3 method:
cuda_ml_ridge(x, ...)

## S3 method for class 'data.frame'
cuda_ml_ridge(x, y, alpha = 1, fit_intercept = TRUE, ...)

## S3 method for class 'matrix'
cuda_ml_ridge(x, y, alpha = 1, fit_intercept = TRUE, ...)

## S3 method for class 'formula'
cuda_ml_ridge(formula, data, alpha = 1, fit_intercept = TRUE, ...)

## S3 method for class 'recipe'
cuda_ml_ridge(x, data, alpha = 1, fit_intercept = TRUE, ...)
```

Arguments

x	Depending on the context: • A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.
alpha	Positive multiplier of the L2 penalty term. Use <code>cuda_ml_ols()</code> for an unpenalized linear model. Default: 1.
fit_intercept	If TRUE, then the model tries to correct for the global mean of the response variable. If FALSE, then the model expects data to be centered. Default: TRUE.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

A ridge regressor that can be used with the 'predict' S3 generic to make predictions on new data points.

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  model <- cuda_ml_ridge(formula = mpg ~ ., data = mtcars, alpha = 1e-3)
  predictors <- subset(mtcars, select = -mpg)
  cuda_ml_predictions <- predict(model, predictors)

  # predictions will be comparable to those from a `glmnet` model with
  # `lambda` set to 2e-3 and `alpha` set to 0
  # (in `glmnet`, `lambda` is the weight of the penalty term, and `alpha` is
  # the elastic mixing parameter between L1 and L2 penalties.

  if (requireNamespace("glmnet", quietly = TRUE)) {
    glmnet_model <- glmnet::glmnet(
      x = as.matrix(predictors), y = mtcars$mpg,
      alpha = 0, lambda = 2e-3, nlambda = 1, standardize = FALSE
    )

    glmnet_predictions <- predict(
      glmnet_model, as.matrix(predictors),
      s = 0
    )
  }
}
```

```
    print(
      all.equal(
        as.numeric(glmnet_predictions),
        cuda_ml_predictions$.pred,
        tolerance = 1e-3
      )
    )
  }
}
```

cuda_ml_runtime_audit *Audit the installed native backend*

Description

Performs an explicit deep integrity check. It recomputes the hashes recorded when the requested backend was installed and validates native registration, loading the backend temporarily when needed. For the complete downloaded backend, it also validates the managed-runtime dependency closure.

Usage

```
cuda_ml_runtime_audit(device = c("gpu", "cpu"))
```

Arguments

device	Backend to audit: the complete "gpu" backend or the CPU-only nvForest backend.
--------	--

Details

Use `cuda_ml_backend_info()` for routine, read-only inspection. That function performs only fast cache and inventory checks and does not load native code. Ordinary runtime reuse likewise performs only fast marker, inventory, size, and link checks.

Value

Invisibly returns TRUE.

See Also

[cuda_ml_backend_info\(\)](#)

cuda_ml_serialize	<i>Save and restore supported cuda.ml models</i>
-------------------	--

Description

cuda_ml_serialize() saves the explicit state of a fitted cuda.ml model. cuda_ml_unserialize() restores that state as a fitted model.

Usage

```
cuda_ml_serialize(model, connection = NULL, ...)
```

```
cuda_ml_unserialize(
  connection,
  ...,
  device = NULL,
  device_id = NULL,
  layout = NULL,
  precision = NULL,
  default_chunk_size = NULL,
  align_bytes = NULL
)
```

Arguments

model	The model object.
connection	For cuda_ml_serialize(), a file path, an open connection, or NULL; a file path writes a gzip-compressed state and NULL returns the state as a raw vector. For cuda_ml_unserialize(), a file path, an open connection, or a raw vector.
...	Additional arguments passed to base::serialize() or base::unserialize().
device, device_id, layout, precision, default_chunk_size, align_bytes	Named nvForest inference options. They are supported only for nvForest and random-forest states. When device is omitted, those states restore for GPU inference. When precision is omitted, the saved prediction precision is used. The remaining omitted options use nvForest defaults.

Value

cuda_ml_serialize() returns NULL when writing to a file or connection and otherwise returns a raw vector. cuda_ml_unserialize() returns the restored fitted model.

Supported models

Explicit state is supported for:

- OLS, ridge, lasso, elastic-net, and SGD linear models;
- logistic and multinomial regression;

- PCA;
- binary and one-vs-rest SVC models and SVR models;
- UMAP;
- random forests and other nvForest-backed models.

KNN and TSVD fits are not currently supported. The pinned KNN API does not expose portable approximate-index state, and the current TSVD binding retains native transform parameters that cuda.ml does not reconstruct.

Deployment

cuda.ml validates the model state and required backend before loading it. Prepare the backend in the target process with `cuda_ml_install()` for GPU operation or `cuda_ml_install(device = "cpu")` for CPU-only nvForest inference.

Random-forest and nvForest states contain device-neutral Treelite model bytes. They retain prediction precision, class labels, preprocessing, and model semantics, but not the inference device, device identifier, tree layout, chunk size, or memory alignment. Select those settings while restoring; omitting device selects GPU inference.

`bundle::bundle()` stores the same explicit state. For an nvForest-backed model, the bundle also stores its chosen deployment device separately from the device-neutral state. A bundle is not required for deployment; `cuda_ml_serialize()` returns the complete state artifact directly.

See Also

[serialize](#), [unserialize](#), and [bundle](#)

cuda_ml_sgd

Train a linear model using mini-batch stochastic gradient descent.

Description

Train a linear model using mini-batch stochastic gradient descent.

Usage

```
cuda_ml_sgd(x, ...)

## Default S3 method:
cuda_ml_sgd(x, ...)

## S3 method for class 'data.frame'
cuda_ml_sgd(
  x,
  y,
  fit_intercept = TRUE,
  penalty = c("none", "l1", "l2", "elasticnet"),
```

```
    alpha = 1e-04,
    l1_ratio = 0.5,
    epochs = 1000L,
    tol = 0.001,
    shuffle = TRUE,
    learning_rate = c("constant", "invscaling", "adaptive"),
    eta0 = 0.001,
    power_t = 0.5,
    batch_size = 32L,
    n_iter_no_change = 5L,
    ...
)

## S3 method for class 'matrix'
cuda_ml_sgd(
  x,
  y,
  fit_intercept = TRUE,
  penalty = c("none", "l1", "l2", "elasticnet"),
  alpha = 1e-04,
  l1_ratio = 0.5,
  epochs = 1000L,
  tol = 0.001,
  shuffle = TRUE,
  learning_rate = c("constant", "invscaling", "adaptive"),
  eta0 = 0.001,
  power_t = 0.5,
  batch_size = 32L,
  n_iter_no_change = 5L,
  ...
)

## S3 method for class 'formula'
cuda_ml_sgd(
  formula,
  data,
  fit_intercept = TRUE,
  penalty = c("none", "l1", "l2", "elasticnet"),
  alpha = 1e-04,
  l1_ratio = 0.5,
  epochs = 1000L,
  tol = 0.001,
  shuffle = TRUE,
  learning_rate = c("constant", "invscaling", "adaptive"),
  eta0 = 0.001,
  power_t = 0.5,
  batch_size = 32L,
  n_iter_no_change = 5L,
```

```

    ...
)

## S3 method for class 'recipe'
cuda_ml_sgd(
  x,
  data,
  fit_intercept = TRUE,
  penalty = c("none", "l1", "l2", "elasticnet"),
  alpha = 1e-04,
  l1_ratio = 0.5,
  epochs = 1000L,
  tol = 0.001,
  shuffle = TRUE,
  learning_rate = c("constant", "invscaling", "adaptive"),
  eta0 = 0.001,
  power_t = 0.5,
  batch_size = 32L,
  n_iter_no_change = 5L,
  ...
)

```

Arguments

x	Depending on the context: • A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.
fit_intercept	If TRUE, then the model tries to correct for the global mean of the response variable. If FALSE, then the model expects data to be centered. Default: TRUE.
penalty	Type of regularization to perform, must be one of {"none", "l1", "l2", "elasticnet"}. • "none": no regularization. • "l1": perform regularization based on the L1-norm (LASSO) which tries to minimize the sum of the absolute values of the coefficients. • "l2": perform regularization based on the L2 norm (Ridge) which tries to minimize the sum of the square of the coefficients. • "elasticnet": perform the Elastic Net regularization which is based on the weighted average of L1 and L2 norms. Default: "none".
alpha	Multiplier of the penalty term. Default: 1e-4.

l1_ratio	The ElasticNet mixing parameter, with $0 \leq \text{l1\_ratio} \leq 1$. For $\text{l1\_ratio} = 0$ the penalty is an L2 penalty. For $\text{l1\_ratio} = 1$ it is an L1 penalty. For $0 < \text{l1\_ratio} < 1$, the penalty is a combination of L1 and L2. The penalty term is computed using the following formula: $\text{penalty} = \alpha * \text{l1\_ratio} * \ \text{w}\ _1 + 0.5 * \alpha * (1 - \text{l1\_ratio}) * \ \text{w}\ _2^2$ where $\ \text{w}\ _1$ is the L1 norm of the coefficients, and $\ \text{w}\ _2$ is the L2 norm of the coefficients.
epochs	The number of times the model should iterate through the entire dataset during training. Default: 1000L.
tol	Threshold for stopping training. Training will stop if $(\text{loss in current epoch}) > (\text{loss in previous epoch}) - \text{tol}$. Default: $1e-3$.
shuffle	Whether to shuffle the training data after each epoch. Default: TRUE.
learning_rate	Must be one of {"constant", "invscaling", "adaptive"}. "constant": the learning rate will be kept constant. "invscaling": $(\text{learning rate}) = (\text{initial learning rate}) / \text{pow}(t, \text{power\_t})$ where t is the number of epochs and power_t is a tunable parameter of this model. "adaptive": $(\text{learning rate}) = (\text{initial learning rate})$ as long as the training loss keeps decreasing. Each time the last n_iter_no_change consecutive epochs fail to decrease the training loss by tol, the current learning rate is divided by 5. Default: "constant".
eta0	The initial learning rate. Default: $1e-3$.
power_t	The exponent used for calculating the invscaling learning rate. Default: 0.5.
batch_size	The number of samples that will be included in each batch. Default: 32L.
n_iter_no_change	The maximum number of epochs to train if there is no improvement in the model. Default: 5.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

A linear model that can be used with the 'predict' S3 generic to make predictions on new data points.

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  model <- cuda_ml_sgd(
    mpg ~ ., mtcars,
    batch_size = 4, epochs = 50000,
    learning_rate = "adaptive", eta0 = 1e-5,
    penalty = "l2", alpha = 1e-5, tol = 1e-6,
```

```

    n_iter_no_change = 10
  )

  predictors <- subset(mtcars, select = -mpg)
  preds <- predict(model, predictors)
  print(all.equal(preds$.pred, mtcars$mpg, tolerance = 0.09))
}

```

 cuda_ml_svm

Train a SVM model.

Description

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

Usage

```

cuda_ml_svm(x, ...)

## Default S3 method:
cuda_ml_svm(x, ...)

## S3 method for class 'data.frame'
cuda_ml_svm(
  x,
  y,
  cost = 1,
  kernel = c("rbf", "tanh", "polynomial", "linear"),
  gamma = NULL,
  coef0 = 0,
  degree = 3L,
  tol = 0.001,
  max_iter = NULL,
  nochange_steps = 1000L,
  cache_size = 1024,
  epsilon = 0.1,
  sample_weights = NULL,
  ...
)

## S3 method for class 'matrix'
cuda_ml_svm(
  x,
  y,
  cost = 1,
  kernel = c("rbf", "tanh", "polynomial", "linear"),
  gamma = NULL,
  coef0 = 0,

```

```

    degree = 3L,
    tol = 0.001,
    max_iter = NULL,
    nochange_steps = 1000L,
    cache_size = 1024,
    epsilon = 0.1,
    sample_weights = NULL,
    ...
)

## S3 method for class 'formula'
cuda_ml_svm(
  formula,
  data,
  cost = 1,
  kernel = c("rbf", "tanh", "polynomial", "linear"),
  gamma = NULL,
  coef0 = 0,
  degree = 3L,
  tol = 0.001,
  max_iter = NULL,
  nochange_steps = 1000L,
  cache_size = 1024,
  epsilon = 0.1,
  sample_weights = NULL,
  ...
)

## S3 method for class 'recipe'
cuda_ml_svm(
  x,
  data,
  cost = 1,
  kernel = c("rbf", "tanh", "polynomial", "linear"),
  gamma = NULL,
  coef0 = 0,
  degree = 3L,
  tol = 0.001,
  max_iter = NULL,
  nochange_steps = 1000L,
  cache_size = 1024,
  epsilon = 0.1,
  sample_weights = NULL,
  ...
)

```

Arguments

x Depending on the context:

	• A data frame of predictors. • A matrix of predictors. • A recipe specifying a set of preprocessing steps created from <code>recipes::recipe()</code>. • A formula specifying the predictors and the outcome.
...	Optional arguments; currently unused.
y	A numeric vector (for regression) or factor (for classification) of desired responses.
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: $1 / (\text{num features})$. The following kernels are implemented: • RBF $K(x_1, x_2) = \exp(-\gamma \ x_1 - x_2\ ^2)$ • TANH $K(x_1, x_2) = \tanh(\gamma \langle x_1, x_2 \rangle + \text{coef0})$ • POLYNOMIAL $K(x_1, x_2) = (\gamma \langle x_1, x_2 \rangle + \text{coef0})^{\text{degree}}$ • LINEAR $K(x_1, x_2) = \langle x_1, x_2 \rangle$, where $\langle, \rangle$ 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 $K(x_1, x_2) = \tanh(\gamma \langle x_1, x_2 \rangle + \text{coef0})$ • POLYNOMIAL $K(x_1, x_2) = (\gamma \langle x_1, x_2 \rangle + \text{coef0})^{\text{degree}}$ • LINEAR $K(x_1, x_2) = \langle x_1, x_2 \rangle$, where $\langle, \rangle$ 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 $K(x_1, x_2) = \tanh(\gamma \langle x_1, x_2 \rangle + \text{coef0})$ • POLYNOMIAL $K(x_1, x_2) = (\gamma \langle x_1, x_2 \rangle + \text{coef0})^{\text{degree}}$ • LINEAR $K(x_1, x_2) = \langle x_1, x_2 \rangle$, where $\langle, \rangle$ denotes the dot product.
tol	Tolerance to stop fitting. Default: $1e-3$.
max_iter	Maximum number of outer iterations in SmoSolver. Default: $100 * (\text{num samples})$.
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	Epsilon 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.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a recipe or formula is used, data is specified as a data frame containing the predictors and (if applicable) the outcome.

Value

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

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  # Classification

  two_class <- modeldata::two_class_dat

  model <- cuda_ml_svm(
    formula = Class ~ .,
    data = two_class,
    kernel = "rbf"
  )

  predictors <- subset(two_class, select = -Class)
  predictions <- predict(model, predictors)

  # Regression

  model <- cuda_ml_svm(
    formula = mpg ~ .,
    data = mtcars,
    kernel = "rbf"
  )

  predictions <- predict(model, mtcars)
}
```

cuda_ml_transform

Transform data with a dimensionality-reduction model

Description

These generics apply a fitted dimensionality-reduction mapping. They are distinct from `predict()`, which produces outcomes from supervised models and returns tidymodels-style prediction columns.

Usage

```
cuda_ml_transform(model, x, ...)

cuda_ml_inverse_transform(model, x, ...)
```

Arguments

model	A model object.
x	The dataset to be transformed.
...	Additional model-specific parameters (if any).

Value

cuda_ml_transform() returns coordinates in the learned representation. cuda_ml_inverse_transform() returns reconstructed predictors in the original feature space.

Supported methods

- cuda_ml_transform() maps predictors into a learned lower-dimensional representation. It supports fitted TSVD and UMAP models.
- cuda_ml_inverse_transform() maps component coordinates back toward the original feature space. It supports fitted PCA and TSVD models.

PCA stores the transformed training input when transform_input = TRUE, but it does not currently provide a method for transforming new data.

See Also

[predict](#), [cuda_ml_pca](#), [cuda_ml_tsvd](#), and [cuda_ml_umap](#)

cuda_ml_tsne

Perform t-distributed stochastic neighbor embedding.

Description

t-distributed stochastic neighbor embedding (t-SNE) for visualizing high- dimensional data.

Usage

```
cuda_ml_tsne(
  x,
  n_components = 2L,
  n_neighbors = ceiling(3 * perplexity),
  method = c("barnes_hut", "fft", "exact"),
  angle = 0.5,
  n_iter = 1000L,
  learning_rate = 200,
```

```

    learning_rate_method = c("adaptive", "none"),
    perplexity = 30,
    perplexity_max_iter = 100L,
    perplexity_tol = 1e-05,
    early_exaggeration = 12,
    late_exaggeration = 1,
    exaggeration_iter = 250L,
    min_grad_norm = 1e-07,
    pre_momentum = 0.5,
    post_momentum = 0.8,
    square_distances = TRUE,
    seed = NULL
)

```

Arguments

x	The input matrix or data frame. Each data point should be a row and should consist of numeric values only.
n_components	Dimension of the embedded space.
n_neighbors	The number of datapoints to use in the attractive forces. Default: ceiling(3 * perplexity).
method	T-SNE method, must be one of {"barnes_hut", "fft", "exact"}. The "exact" method will be more accurate but slower. Both "barnes_hut" and "fft" methods are fast approximations.
angle	Valid values are between 0.0 and 1.0, which trade off speed and accuracy, respectively. Generally, these values are set between 0.2 and 0.8. (Barnes-Hut only.)
n_iter	Maximum number of iterations for the optimization. Should be at least 250. Default: 1000L.
learning_rate	Learning rate of the t-SNE algorithm, usually between (10, 1000). If the learning rate is too high, then t-SNE result could look like a cloud / ball of points.
learning_rate_method	Must be one of {"adaptive", "none"}. If "adaptive", then learning rate, early exaggeration, and perplexity are automatically tuned based on input size. Default: "adaptive".
perplexity	The target value of the conditional distribution's perplexity (see https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding for details).
perplexity_max_iter	The number of epochs the best Gaussian bands are found for. Default: 100L.
perplexity_tol	Stop optimizing the Gaussian bands when the conditional distribution's perplexity is within this desired tolerance compared to its target value. Default: 1e-5.
early_exaggeration	Controls the space between clusters. Not critical to tune this. Default: 12.0.
late_exaggeration	Controls the space between clusters. It may be beneficial to increase this slightly to improve cluster separation. This will be applied after exaggeration_iter iterations (FFT only).

exaggeration_iter	Number of exaggeration iterations. Default: 250L.
min_grad_norm	If the gradient norm is below this threshold, the optimization will be stopped. Default: 1e-7.
pre_momentum	During the exaggeration iteration, more forcefully apply gradients. Default: 0.5.
post_momentum	During the late phases, less forcefully apply gradients. Default: 0.8.
square_distances	Whether TSNE should square the distance values.
seed	Seed to the pseudorandom number generator. Setting this can make repeated runs look more similar. Note, however, that this highly parallelized t-SNE implementation is not completely deterministic between runs, even with the same seed being used for each run. Default: NULL.

Value

A matrix containing the embedding of the input data in a low- dimensional space, with each row representing an embedded data point.

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  oils <- modeldata::oils
  oil_predictors <- oils |>
    subset(select = -class) |>
    scale()

  embedding <- cuda_ml_tsne(oil_predictors, method = "exact")

  set.seed(0)
  print(kmeans(embedding, centers = 7))
}
```

cuda_ml_tsvd	<i>Truncated SVD.</i>
--------------	-----------------------

Description

Dimensionality reduction using Truncated Singular Value Decomposition.

Usage

```
cuda_ml_tsvd(
  x,
  n_components = 2L,
  eig_algo = c("dq", "jacobi"),
```

```

    tol = 1e-07,
    n_iters = 15L,
    transform_input = TRUE
  )

```

Arguments

<code>x</code>	The input matrix or data frame. Each data point should be a row and should consist of numeric values only.
<code>n_components</code>	Desired dimensionality of output data. Must be strictly less than <code>ncol(x)</code> (i.e., the number of features in input data). Default: 2.
<code>eig_algo</code>	Eigen decomposition algorithm to be applied to the covariance matrix. Valid choices are "dq" (divid-and-conquer method for symmetric matrices) and "jacobi" (the Jacobi method for symmetric matrices). Default: "dq".
<code>tol</code>	Tolerance for singular values computed by the Jacobi method. Default: 1e-7.
<code>n_iters</code>	Maximum number of iterations for the Jacobi method. Default: 15.
<code>transform_input</code>	If TRUE, then compute an approximate representation of the input data. Default: TRUE.

Value

A TSVD model object with the following attributes:

- "components": a matrix of `n_components` rows to be used for dimensionality reduction on new data points.
- "explained_variance": (only present if "transform_input" is set to TRUE) amount of variance within the input data explained by each component.
- "explained_variance_ratio": (only present if "transform_input" is set to TRUE) fraction of variance within the input data explained by each component.
- "singular_values": The singular values corresponding to each component. The singular values are equal to the 2-norms of the `n_components` variables in the lower-dimensional space.
- "tsvd_params": opaque pointer to TSVD parameters which will be used for performing inverse transforms.

Examples

```

library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  oils <- modeldata::oils
  oil_predictors <- oils |>
    subset(select = -class) |>
    scale()

  oil_tsvd <- cuda_ml_tsvd(oil_predictors, n_components = 2)
  print(oil_tsvd)
}

```

cuda_ml_umap	<i>Uniform Manifold Approximation and Projection (UMAP) for dimension reduction.</i>
--------------	--

Description

Run the Uniform Manifold Approximation and Projection (UMAP) algorithm to find a low dimensional embedding of the input data that approximates an underlying manifold.

Usage

```
cuda_ml_umap(
    x,
    y = NULL,
    n_components = 2L,
    n_neighbors = 15L,
    n_epochs = 500L,
    learning_rate = 1,
    init = c("spectral", "random"),
    min_dist = 0.1,
    spread = 1,
    set_op_mix_ratio = 1,
    local_connectivity = 1L,
    repulsion_strength = 1,
    negative_sample_rate = 5L,
    transform_queue_size = 4,
    a = NULL,
    b = NULL,
    target_n_neighbors = n_neighbors,
    target_metric = c("categorical", "euclidean"),
    target_weight = 0.5,
    transform_input = TRUE,
    seed = NULL
)
```

Arguments

x	The input matrix or data frame. Each data point should be a row and should consist of numeric values only.
y	An optional numeric vector of target values for supervised dimension reduction. Default: NULL.
n_components	The dimension of the space to embed into. Default: 2.
n_neighbors	The size of local neighborhood (in terms of number of neighboring sample points) used for manifold approximation. Default: 15.
n_epochs	The number of training epochs to be used in optimizing the low dimensional embedding. Default: 500.

learning_rate	The initial learning rate for the embedding optimization. Default: 1.0.
init	Initialization mode of the low dimensional embedding. Must be one of {"spectral", "random"}. Default: "spectral".
min_dist	The effective minimum distance between embedded points. Default: 0.1.
spread	The effective scale of embedded points. In combination with min_dist this determines how clustered/clumped the embedded points are. Default: 1.0.
set_op_mix_ratio	Interpolate between (fuzzy) union and intersection as the set operation used to combine local fuzzy simplicial sets to obtain a global fuzzy simplicial sets. Both fuzzy set operations use the product t-norm. The value of this parameter should be between 0.0 and 1.0; a value of 1.0 will use a pure fuzzy union, while 0.0 will use a pure fuzzy intersection. Default: 1.0.
local_connectivity	The local connectivity required – i.e. the number of nearest neighbors that should be assumed to be connected at a local level. Default: 1.
repulsion_strength	Weighting applied to negative samples in low dimensional embedding optimization. Values higher than one will result in greater weight being given to negative samples. Default: 1.0.
negative_sample_rate	The number of negative samples to select per positive sample in the optimization process. Default: 5.
transform_queue_size	For transform operations (embedding new points using a trained model), this controls how aggressively to search for nearest neighbors. Default: 4.0.
a, b	More specific parameters controlling the embedding. If not set, then these values are set automatically as determined by min_dist and spread. Default: NULL.
target_n_neighbors	The number of nearest neighbors to use to construct the target simplicial set. Default: n_neighbors.
target_metric	The metric for measuring distance between the actual and the target values (y) if using supervised dimension reduction. Must be one of {"categorical", "euclidean"}. Default: "categorical".
target_weight	Weighting factor between data topology and target topology. A value of 0.0 weights entirely on data, a value of 1.0 weights entirely on target. The default of 0.5 balances the weighting equally between data and target.
transform_input	If TRUE, then compute an approximate representation of the input data. Default: TRUE.
seed	Optional seed for pseudo random number generator. Default: NULL. Setting a PRNG seed will enable consistency of trained embeddings, allowing for reproducible results to 3 digits of precision, but at the expense of potentially slower training and increased memory usage. If the PRNG seed is not set, then the trained embeddings will not be deterministic.

Value

A UMAP model object that can be used as input to the `cuda_ml_transform()` function. If `transform_input` is set to `TRUE`, then the model object will contain a `"transformed_data"` attribute containing the lower dimensional embedding of the input data.

Examples

```
library(cuda.ml)

if (interactive() && cuda_ml_backend_info()$runtime_installed) {
  oils <- modeldata::oils
  oil_predictors <- oils |>
    subset(select = -class) |>
    scale()

  model <- cuda_ml_umap(
    x = oil_predictors,
    y = oils$class,
    n_components = 2,
    n_epochs = 200,
    transform_input = TRUE
  )

  set.seed(0)
  print(kmeans(model$transformed_data, iter.max = 100, centers = 7))
}
```

`predict.cuda_ml_knn` *Make predictions on new data points.*

Description

Make predictions on new data points using a cuML KNN model.

Usage

```
## S3 method for class 'cuda_ml_knn'
predict(object, new_data, type = NULL, ...)
```

Arguments

<code>object</code>	A trained CuML model.
<code>new_data</code>	A matrix or data frame containing new data points.
<code>type</code>	Type of prediction. Classification models support <code>"class"</code> and <code>"prob"</code> ; regression models support <code>"numeric"</code> . The default is <code>"class"</code> for classification and <code>"numeric"</code> for regression.
<code>...</code>	Additional arguments to <code>predict()</code> . Currently unused.

Value

Predictions on new data points.

```
predict.cuda_ml_linear_model
```

Make predictions on new data points.

Description

Make predictions on new data points using a linear model.

Usage

```
## S3 method for class 'cuda_ml_linear_model'
predict(object, new_data, ...)
```

Arguments

object	A trained CuML model.
new_data	A matrix or data frame containing new data points.
...	Additional arguments to predict(). Currently unused.

Value

Predictions on new data points.

```
predict.cuda_ml_logistic_reg
```

Predict from a logistic or multinomial regression model

Description

Predict from a logistic or multinomial regression model

Usage

```
## S3 method for class 'cuda_ml_logistic_reg'
predict(object, new_data, type = c("class", "prob"), ...)
```

Arguments

object	A fitted cuda_ml_logistic_reg model.
new_data	New predictor data.
type	Either "class" or "prob".
...	Unused.

```
predict.cuda_ml_nvforest
```

Predict with an nvForest model

Description

Predict with an nvForest model

Usage

```
## S3 method for class 'cuda_ml_nvforest'
predict(
  object,
  new_data,
  type = NULL,
  threshold = NULL,
  chunk_size = NULL,
  ...
)
```

Arguments

<code>object</code>	An nvForest-backed model.
<code>new_data</code>	Numeric predictor data.
<code>type</code>	Classification models support "class" and "prob"; regression models support "numeric". Probability prediction is available only when <code>cuda_ml_nvforest_info(object)\$has_prob</code> is true. Unsupported Treelite postprocessors fail explicitly.
<code>threshold</code>	Binary classification threshold, or NULL for 0.5.
<code>chunk_size</code>	Native prediction chunk size, or NULL for the model default. It controls native batching and does not limit the size of the returned R object.
<code>...</code>	Unused.

Value

A tibble with `.pred` for regression, `.pred_class` for class prediction, or one probability column named `.pred_<level>` for each class.

predict.cuda_ml_svm	<i>Make predictions on new data points.</i>
---------------------	---

Description

Make predictions on new data points using a cuML SVM model.

Usage

```
## S3 method for class 'cuda_ml_svm'  
predict(object, new_data, ...)
```

Arguments

object	A trained CuML model.
new_data	A matrix or data frame containing new data points.
...	Additional arguments to predict(). Currently unused.

Value

Predictions on new data points.

Index

bundle, [40](#)
bundle.cuda_ml_model, [4](#)
bundle.cuda_ml_nvforest
 (bundle.cuda_ml_model), [4](#)

cuda.ml (cuda.ml-package), [3](#)
cuda.ml-package, [3](#)
cuda_ml_agglomerative_clustering, [5](#)
cuda_ml_backend_info, [6](#), [38](#)
cuda_ml_cache_clean, [7](#)
cuda_ml_dbscan, [7](#)
cuda_ml_elastic_net, [8](#)
cuda_ml_install, [3](#), [11](#), [26](#), [28](#), [36](#)
cuda_ml_inverse_transform
 (cuda_ml_transform), [47](#)
cuda_ml_kmeans, [13](#)
cuda_ml_knn, [14](#), [18](#)
cuda_ml_knn_algo, [17](#)
cuda_ml_knn_algo_ivfflat
 (cuda_ml_knn_algo), [17](#)
cuda_ml_knn_algo_ivfpq
 (cuda_ml_knn_algo), [17](#)
cuda_ml_lasso, [18](#)
cuda_ml_linear_reg, [21](#)
cuda_ml_logistic_reg, [22](#)
cuda_ml_nvforest_export, [3](#), [24](#), [29](#), [36](#)
cuda_ml_nvforest_import, [3](#), [29](#), [36](#)
cuda_ml_nvforest_import
 (cuda_ml_nvforest_export), [24](#)
cuda_ml_nvforest_info, [26](#), [29](#)
cuda_ml_nvforest_leaf_ids, [27](#), [29](#)
cuda_ml_nvforest_load_model, [26](#), [27](#)
cuda_ml_nvforest_predict_per_tree, [29](#),
 [29](#)
cuda_ml_ols, [30](#)
cuda_ml_pca, [31](#), [48](#)
cuda_ml_rand_forest, [3](#), [33](#)
cuda_ml_ridge, [36](#)
cuda_ml_runtime_audit, [7](#), [38](#)
cuda_ml_serialize, [3](#), [5](#), [26](#), [29](#), [36](#), [39](#)

cuda_ml_sgd, [40](#)
cuda_ml_svm, [44](#)
cuda_ml_transform, [47](#)
cuda_ml_tsne, [48](#)
cuda_ml_tsvd, [48](#), [50](#)
cuda_ml_umap, [48](#), [52](#)
cuda_ml_unserialize, [5](#), [26](#), [29](#)
cuda_ml_unserialize
 (cuda_ml_serialize), [39](#)

predict, [48](#)
predict.cuda_ml_knn, [54](#)
predict.cuda_ml_linear_model, [55](#)
predict.cuda_ml_logistic_reg, [55](#)
predict.cuda_ml_nvforest, [56](#)
predict.cuda_ml_svm, [57](#)

recipes::recipe(), [10](#), [16](#), [20](#), [23](#), [30](#), [35](#),
 [37](#), [42](#), [46](#)

serialize, [40](#)

unserialize, [40](#)