

Package: faissR (via r-universe)

September 22, 2026

Type Package

Title FAISS-Backed Nearest Neighbours, kNN Models, and k-Means

Version 0.99.47

Description Native nearest-neighbour search, k-nearest neighbour prediction, and k-means helpers for large dimensionality reduction workflows in high-throughput biological data analysis, including single-cell, flow cytometry, imaging, and mass spectrometry applications. Functional native builds require the FAISS C++ library, including CPU-only indexes, and can optionally use FAISS GPU indexes with NVIDIA cuVS integration or direct RAPIDS cuVS/CUDA indexes on NVIDIA machines. CPU-only systems do not need NVIDIA libraries, but NVIDIA CUDA/cuVS libraries are mandatory when a GPU-enabled build is explicitly requested. Unsupported WebAssembly and Bioconductor staging platforms may build diagnostic-only stubs that report the unavailable native functionality. Supervised kNN models use `knn(Xtrain, Ytrain)` and `predict()`, or `knn(Xtrain, Ytrain, Xtest)` for immediate prediction. Class probabilities are returned through `predict(type = ``prob")` for kNN classification models.

License MIT + file LICENSE

URL <https://github.com/tkcaccia/faissR>

BugReports <https://github.com/tkcaccia/faissR/issues>

biocViews Software, Infrastructure, GPU, SingleCell, FlowCytometry, MassSpectrometry, ImagingMassSpectrometry, Proteomics, Clustering, DimensionReduction, Classification, FunctionalPrediction

Encoding UTF-8

Depends R (>= 4.5.0)

Imports Biobase, methods, Rcpp (>= 1.1.0)

Suggests BiocNeighbors, BiocSingular, BiocStyle, S4Vectors, SingleCellExperiment, SummarizedExperiment, knitr, Matrix,

float, rmarkdown, scater, scRNAseq, scan, scuttle, testthat,
withr

LinkingTo Rcpp (>= 1.1.0)

SystemRequirements C++20, Fortran on Unix, FAISS C++ development library (Debian/Ubuntu: libfaiss-dev, libblas-dev, liblapack-dev; RPM distributions: faiss-devel, blas-devel, lapack-devel where available; macOS: static FAISS and OpenMP supplied through the R-macos recipes/toolchain or a compatible local prefix; Windows: an Rtools-compatible FAISS CPU build), complete LP64 BLAS and LAPACK with single-precision routines on Linux and Windows. Optional CUDA and cuVS libraries are used only when a GPU-enabled build is explicitly requested.

VignetteBuilder knitr

RoxygenNote 8.1.0

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/pak/sysreqs nvidia-cuda-dev libfaiss-dev

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

Date/Publication 2026-09-22 15:08:10 UTC

RemoteUrl <https://github.com/BiocStaging/faissR>

RemoteRef HEAD

RemoteSha 82ed27005c5292b15f3c8499ac61047c37240c5b

Contents

backend_info	3
candidate_knn	3
cuda_available	5
cuvs_available	5
faiss_available	6
faiss_gpu_available	6
faissR-c-api	7
faissR_backend	8
fast_kmeans	8
gpu_knn_to_host	10
knn	11
nn	13
nn_capabilities	19
nn_gpu	20
nn_metric_preflight	22
predict.faissR_knn_model	23
print.faissR_knn_model	25
summary.faissR	25
zeisel_brain_pca	26

backend_info	<i>Summarize native neighbour-search backend availability</i>
--------------	---

Description

‘backend_info()’ reports which ‘faissR’ nearest-neighbour backends can currently run. It never silently falls back from an explicit GPU request to CPU; this table is informational only.

Usage

```
backend_info()
```

Value

A data frame with one row per compiled/runtime backend family and columns describing availability, public call hints, public backend names, supported public method/metric summaries, non-public implementation route labels, device/runtime hints, and a short note. Use [nn_capabilities\(\)](#) for the full method/backend/metric matrix.

Hardware visibility and package capability

A visible accelerator or installed driver does not establish that ‘faissR’ was compiled with a compatible provider. ‘backend_info()’ reports package-specific compiled and runtime capability; `[cuda_available()]` is likewise specific to the current ‘faissR’ installation. Framework-neutral hardware discovery is a separate concern, as implemented by the development ‘gpuinfo’ project at <https://github.com/tkcaccia/gpuinfo>. A hardware report from that project complements, but does not replace, these package-specific checks.

Examples

```
info <- backend_info()
info[, c("backend", "available", "public_backends")]
```

candidate_knn	<i>Select nearest neighbours from a candidate matrix</i>
---------------	--

Description

‘candidate_knn()’ computes exact top-k neighbours restricted to a supplied per-query candidate matrix. It is useful after an approximate candidate generation step, for NN-descent refinement, graph refinement, or landmark projection. The function does not generate candidates; it only scores and ranks the candidates that you pass in.

Usage

```
candidate_knn(
  data,
  candidates,
  points = data,
  k,
  backend = NULL,
  metric = c("euclidean", "cosine", "correlation"),
  n_threads = NULL,
  exclude_self = FALSE
)
```

Arguments

<code>data</code>	Dense numeric reference matrix with observations in rows. Sparse, delayed, file-backed, and other matrix-like objects are rejected before conversion.
<code>candidates</code>	Integer matrix of 1-based candidate reference row indices. It must have one row per query. Invalid, missing, zero, or out-of-range entries are ignored.
<code>points</code>	Dense numeric query matrix with observations in rows. Defaults to <code>'data'</code> , i.e. self-query candidate KNN.
<code>k</code>	Number of neighbours to return from each candidate row.
<code>backend</code>	<code>"cpu"</code> for the general CPU implementation or <code>"cuda"</code> for the native CUDA row-candidate kernel. <code>'NULL'</code> follows the package backend configuration. GPU backends currently require self-query candidates with <code>'exclude_self = TRUE'</code> and <code>'k <= 256'</code> .
<code>metric</code>	<code>"euclidean"</code> , <code>"cosine"</code> , or <code>"correlation"</code> . Legacy metric aliases such as <code>"l2"</code> , <code>"cor"</code> , <code>"pearson"</code> , and <code>"ip"</code> are rejected. Correlation is centered cosine similarity. CUDA candidate scoring supports Euclidean directly and cosine/correlation through normalized Euclidean scoring. CUDA rejects zero-vector cosine and constant-row correlation inputs; inspect affected rows with <code>[nn_metric_preflight()]</code> .
<code>n_threads</code>	CPU threads for the CPU backend.
<code>exclude_self</code>	If <code>'TRUE'</code> , remove each query row from its own candidate set. This is valid only for self-query candidate KNN.

Value

A `'faissR_nn'` object with `'indices'` and `'distances'`. If a row has fewer than `'k'` valid unique candidates, remaining entries are `'NA'` and `'Inf'` on both CPU and CUDA. No neighbors outside the supplied candidate set are added. Distance-contract metadata states whether the values are a metric and comparable across query rows.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
rough <- nn(x, k = 10, backend = "cpu")
refined <- candidate_knn(x, rough$indices, k = 5, exclude_self = TRUE)
refined
```

cuda_available	<i>Check whether the native CUDA backend is available</i>
----------------	---

Description

Check whether the native CUDA backend is available

Usage

```
cuda_available()
```

Value

‘TRUE’ when the package was built with CUDA support and the CUDA runtime reports at least one available device.

Examples

```
cuda_available()
```

cuvs_available	<i>Check whether the RAPIDS cuVS backend is available</i>
----------------	---

Description

Check whether the RAPIDS cuVS backend is available

Usage

```
cuvs_available()
```

Value

‘TRUE’ when faissR was compiled and linked against RAPIDS cuVS and the CUDA runtime reports at least one available device.

Examples

```
cuvs_available()
```

faiss_available	<i>Check whether the real FAISS C++ backend is available</i>
-----------------	--

Description

Check whether the real FAISS C++ backend is available

Usage

```
faiss_available()
```

Value

‘TRUE’ when faissR was compiled and linked against FAISS.

Examples

```
faiss_available()
```

faiss_gpu_available	<i>Check whether FAISS GPU support is available</i>
---------------------	---

Description

Check whether FAISS GPU support is available.

Usage

```
faiss_gpu_available()
```

Value

TRUE when faissR was compiled and linked against a FAISS build that reports GPU support.

Examples

```
faiss_gpu_available()
```

Description

Downstream compiled R packages can retrieve faissR's registered C-callable entry points by including 'faissR_api.h' from the installed 'include' directory. The header contains the function-pointer types and inline lookup helpers; these entry points are not called directly from R.

Value

The header accessor 'faissR_c_api_version()' returns an integer ABI version. The 'faissR_get_*()' accessors return typed C function pointers; the retrieved nearest-neighbour callables return R 'SEXP' objects. Callers must protect returned objects before further R allocations (for example, with 'PROTECT()' or 'Rcpp::Shield<SEXP>()'). Their structure and ownership rules are defined in 'faissR_api.h'. This documentation topic does not create an R object.

ABI version

Call 'faissR_c_api_version()' before retrieving another entry point. The current ABI version is 1. A downstream package should reject an unsupported ABI version instead of assuming that signatures or object ownership remain compatible.

Registered callables

'faissR_get_nn_float32()' retrieves the host-result self-KNN callable. 'faissR_get_nn_float32_output()' additionally accepts a distance-storage selector. 'faissR_get_nn_cuda_tuned_gpu()' retrieves the CUDA-resident self-KNN callable for "auto", "exact", "flat", and "bruteforce". The exact signatures are declared in 'faissR_api.h'.

Self-neighbour polarity and indices

The native API uses 'include_self': 'TRUE' retains each query row in its own self-search result and 'FALSE' removes it. This is the opposite polarity of the R-level 'exclude_self' argument. Host results contain one-based R row identifiers.

GPU ownership

A GPU-resident result owns its device allocations through an external pointer. The owner object must remain reachable while a downstream consumer uses either child device pointer. Consumers must not free those pointers. Convert explicitly with [gpu_knn_to_host()] when host matrices are needed.

faissR_backend	<i>Configure the default faissR execution backend</i>
----------------	---

Description

Explicit function arguments take precedence over 'options(faissR.backend)', then 'FAISSR_BACKEND'; CPU is the final default. The current faissR source supports CPU and CUDA. "metal" is rejected explicitly.

Usage

```
faissR_backend(backend = NULL)
```

Arguments

backend Optional backend: "cpu" or "cuda".

Value

The active backend. Setting returns the previous option invisibly.

Examples

```
old <- getOption("faissR.backend")
faissR_backend("cpu")
faissR_backend()
options(faissR.backend = old)
```

fast_kmeans	<i>Fast k-means clustering</i>
-------------	--------------------------------

Description

Run k-means using the fastest available backend. CPU acceleration uses FAISS when faissR was built with FAISS. CUDA acceleration first tries FAISS GPU, which can use NVIDIA/cuVS-enabled FAISS builds, then direct RAPIDS cuVS when available. Explicit GPU requests fail clearly instead of silently changing to CPU.

Usage

```
fast_kmeans(
  data,
  centers,
  backend = NULL,
  max_iter = "auto",
  n_init = "auto",
```

```

    tol = "auto",
    seed = 1L,
    n_threads = NULL,
    streaming_batch_size = 0L,
    init = c("kmeans++", "random"),
    tuning = c("auto", "fixed", "off", "none")
)

```

Arguments

data	Dense numeric matrix with observations in rows. Sparse, delayed, file-backed, and other matrix-like objects are rejected before conversion.
centers	Number of clusters.
backend	Device backend: "cpu" or "cuda". 'NULL' follows 'options(faissR.backend)', then 'FAISSR_BACKEND', with CPU as the final default.
max_iter	Maximum number of Lloyd iterations, or "auto" for a deterministic shape-aware default computed by the compiled C++ tuning helper.
n_init	Number of random restarts where supported, or "auto" for a deterministic shape-aware default computed by the compiled C++ tuning helper.
tol	Single non-negative finite relative convergence tolerance where supported, or "auto" for a deterministic shape-aware default computed by the compiled C++ tuning helper.
seed	Random seed for CPU/statistics and FAISS paths. The current direct cuVS C API path does not expose an explicit seed in the stable params structure.
n_threads	Number of CPU threads for FAISS/statistics paths.
streaming_batch_size	cuVS host-data streaming batch size. '0' lets cuVS choose its default.
init	Initialization method, "kmeans++" or "random" where supported.
tuning	Tuning policy. "auto" uses deterministic C++ rules based on 'nrow(data)', 'ncol(data)', and 'centers' without running pilot searches. Small many-cluster jobs can use extra restarts when 'n / centers' remains large enough, and cheap many-cluster jobs with few observations per center also use a small multistart budget for stability; large or high-dimensional jobs use cheaper iteration and tolerance defaults. 'centers = 1' uses the exact column-mean solution for every backend request with 'max_iter = 1', 'n_init = 1', and 'tol = 0', records 'single_cluster_exact_mean', and stays on CPU because no iterative k-means backend can improve that solution. 'centers = nrow(data)' uses the exact singleton assignment for every backend request and records 'singleton_exact_identity'; explicit CUDA requests are recorded as resolved by the exact faissR trivial route rather than launching GPU work. "fixed", "off", and "none" use the historical fixed defaults unless 'max_iter', 'n_init', or 'tol' are explicitly supplied.

Value

A list with 'cluster', 'centers', 'withinss', 'tot.withinss', 'size', 'iter', 'converged', 'hit_max_iter', 'backend', and 'parameters'. 'backend' records the implementation that actually ran, while 'parameters\$requested_backend' and 'parameters\$resolved_backend' record the public backend request

and resolved device. ‘parameters\$tuning’ records the deterministic k-means policy, stable ‘rule’ label, shape metadata, and whether ‘max_iter’, ‘n_init’, and ‘tol’ were auto-selected or supplied explicitly. The parameter rules are computed by compiled C++ helpers and record ‘tuning_source = "cpp"’. ‘parameters\$tuning\$rule_detail’ records the exact ‘n’/‘p’/‘centers’/work values used to choose the rule. ‘parameters\$tuning\$effective’ records the final values used after explicit overrides and “auto” defaults have been resolved; ‘parameters\$tuning\$effective_max_iter’, ‘parameters\$tuning\$effective_n_init’, and ‘parameters\$tuning\$effective_tol’ expose the same values as flat fields for benchmark summaries. ‘parameters\$tuning\$selection’ stores the explicit device request, effective-parameter decision, runtime capability flags, and input shape used by the compiled selector. CUDA runs also record ‘parameters\$cuda_provider_selection’ as “faiss_gpu”, “direct_cuvs”, or “direct_cuvs_after_faiss_gpu_unavailable_or_failed”; ‘parameters\$backend_resolution_note’ describes the provider route, and ‘parameters\$faiss_gpu_error’ is present when direct cuVS was used after a FAISS GPU route was unavailable or failed. ‘hit_max_iter’ records whether the run reached the effective iteration cap, and ‘converged’ is the corresponding conservative convergence flag used by benchmark summaries.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
fit <- fast_kmeans(x, centers = 3, backend = "cpu", n_threads = 2)
table(fit$cluster)
```

gpu_knn_to_host

Copy a GPU-resident KNN result to host matrices

Description

‘gpu_knn_to_host()’ is an explicit diagnostic/conversion helper for ‘faissR_gpu_knn’ objects. It copies device ‘indices’ and ‘distances’ into ordinary R matrices. It is never called automatically by ‘nn_gpu()’.

Usage

```
gpu_knn_to_host(x)
```

Arguments

x A ‘faissR_gpu_knn’ object.

Value

A host ‘faissR_nn’ list with integer ‘indices’ and numeric ‘distances’. The distance-contract metadata from the GPU object is retained.

Examples

```
if (cuda_available()) {
  x <- matrix(rnorm(200), ncol = 4)
  gpu_knn <- nn_gpu(x, k = 5, exclude_self = TRUE)
  host_knn <- gpu_knn_to_host(gpu_knn)
  str(host_knn)
}
```

knn

Fit or apply a k-nearest-neighbour classifier or regressor

Description

`knn()` is the high-level supervised kNN API. With `Xtrain` and `Ytrain` only, it returns a reusable model. With `Xtest`, it immediately predicts the query rows by fitting the model and calling `predict()` internally. The low-level nearest-neighbour search API remains `nn()`. For explicit CPU FAISS-backed `method = "flat"`, `"hnsb"`, `"ivf"`, and `"ivfpq"` models, the fitted object stores a session-local NN index and `predict()` reuses it for compatible prediction calls. For IVF/IVFPQ, this reuses trained centroids, inverted lists, and indexed vectors; IVFPQ also reuses trained product-quantizer codebooks and compressed codes.

Usage

```
knn(
  Xtrain,
  Ytrain,
  Xtest = NULL,
  backend = NULL,
  method = NULL,
  metric = NULL,
  tuning = c("auto", "cache", "pilot", "fixed", "off", "none"),
  target_recall = 0.99,
  cagra_implementation = NULL,
  cagra_build_algo = NULL,
  task = c("auto", "classification", "regression"),
  k = 15L,
  n_threads = NULL,
  vote = c("majority", "weighted"),
  type = c("response", "prob"),
  ...
)
```

Arguments

<code>Xtrain</code>	Dense numeric training matrix or optional <code>float::f1()/float32</code> matrix with observations in rows. Float32 inputs are preserved for <code>nn()</code> methods with direct float32 adapters. Sparse, delayed, file-backed, and other matrix-like objects are rejected before conversion.
---------------------	---

<code>Ytrain</code>	Training labels or numeric response.
<code>Xtest</code>	Optional dense numeric or float32 query matrix. If supplied, <code>knn()</code> returns predictions for <code>Xtest</code> ; otherwise it returns a fitted model.
<code>backend</code>	Device backend passed to <code>nn()</code> : "cpu" or "cuda". NULL follows the package backend configuration.
<code>method</code>	Nearest-neighbour algorithm selector passed to <code>nn()</code> . "auto" chooses the most appropriate method for the selected backend. See <code>?nn</code> for descriptions and references for "exact", "flat", "bruteforce", "grid", "hsw", "ivf", "ivfpq", "ivfpq_fastscan", "vamana_style", "nsg_style", "nndescent_style", and "cagra". The package-owned derived graph routes are experimental.
<code>metric</code>	Distance metric passed to <code>nn()</code> : "euclidean", "cosine", or "correlation". Legacy metric aliases are rejected. Correlation is centered cosine similarity. See <code>nn()</code> for metric/backend support details, including metric-aware CPU HNSW routing.
<code>tuning</code>	Tuning policy passed to <code>nn()</code> . "auto" uses the deterministic default for the resolved method; "cache" and "pilot" opt into pilot tuning where implemented. FAISS GPU IVF pilot/cache tuning is Euclidean-only; non-Euclidean IVF routes use deterministic metric-aware defaults.
<code>target_recall</code>	Speed/recall tier passed to <code>nn()</code> . Use 0.9, 0.95, or 0.99. CUDA method = "auto" uses it for Flat-vs-IVF selection, CUDA IVF uses it for probing defaults, and HNSW uses it for graph-search tiers. CUDA HNSW metadata records that the available cuVS route is a CAGRA-to-HNSW wrapper. For approximate methods this is a selection objective, not a guarantee; inspect the underlying NN metadata, including <code>tuning_benchmark_target_met</code> , and validate recall for the application.
<code>cagra_implementation</code>	CUDA CAGRA provider passed to <code>nn()</code> for method = "cagra" or CUDA-auto routes that select CAGRA. NULL uses the global <code>faissR.cagra_implementation</code> option; "auto" uses the same deterministic shape-aware provider rule as <code>nn()</code> , while "faiss_gpu" or "cuvs" force one provider.
<code>cagra_build_algo</code>	Direct RAPIDS cuVS CAGRA graph-build algorithm passed to <code>nn()</code> for direct cuVS CAGRA routes. NULL uses the global <code>faissR.cuvs_cagra_build_algo</code> option.
<code>task</code>	"auto", "classification", or "regression". "auto" treats numeric responses as regression and other response types as classification.
<code>k</code>	Default number of neighbours used by <code>predict()</code> and by immediate predictions when <code>Xtest</code> is supplied.
<code>n_threads</code>	CPU threads passed to <code>nn()</code> for CPU backends.
<code>vote</code>	"majority" or "weighted" for immediate predictions.
<code>type</code>	"response" for class/regression predictions or "prob" for class probability matrices from classification models.
<code>...</code>	Reserved for future options.

Value

If `Xtest` is not supplied, a fitted `faissR_knn_model` object that stores the training data, response, task, backend, method, metric, tuning, `target_recall`, `k`, and CPU thread settings used by later `predict()` calls. Explicit CPU FAISS Flat/HNSW/IVF/IVFPQ models also store a session-local fitted index for compatible prediction calls; saved/reloaded models safely rebuild the same route when the external pointer is no longer valid. IVF metadata records whether trained centroids/inverted lists were reused and whether search used a query-specific `nprobe`; IVFPQ metadata also records product-quantizer codebook/code reuse. If `Xtest` is supplied, a factor for classification, a numeric vector for regression, or a numeric class-probability matrix when `type = "prob"`; prediction outputs carry `attr(result, "faissR_nn")` route metadata, approximation parameters, and auto-selection metadata from the underlying `nn()` call, including `batch_query = TRUE`, `query_n`, and `query_call_count = 1L` for the full-matrix prediction query.

Changing `target_recall` at prediction time bypasses an IVF/IVFPQ index fitted for a different tier and resolves the requested settings afresh. The prediction records `query_source = "nn"` in `attr(result, "faissR_nn")`; the original fitted model is unchanged.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
model <- knn(x, iris$Species, backend = "cpu", k = 5)
head(predict(model, x, k = 5))
head(predict(model, x, k = 5, type = "prob"))

pred <- knn(x, iris$Species, x, backend = "cpu", k = 5)
head(pred)
```

nn

Nearest neighbors from row-wise matrices

Description

`'nn()'` provides a package-native nearest-neighbor entry point compatible with the common `'nn(data, points, k)'` use case. The public API separates device selection from algorithm selection. `'backend'` is either `"cpu"` or `"cuda"`; `'method'` chooses the algorithm. For example, `'backend = "cpu", method = "grid"` uses the CPU grid implementation, while `'backend = "cuda", method = "grid"` uses the CUDA grid implementation. Invalid combinations stop clearly before computation; for example, `'backend = "cpu", method = "cagra"` errors because CAGRA is CUDA-only.

Usage

```
nn(
  data,
  points = data,
  k = NULL,
  exclude_self = FALSE,
  backend = NULL,
  method = c("auto", "exact", "flat", "bruteforce", "grid", "hnsw",
```

```

    "ivf", "ivfpq",
    "vamana_style", "nsg_style", "nndescent_style", "ivfpq_fastscan", "cagra"),
metric = c("euclidean", "cosine", "correlation"),
tuning = c("auto", "cache", "pilot", "fixed", "off", "none"),
target_recall = 0.99,
cagra_implementation = NULL,
cagra_build_algo = NULL,
output = c("double", "float"),
distances = NULL,
n_threads = NULL
)

```

Arguments

data	Numeric matrix/data frame or optional <code>'float::fl()'</code> / <code>'float32'</code> object of reference observations in rows. FAISS CPU/GPU and RAPIDS cuVS nearest-neighbour routes use direct float-pointer adapters for float32 inputs. Native routes without a direct float32 adapter fail clearly instead of silently converting benchmark input back to R double. Sparse, delayed, file-backed, and other matrix-like objects are rejected before conversion to prevent accidental dense materialization; compute a dense low-dimensional representation or explicitly materialize a manageable block first.
points	Numeric matrix/data frame or optional <code>'float::fl()'</code> / <code>'float32'</code> query object with observations in rows. Defaults to <code>'data'</code> . A float32 query can be paired with an ordinary R double reference matrix on direct FAISS/cuVS float32 routes.
k	Number of neighbors to return. <code>'NULL'</code> chooses the package's automatic neighborhood size and includes the self-neighbor when <code>'points'</code> is <code>'data'</code> .
exclude_self	Logical; when <code>'TRUE'</code> , remove each query row from its own neighbour list. This is valid only for self-query calls where <code>'points'</code> is omitted or identical to <code>'data'</code> . Self-neighbour removal is passed to the compiled backend path rather than repaired by R-side post-processing. CUDA graph routes that do not yet expose compiled include-self output shaping require <code>'exclude_self = TRUE'</code> and fail clearly instead of reshaping in R.
backend	Requested execution device: <code>"cpu"</code> or <code>"cuda"</code> . <code>'NULL'</code> follows <code>'options(faissR.backend)'</code> , then <code>'FAISSR_BACKEND'</code> , with CPU as the final default. The historical result field <code>'backend_used'</code> instead names the concrete resolved provider/route and is retained for API compatibility. CUDA requests fail clearly when CUDA support or the selected CUDA combination is unavailable.
method	Algorithm selector. <code>"auto"</code> chooses a shape-aware default for the selected backend. Other values include <code>"exact"</code> , <code>"flat"</code> , <code>"bruteforce"</code> , <code>"grid"</code> , <code>"hnsu"</code> , <code>"ivf"</code> , <code>"ivfpq"</code> , <code>"vamana_style"</code> , <code>"nsg_style"</code> , <code>"nndescent_style"</code> , <code>"ivfpq_fastscan"</code> , and <code>"cagra"</code> . The historical shorter spellings <code>"vamana"</code> , <code>"nsg"</code> , and <code>"nndescent"</code> remain accepted compatibility aliases. Use these public lowercase method labels; resolved implementation labels such as <code>"faiss_hnsu"</code> or <code>"cuda_cuvs_cagra"</code> are not public <code>'method'</code> values. Unsupported backend/method combinations fail clearly; for example, <code>'method = "cagra", backend = "cpu"</code> errors because CAGRA is CUDA-only, and CUDA <code>'method = "ivfpq_fastscan"</code> accepts Euclidean/L2, cosine, and correlation. CPU FastScan accepts the same metrics;

cosine is implemented by row L2 normalization, correlation by row centering plus L2 normalization followed by FastScan L2 search. The preferred names for package-owned graph refinement are `"nsg_style"`, `"vamana_style"`, and `"nn_descent_style"`; the shorter names remain compatibility aliases. The `'_style'` suffix is a scope marker, not an algorithm name: native results explicitly report a distinct package-owned derived graph-refinement algorithm rather than a feature-complete canonical reproduction. CUDA cuVS NN-descent is identified separately as an external-provider implementation. Package-owned routes carry `'implementation_status = "experimental"'` and `'experimental = TRUE'` in their returned metadata. They are available for explicit evaluation but are excluded from the package's principal comparative performance claims.

metric

Distance metric. The intentionally small public set is `"euclidean"`, `"cosine"`, and `"correlation"`. Euclidean results are ordinary L2 distances, not squared L2 values. Legacy aliases such as `"l2"`, `"cor"`, `"pearson"`, and `"ip"` are rejected; use the canonical metric names. `"cosine"` uses row L2 normalization, and `"correlation"` is centered cosine similarity after subtracting each row mean and L2-normalizing each row. `"euclidean"` is the validated high-performance default. `"cosine"` and `"correlation"` are implemented for exact CPU KNN, native 2D/3D grid search, FAISS CPU/GPU Flat, FAISS CPU/GPU IVF-Flat, FAISS CPU/GPU IVFPQ, FAISS CPU FastScan, CUDA cuVS IVFPQ FastScan for cosine and correlation, FAISS CPU HNSW, and native graph-refinement routes. Similarity-capable routes use row L2 normalization for cosine and row centering plus L2 normalization for correlation before search; distances are returned as `'1 - similarity'`. All-zero cosine rows and constant correlation rows are zero-normalized edge cases: two zero-normalized rows have distance `'0'`, while a zero-normalized row versus a nonzero row has distance `'1'`. CPU FAISS Flat uses the exact CPU scorer for those rows to preserve deterministic small-`'k'` tie handling; explicit CUDA routes error clearly instead of repairing those rows on CPU. These finite values are software conventions for otherwise undefined cosine or correlation cases, not mathematical cosine similarities or Pearson correlations. All backends reject rows containing `'NA'`, `'NaN'`, `'Inf'`, or `'-Inf'`. Use `[nn_metric_preflight()]` to obtain affected one-based row indices and the requested backend's action before search. CUDA FAISS/cuVS results carry `'attr(result, "gpu_residency")'` metadata with provider, index residency, host/device transfer strategy, query device reuse, and CPU fallback flags. CPU `'method = "auto"'` can use FAISS Flat for larger exact non-Euclidean query workloads, FAISS HNSW for large non-Euclidean self-search, native CPU NSG/Vamana refinement for selected larger self-KNN cases, and native CPU NN-descent for other large self-KNN cases. CPU `'method = "hnsw"'` uses FAISS HNSW for all three metrics. CUDA HNSW metadata records the available cuVS HNSW wrapper design. Unsupported method/backend/metric combinations fail clearly instead of changing the requested metric, method, or device. Use `'nn_capabilities(runtime = TRUE)'` to preflight both design support and the libraries available in the current installation.

tuning

Tuning policy. `"auto"` uses deterministic compiled defaults selected from backend, method, metric, data shape, `'k'`, and `'target_recall'`; it does not run a timing pilot. `"cache"` reuses or stores pilot results, `"pilot"` tunes for the current call, `"fixed"` uses untuned fixed defaults while retaining metadata, and

	"off"/"none" disables tuning. Result metadata records the selected parameters, their provenance, and 'tuning_benchmark_target_met' when a benchmark-derived row did not attain the requested target. Exact methods record recall 1 by construction; their tuning controls batching, copies, and resource reuse.
target_recall	Requested recall tier: '0.9', '0.95', or '0.99' (default). Approximate methods use it to select a speed/quality policy; 'method = "auto"' may also use it when choosing a method. It is a target, not a guarantee: inspect 'tuning_benchmark_target_met' and measured recall. Other numeric values are rejected; faissR does not round or interpolate between the three calibrated tiers. Exact methods remain exact and use this value only as policy metadata.
cagra_implementation	CUDA CAGRA provider for this call. 'NULL' uses the global 'options(faissR.cagra_implementation = ...)' value. "auto" uses a deterministic provider rule: compact high-dimensional self-KNN selects direct RAPIDS cuVS CAGRA when both providers are available, while FAISS GPU CAGRA remains the default for other shapes. "faiss_gpu" and "cuvs" force one provider for benchmarking. This argument affects only public 'backend = "cuda", method = "cagra" requests and CUDA 'method = "auto" routes that select CAGRA.
cagra_build_algo	Direct RAPIDS cuVS CAGRA graph-build algorithm for this call. 'NULL' uses 'options(faissR.cuvs_cagra_build_algo = "auto")'. For direct cuVS CAGRA, "auto" applies faissR's deterministic shape-aware CAGRA build rule, choosing iterative CAGRA construction for compact high-dimensional self-KNN cases and IVF-PQ construction otherwise. "ivf_pq" requests the IVF-PQ graph builder, "nn_descent" requests cuVS NN-descent graph construction, and "iterative_cagra_search" requests cuVS iterative CAGRA graph building. This is a CAGRA construction parameter, not a fallback to a different public method; successful results record the selected value in 'attr(result, "approximation")\$cagra_build_algo'.
output	Distance storage type for the returned object. "double" returns the default R numeric distance matrix. "float" returns 'distances' as a 'float::fl()/'float32' object and records 'distance_type = "float32"' plus 'attr(result, "distance_type") = "float32"'; this requires the optional 'float' package. When either 'data' or 'points' is a 'float::fl()' matrix, FAISS Flat/IVF/IVFPQ/HNSW/FastScan/NSG/NNDescent, FAISS GPU Flat/IVF/IVFPQ/CAGRA, and RAPIDS cuVS brute-force/CAGRA/IVF/IVFPQ/NN-descent routes consume float32 input directly through C++ adapters. Methods without a direct float32 adapter now error instead of silently converting the benchmark input back to R double. Ordinary R double inputs with 'output = "float"' also enter float-pointer routes for CPU FAISS Flat/IVF/IVFPQ/FastScan, cached CPU FAISS fitted indexes, FAISS GPU Flat/IVF/IVFPQ, and direct Euclidean RAPIDS cuVS brute-force/CAGRA/IVF/IVFPQ and IVFPQ FastScan routes. On direct FAISS/cuVS float routes, float distance output is constructed directly from backend float results instead of first materializing an R double distance matrix, except for routes that need R-side metric transformation or zero-row cosine/correlation correction.
distances	Optional alias for 'output', kept for callers that prefer 'distances = "double"' or 'distances = "float"' to describe the returned distance storage type.
n_threads	Number of CPU worker threads for CPU backends. GPU backends ignore this argument.

Details

The public methods are:

- `"auto"`: deterministic selection from the requested backend, metric, data shape, `'k'`, and target recall. It does not run a timing pilot. The CPU policy is calibration-informed and experimental: it has not received the independent installed-policy validation reported for CUDA.
- `"exact"`: exact search using FAISS Flat or direct cuVS brute force.
- `"flat"`: an explicit FAISS exhaustive Flat index.
- `"bruteforce"`: exhaustive search, using FAISS on CPU and preferring direct cuVS on CUDA when available.
- `"grid"`: exact 2D/3D self-KNN for Euclidean, cosine, or correlation.
- `"hnsw"`: FAISS HNSW on CPU; CUDA uses the cuVS HNSW wrapper that converts a CAGRA seed graph and is not a pure all-GPU HNSW search.
- `"ivf"`: FAISS IVF-Flat approximate search with coarse-list probing.
- `"ivfpq"`: compressed FAISS IVF-PQ approximate search.
- `"ivfpq_fastscan"`: FAISS FastScan on CPU and the package's separately identified cuVS 4-bit IVF-PQ route on CUDA.
- `"vamana_style"` (`"vamana"` compatibility alias): an experimental package-owned robust-pruned candidate graph inspired by DiskANN/Vamana, followed by CPU or CUDA candidate refinement. It is not a feature-complete Vamana reproduction.
- `"nsg_style"` (`"nsg"` compatibility alias): a distinct experimental package-owned candidate-graph algorithm derived from selected NSG/MRNG pruning ideas, followed by CPU or CUDA candidate refinement. It is not a feature-complete NSG reproduction.
- `"ndescent_style"` (`"ndescent"` compatibility alias): a distinct experimental package-owned CPU graph-refinement algorithm derived from NN-descent ideas, or direct external-provider cuVS NN-descent on CUDA.
- `"cagra"`: CUDA-only FAISS GPU CAGRA or direct cuVS CAGRA. Use `'cagra_implementation'` to request a provider explicitly.

Exact-family names are distinct public policies even when a backend resolves more than one of them to the same exhaustive provider. Result metadata records both the requested method and the concrete provider. Unsupported combinations fail instead of silently changing method, metric, or device.

Value

A list with integer matrix `'indices'`, `'distances'`, and stable metadata fields `'index_base'`, `'distance_type'`, `'metric'`, and `'backend_used'`. Every result declares `'distance_is_metric'`, `'distance_semantics'`, `'distance_comparable_across_queries'`, and `'distance_order'`. Float32 routes also record `'input_layout'` and `'input_owns_data'` so downstream packages can distinguish direct float32 payload use from one-time row-major conversion. Normalized Euclidean graph routes for cosine/correlation record `'metric_transform'` and `'attr(result, "distance_transform")'`. Indices are 1-based signed R integers; CUDA result identifiers are signed int32. The public identifier space is therefore limited to `'Machine$integer.max'` reference rows, although provider and memory limits are usually much lower. The requested backend/method, tuning policy, resolved backend, metric, exact/approximate flag,

and self-query flag are stored in attributes including `'attr(result, "requested_backend")'`, `'attr(result, "requested_method")'`, `'attr(result, "tuning")'`, and `'attr(result, "resolved_backend")'`. `'method = "auto"'` requests also include `'attr(result, "auto_selection")'`, a static workload/shape/k/metric decision record. It includes reference rows and variables, query-row count, self-query status, the `'n * n_points * p'` work estimate, and the predicted internal backend, public method class, device class, explicit backend/method flags, backend/method decision reasons, and hardware provenance. CPU `'method = "auto"'` records `'auto_policy_status = "calibration_informed_not_independently_validated"'` and `'auto_policy_evidence_scope = "cpu_static_policy_experimental"'`. CUDA `'method = "auto"'` is a separately evaluated but still experimental L40S-calibrated policy for cold full-self-search and records `'auto_policy_status = "experimental_l40s_calibrated_cold_full_self_search"'`; neither label implies general workload or hardware validation. Expected future query batches are not an input; use a fitted or cached index when construction will be amortized. A capability-compatible machine that differs from the calibration hardware keeps the compiled policy but is labelled `'hardware_extrapolated_unvalidated'`. If either hardware identity cannot be determined, the label is `'hardware_unidentified'`. Hardware identity alone never causes a silent method or device fallback. CUDA method selection emits one warning per confirmed, unmatched runtime GPU model unless `'options(faissR.warn_hardware_extrapolation = FALSE)'` is set. Pilot/cache tuning adjusts parameters within a method and does not install a new cross-method policy. The selector does not run pilot tuning. CPU FAISS Flat/HNSW/IVF/IVFPQ/FastScan routes use a bounded session-local fitted-index cache for repeated raw `'nn()'` calls with matching data and parameters. CUDA `'method = "ivfpq_fastscan"'` also reuses a fitted cuVS IVF-PQ index, dataset device buffer, and cuVS resources through a separate bounded cache. Self-query uses the fitted dataset device buffer directly, and repeated separate-query calls can reuse one cached query device buffer with `'options(faissR.cache_cuda_ivfpq_query_buffers = TRUE)'`. Metadata reports `'persistent_index_cache'`, `'index_cache_hit'`, `'dataset_residency'`, `'query_residency'`, and `'query_host_to_device_copies'`. Disable CPU and CUDA fitted-index caches with `'options(faissR.cache_fitted_nn_indexes = FALSE)'`, bound CPU memory with `'options(faissR.cache_fitted_nn_indexes_max_entries = <n>)'`, or bound CUDA FastScan GPU memory with `'options(faissR.cache_fitted_cuda_ivfpq_indexes_max_entries = <n>)'`. CPU Euclidean, cosine, and correlation `'method = "ivfpq_fastscan"'` resolve `'tuning = "auto"'` in C++ from shape/k/target-recall defaults for `'nlist'`, `'nprobe'`, `'pq_m'`, `'refine_factor'`, and FastScan block size. Cosine uses row L2 normalization and correlation uses row centering plus L2 normalization before FastScan L2. CUDA FastScan cosine and correlation auto policies are seeded from the CUDA Euclidean FastScan table until metric-specific HPC sweeps replace them, and those rows report `'tuning_benchmark_target_met = FALSE'`.

References

- Johnson J, Douze M, Jegou H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*. 2021;7:535-547.
- Douze M, Guzhva A, Deng C, Johnson J, Szilvasy G, Mazaré PE, et al. The FAISS library. *arXiv* 2024. See also the FAISS C++ API documentation.
- RAPIDS Development Team. RAPIDS cuVS: GPU-accelerated vector search and clustering. <https://github.com/NVIDIA/cuvs>.
- Dong W, Moses C, Li K. Efficient k-nearest neighbor graph construction for generic similarity measures. *WWW* 2011:577-586.
- Malkov YA, Yashunin DA. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE TPAMI*. 2020;42:824-836.

Jégou H, Douze M, Schmid C. Product quantization for nearest neighbor search. IEEE TPAMI. 2011;33:117-128.

NVIDIA, Meta, and FAISS documentation for FAISS GPU indexes backed by NVIDIA cuVS, including IVF and CAGRA integration.

Examples

```
x <- scale(as.matrix(iris[, 1:4]))
knn_euclidean <- nn(x, k = 16, metric = "euclidean", backend = "cpu")
knn_cosine <- nn(x, k = 16, metric = "cosine", backend = "cpu")
knn_correlation <- nn(x, k = 16, metric = "correlation", backend = "cpu")

if (faiss_available() && requireNamespace("Biobase", quietly = TRUE)) {
  data("sample.ExpressionSet", package = "Biobase")
  expression_data <- Biobase::exprs(sample.ExpressionSet)
  x_biobase <- scale(t(expression_data[seq_len(32L), , drop = FALSE]))
  expression_knn <- nn(
    x_biobase,
    k = 3,
    backend = "cpu",
    method = "exact",
    exclude_self = TRUE
  )
  dim(expression_knn$indices)
}
```

nn_capabilities

Nearest-neighbour method capabilities

Description

nn_capabilities() returns the public method/backend/metric support table used by the nearest-neighbour API. It separates combinations that are supported by design from combinations that should be treated as expected skips in benchmarks.

Usage

```
nn_capabilities(runtime = FALSE)
```

Arguments

runtime	Logical; when FALSE (the default), report support by design without checking the current compiled/runtime libraries. When TRUE, add resolved_backend, runtime_available, runtime_reason, and runtime_notes columns for the current installation. runtime_reason uses stable labels such as "available", "unsupported_combination", "missing_faiss", "missing_faiss_gpu", "missing_cuda", "missing_cuda_route", and "missing_cuvs" for benchmark preflight tables.
---------	---

Details

faissR exposes three public nearest-neighbour metrics: "euclidean", "cosine", and "correlation". Correlation is centered cosine similarity.

The capability table reports metric support only when the selected route computes neighbours under that metric. FAISS Flat, IVF-Flat, IVFPQ, and HNSW support cosine and correlation by using normalized dot-product searches. CAGRA and NNDescent support cosine and correlation by using normalized Euclidean graph search. Public CUDA HNSW uses RAPIDS cuVS HNSW from a CAGRA seed graph with a cuVS CPU hierarchy and records that wrapper design in metadata rather than presenting it as a pure all-GPU HNSW implementation. Public CPU NSG uses faissR's distinct native NSG/MRNG-derived route for all metrics because linked FAISS graph builders can abort during graph construction; large high-dimensional CPU inputs use deterministic FAISS HNSW seeding before the derived pruning pass. Public CUDA method = "cagra" can resolve to FAISS GPU CAGRA or direct cuVS CAGRA; options(faissR.cagra_implementation = "faiss_gpu") or "cuvs" forces one provider, while "auto" uses a deterministic shape rule: direct cuVS CAGRA is selected for compact high-dimensional self-KNN, and FAISS GPU CAGRA remains the default when both providers are available for other shapes. Availability preflights respect the forced provider for all supported CAGRA metrics, and returned approximate NN objects record cagra_provider plus cagra_provider_option.

Use runtime = TRUE for benchmark preflight checks that must distinguish a valid method/metric combination from a route that is unavailable in the current build, such as FAISS GPU rows on a CPU-only installation.

Value

A data frame with one row per public method, backend ("cpu" or "cuda"), and metric combination. Columns include supported, exact, implementation, implementation_status, and notes. Package-owned *_style routes are marked "experimental"; external-provider NN-descent is identified separately. If runtime = TRUE, runtime availability columns are appended.

Examples

```
caps <- nn_capabilities()
subset(caps, method == "flat" & supported)
runtime_caps <- nn_capabilities(runtime = TRUE)
```

nn_gpu

GPU-resident tuned nearest-neighbour search

Description

'nn_gpu()' runs a CUDA nearest-neighbour search and returns a GPU-resident result object. Unlike 'nn()', the 'indices' and 'distances' are not copied back into R matrices. The returned object stores owning and non-owning external pointers so downstream C/C++ packages can consume the KNN output on the CUDA device.

Usage

```
nn_gpu(
  data,
  points = data,
  k = NULL,
  exclude_self = FALSE,
  method = c("auto", "exact", "flat", "bruteforce"),
  metric = c("euclidean", "cosine", "correlation"),
  tuning = c("auto", "cache", "pilot", "fixed", "off", "none"),
  target_recall = 0.99
)
```

Arguments

data	Dense numeric matrix/data frame or optional <code>'float::fl()'</code> / <code>'float32'</code> reference matrix. Sparse, delayed, file-backed, and other matrix-like objects are rejected before conversion.
points	Optional query matrix. Defaults to <code>'data'</code> .
k	Number of neighbours.
exclude_self	Logical; remove each row from its own neighbour list for self-query calls.
method	<code>"auto"</code> , <code>"exact"</code> , <code>"flat"</code> , or <code>"bruteforce"</code> . <code>"auto"</code> consults the compiled shape/k/metric/target-recall selector but currently returns GPU-resident exact-family buffers.
metric	<code>"euclidean"</code> , <code>"cosine"</code> , or <code>"correlation"</code> .
tuning	Tuning label to record. The current GPU-resident route is exact, so <code>'target_recall'</code> is metadata for the executed route; auto-selected approximate settings are recorded separately as <code>'auto_preferred_tuning'</code> .
target_recall	Target recall label to record; use <code>'0.9'</code> , <code>'0.95'</code> , or <code>'0.99'</code> .

Details

This is intentionally narrower than `'nn()'`: for Euclidean `'method = "auto"`, `"exact"`, `"flat"`, or `"bruteforce"`, `'nn_gpu()'` uses a FAISS GPU direct `'bfKnn'` route and keeps the result buffers on the CUDA device when FAISS GPU is available. Euclidean inputs with two or three columns instead use the native direct-difference CUDA exact kernel. This avoids cancellation in the dot-product L2 identity for nearly coincident low-dimensional vectors while retaining GPU-resident results. When FAISS was built with cuVS support, FAISS may dispatch the brute-force GPU distance primitive through cuVS internally. Cosine and correlation currently keep using the native CUDA GPU-resident exact route; they are transformed to normalized squared L2 on the C++ side and stored as `'1 - similarity'`. Approximate cuVS/FAISS GPU methods currently fail clearly here because those provider result buffers still need provider-specific persistent GPU ownership. For `'method = "auto"`, `'nn_gpu()'` records the same C++ auto-selection metadata used by `'nn()'`. If that policy would prefer an approximate CUDA method whose result buffers are not yet exposed as persistent GPU-resident objects, `'nn_gpu()'` keeps the exact-family GPU-resident route and records `'auto_preferred_backend'`, `'auto_preferred_method'`, and `'auto_residency_constraint'`. It also records the actual exact-family `'execution_tuning'` used by the GPU-resident route and the

‘auto_preferred_tuning’ row for the approximate method selected by the compiled policy when available. That policy is an optional experimental L40S-calibrated heuristic for cold full-self-search. Because it is L40S-informed, unmatched or unidentified CUDA hardware emits the same once-per-model extrapolation warning as ‘nn()’; set ‘options(faissR.warn_hardware_extrapolation = FALSE)’ only after reviewing the returned ‘auto_selection’ metadata.

Value

A ‘faissR_gpu_knn’ list. It contains an owning ‘handle’, non-owning ‘indices_ptr’ and ‘distances_ptr’, ‘n_query’, ‘k’, ‘index_base = 1L’, ‘indices_type = "int32"’, ‘distance_type = "float32"’, ‘layout’, ‘metric’, and ‘backend_used’. The object also declares the same distance-contract fields as ‘nn()’. With ‘tuning = "auto"’, it also includes ‘execution_tuning’; with ‘method = "auto"’, it includes ‘auto_preferred_tuning’ when the compiled selector has a preferred CUDA method/tuning row. Ownership metadata states that ‘handle’ owns both allocations, the buffer pointers are non-owning, the producing device is synchronized before return, and serialization/interprocess sharing are not supported.

Examples

```
if (cuda_available()) {
  x <- matrix(rnorm(200), ncol = 4)
  gpu_knn <- nn_gpu(x, k = 5, exclude_self = TRUE)
  print(gpu_knn)
}
```

nn_metric_preflight	<i>Preflight nearest-neighbor metric inputs</i>
---------------------	---

Description

Inspect reference and query rows for values that affect the public distance contract before calling [nn()]. This function performs no neighbor search.

Usage

```
nn_metric_preflight(
  data,
  points = NULL,
  metric = c("euclidean", "cosine", "correlation"),
  backend = NULL
)
```

Arguments

data	Dense numeric reference matrix, data frame, or optional ‘float::fl()’/‘float32’ matrix.
points	Optional dense query object with the same number of columns as ‘data’. ‘NULL’ represents self-query and reuses the ‘data’ inspection.

metric	One of "euclidean", "cosine", or "correlation".
backend	Requested backend ("cpu" or "cuda"). 'NULL' follows the package backend option and environment-variable policy.

Details

For cosine, a degenerate row is exactly all zero after conversion to the inspected representation. For correlation, it is a row whose entries are all equal. 'NA', 'NaN', 'Inf', and '-Inf' are non-finite and are rejected by every backend. Row indices are one-based. The scan is 'O((n + m) p)' and is intentionally explicit so it need not add another full pass to every nearest-neighbor call. Cosine/correlation normalization rescales finite rows in double precision before conversion to float32. This avoids interpreting numerical overflow as a zero vector or constant row. The preflight checks metric inputs, not installed capabilities, method-specific size limits, or device memory. Euclidean routes using float32 additionally require coordinates within its finite range; large squared distances can also exceed provider precision.

Value

A list containing one-based 'data_degenerate_rows' and 'points_degenerate_rows' for the metric-specific degenerate condition, corresponding non-finite row indices, 'would_succeed', and a stable 'action' label. 'data_rows' and 'points_rows' are retained as compatibility aliases. CPU cosine and correlation use the documented zero-normalized convention; CUDA rejects affected rows.

Examples

```
x <- rbind(c(0, 0), c(1, 0), c(1, 1))
nn_metric_preflight(x, metric = "cosine", backend = "cuda")
```

predict.faissR_knn_model

Predict from a faissR kNN model

Description

Predict from a faissR kNN model

Usage

```
## S3 method for class 'faissR_knn_model'
predict(
  object,
  newdata,
  k = NULL,
  backend = NULL,
  tuning = c("auto", "cache", "pilot", "fixed", "off", "none"),
  target_recall = NULL,
  cagra_implementation = NULL,
```

```

    cagra_build_algo = NULL,
    vote = c("majority", "weighted"),
    type = c("response", "prob"),
    ...
)

```

Arguments

object	A model returned by knn() .
newdata	Numeric or optional 'float::fl()'/float32' query matrix with observations in rows. Float32 queries are preserved for methods with direct float32 adapters.
k	Number of neighbours.
backend	Device backend used for this prediction call: "cpu" or "cuda". 'NULL' follows the package backend configuration. The fitted model's method and metric are always reused.
tuning	Tuning policy used for this prediction call. "auto" uses the deterministic default for the resolved method; pilot/cache tuning is opt-in where implemented. FAISS GPU IVF pilot/cache tuning is Euclidean-only; non-Euclidean IVF routes use deterministic metric-aware defaults.
target_recall	Optional speed/recall tier for this prediction call. 'NULL' reuses the fitted model's value; otherwise use '0.9', '0.95', or '0.99'. It affects CUDA auto Flat-vs-IVF selection, CUDA IVF probing, and HNSW graph-search tiers when prediction needs a new NN search. For approximate methods this is a selection objective rather than a recall guarantee.
cagra_implementation	CUDA CAGRA provider for this prediction call. 'NULL' reuses the fitted model's setting, then the global option.
cagra_build_algo	Direct RAPIDS cuVS CAGRA graph-build algorithm for this prediction call. 'NULL' reuses the fitted model's setting, then the global option.
vote	"majority" or "weighted" for classification; "majority" means an unweighted neighbour mean for regression.
type	"response" for class/regression predictions or "prob" for class probability matrices from classification models.
...	Reserved for future options.

Value

A factor for classification, a numeric vector for regression, or a numeric class-probability matrix when 'type = "prob"'. Outputs carry 'attr(result, "faissR_nn")' route metadata, approximation parameters, and auto-selection metadata from the underlying [nn\(\)](#) call. The prediction-time neighbour search is batched: the full 'newdata' matrix is sent to the resolved FAISS/cuVS/native NN route in one call, and metadata records 'batch_query', 'query_n', and 'query_call_count'.

```
print.faissR_knn_model
```

Print a fitted faissR kNN model

Description

Print a fitted faissR kNN model

Usage

```
## S3 method for class 'faissR_knn_model'
print(x, ...)
```

Arguments

x	A model returned by <code>knn()</code> .
...	Reserved for compatibility with <code>print()</code> .

Value

'x', invisibly.

```
summary.faissR
```

Summaries of faissR result objects

Description

These methods return compact one-row data frames describing nearest-neighbor results, GPU-resident results, k-means fits, and fitted kNN models.

Usage

```
## S3 method for class 'faissR_nn'
summary(object, ...)

## S3 method for class 'faissR_gpu_knn'
summary(object, ...)

## S3 method for class 'faissR_kmeans'
summary(object, ...)

## S3 method for class 'faissR_knn_model'
summary(object, ...)
```

Arguments

object	A faissR result or model object.
...	Reserved for compatibility with the S3 generics.

Value

A one-row data frame containing the principal dimensions, backend, metric, and fit or residency metadata for 'object'.

zeisel_brain_pca	<i>Zeisel brain PCA vignette artifact</i>
------------------	---

Description

A compact SingleCellExperiment containing 50 principal-component coordinates and selected cell annotations for 3,005 cells from the Zeisel mouse-brain single-cell RNA-sequencing dataset. The expression assay is not redistributed. The object is installed as `extdata/zeisel_brain_pca.rds` and is loaded with `readRDS()`.

Format

A SingleCellExperiment with zero assay rows, 3,005 columns, a 3,005 by 50 "PCA" reduced-dimension matrix, and the available `level1class`, `level2class`, `tissue`, and `group` # cell annotations. `metadata(object)$faissR_provenance` records the upstream source, preprocessing, seed, source dimensions, access date, and package versions.

Details

The installed artifact is generated by `inst/scripts/prepare_zeisel_brain_pca.R`. That script obtains `scRNAseq::ZeiselBrainData(ensembl = TRUE, location = FALSE)`, applies `scuttle::logNormCounts()`, selects up to 2,000 highly variable genes from `scran::modelGeneVar()`, and computes 50 principal components with `scater::runPCA()` and `BiocSingular::IrlbaParam()` using seed 1001.

Value

This documentation topic describes the object returned by `readRDS(system.file("extdata", "zeisel_brain_pca.rds", package = "faissR"))`. It is a SingleCellExperiment; no object is attached to the search path.

Source

Zeisel A, Munoz-Manchado AB, Codeluppi S, et al. (2015). Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. **Science**, 347(6226), 1138-1142. doi:10.1126/science.aaa1934

Upstream Bioconductor package: <https://bioconductor.org/packages/scRNAseq>

Consult the upstream package and publication for the source-data terms.

Examples

```
path <- system.file("extdata", "zeisel_brain_pca.rds", package = "faissR")
if (
  nzchar(path) &&
  requireNamespace("SingleCellExperiment", quietly = TRUE)
) {
  sce <- readRDS(path)
  dim(SingleCellExperiment::reducedDim(sce, "PCA"))
  S4Vectors::metadata(sce)$faissR_provenance$source_reference
}
```

Index

- * **datasets**
 - zeisel_brain_pca, [26](#)
- * **programming**
 - faissR-c-api, [7](#)
- backend_info, [3](#)
- candidate_knn, [3](#)
- cuda_available, [5](#)
- cuvs_available, [5](#)
- faiss_available, [6](#)
- faiss_gpu_available, [6](#)
- faissR-c-api, [7](#)
- faissR_backend, [8](#)
- fast_kmeans, [8](#)
- gpu_knn_to_host, [10](#)
- knn, [11](#), [24](#), [25](#)
- nn, [13](#), [13](#), [24](#)
- nn_capabilities, [3](#), [19](#)
- nn_gpu, [20](#)
- nn_metric_preflight, [22](#)
- predict.faissR_knn_model, [23](#)
- print.faissR_knn_model, [25](#)
- summary.faissR, [25](#)
- summary.faissR_gpu_knn
(summary.faissR), [25](#)
- summary.faissR_kmeans (summary.faissR),
[25](#)
- summary.faissR_knn_model
(summary.faissR), [25](#)
- summary.faissR_nn (summary.faissR), [25](#)
- zeisel_brain_pca, [26](#)