

Package: annsearchR (via r-universe)

October 8, 2026

Title Fast Approximate Nearest Neighbour Search

Version 0.0.1

Description R interface to the Rust crate 'ann-search-rs'. Build an exact or approximate nearest neighbour index once (HNSW, Annoy, IVF, NNDescent, Vamana and more), keep it in memory behind an R6 object and query it as often as you like. Multi-threaded via Rayon, with f32 or f64 storage.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports checkmate, lifecycle, methods, R6, stats

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.90, xz

Depends R (>= 4.2)

Suggests BiocNeighbors, data.table, FNN, ggplot2, hdf5r, knitr, quarto, RANN, RcppAnnoy, RcppHNSW, rnndescent, scales, tinytest

Config/roxygen2/version 8.1.0

VignetteBuilder quarto

URL <https://gregorlueg.github.io/annsearchR/>,
<https://github.com/GregorLueg/annsearchR>

Config/pak/sysreqs xz-utils libclang-dev

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

Date/Publication 2026-10-08 17:56:07 UTC

RemoteUrl <https://github.com/GregorLueg/annsearchR>

RemoteRef v0.0.1

RemoteSha 493b4bf347b23c4ae594d32dd25e7f6031384c4b

Contents

ann_get_threads	2
ann_set_threads	3
AnnIndex	3
AnnoyIndex	5
BallTreeIndex	6
ExhaustiveIndex	8
generate_cell_embeddings	9
generate_clustered_data	9
generate_correlated_data	10
generate_low_rank_data	11
HnswIndex	12
IvfIndex	13
KdTreeIndex	15
KmknIndex	16
knn_recall	17
load_ann_index	18
LshIndex	19
NNDescentIndex	20
NsgIndex	22
predict.AnnIndex	24
RnnDescentIndex	24
SoarIndex	26
subsample_queries	27
VamanaIndex	28
Index	30

ann_get_threads	<i>Number of threads used for index builds and queries</i>
-----------------	--

Description

Number of threads used for index builds and queries

Usage

ann_get_threads()

Value

Integer.

ann_set_threads	<i>Set the number of threads for index builds and queries</i>
-----------------	---

Description

Installs a dedicated thread pool for all subsequent builds and queries. Can be called as often as you like.

Usage

```
ann_set_threads(n)
```

Arguments

n	Integer. Number of threads. 0L returns to the default pool, which uses all cores or honours RAYON_NUM_THREADS if set.
---	---

Value

The previous thread count, invisibly.

AnnIndex	<i>Base class for all nearest neighbour indices</i>
----------	---

Description

Holds a pointer to an index living in Rust memory, plus the bits R needs to talk to it (metric, precision, shape). Never instantiated directly: use one of the algorithm classes such as [HnswIndex](#), [AnnoyIndex](#) or [ExhaustiveIndex](#).

The index itself is not serialised by [saveRDS\(\)](#). A restored object holds a dead pointer and every method errors. Use [\\$save\(\)](#) and [load_ann_index\(\)](#) instead.

Active bindings

n	Integer. Number of indexed samples. Read-only.
dim	Integer. Number of features. Read-only.
metric	String. Distance metric. Read-only.
precision	String. "float" or "double". Read-only.

Methods

Public methods:

- [AnnIndex\\$predict\(\)](#)
- [AnnIndex\\$query_self\(\)](#)
- [AnnIndex\\$save\(\)](#)
- [AnnIndex\\$print\(\)](#)
- [AnnIndex\\$clone\(\)](#)

AnnIndex\$predict(): Find the k nearest indexed samples for each row of newdata.

Usage:

```
AnnIndex$predict(newdata, k = 15L, return_dist = TRUE, .verbose = FALSE)
```

Arguments:

newdata Numeric matrix or data.frame. Queries x features, same number of features as the index.

k Integer. Number of neighbours.

return_dist Boolean. Return the distances as well.

.verbose Boolean. Print progress from Rust.

Returns: A list with:

- **idx** - Integer matrix, queries x k . 1-based row indices into the indexed data. NA where fewer than k neighbours were found.
- **dist** - Numeric matrix, queries x k , or NULL if **return_dist** = FALSE. Inf where **idx** is NA.

AnnIndex\$query_self(): kNN graph over the indexed data itself. Uses each index's own self-query path, which is faster than **\$predict()** on the training data.

Usage:

```
AnnIndex$query_self(k = 15L, return_dist = TRUE, .verbose = FALSE)
```

Arguments:

k Integer. Number of neighbours, including the sample itself.

return_dist Boolean. Return the distances as well.

.verbose Boolean. Print progress from Rust.

Returns: A list with **idx** and **dist**, as for **\$predict()**, with one row per indexed sample.

AnnIndex\$save(): Write the index to a directory. Read it back with [load_ann_index\(\)](#).

Usage:

```
AnnIndex$save(dir)
```

Arguments:

dir String. Target directory, created if missing.

Returns: The object, invisibly.

AnnIndex\$print(): Print a one-line summary.

Usage:

```
AnnIndex$print(...)
```

Arguments:

... Ignored.

Returns: The object, invisibly.

`AnnIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

`AnnIndex$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

AnnoyIndex

Annoy index

Description

Random projection forest, as in Spotify's Annoy. More trees means better recall, a larger index and a slower build. Raise `search_budget` for recall at query time. No Manhattan.

Super class

[AnnIndex](#) -> AnnoyIndex

Active bindings

`search_budget` Integer or NULL. Candidates inspected per query.

Methods

Public methods:

- [AnnoyIndex\\$new\(\)](#)
- [AnnoyIndex\\$clone\(\)](#)

`AnnoyIndex$new()`: Build the index.

Usage:

```
AnnoyIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine"),
  n_trees = 25L,
  search_budget = NULL,
  seed = 42L,
  precision = c("float", "double")
)
```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; [load_ann_index\(\)](#) uses that path.

metric String. One of c("euclidean", "sqeuclidean", "cosine").
 n_trees Integer. Number of trees in the forest.
 search_budget Integer or NULL. Candidates inspected per query. NULL uses $k * n_trees * 20$.
 Can be changed after the build.
 seed Integer. Fixes the random hyperplanes.
 precision String. "float" stores the data as f32, "double" as f64.

AnnoyIndex\$clone(): The objects of this class are cloneable with this method.

Usage:

AnnoyIndex\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

References

Bernhardsson, Annoy, <https://github.com/spotify/annoy>

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- AnnoyIndex$new(x, n_trees = 25L)
res <- idx$predict(x[1:5, ], k = 10L)
```

BallTreeIndex

Ball tree index

Description

Metric tree of nested hyperspheres, pruned by the triangle inequality. Pays off on data with real cluster structure at moderate dimensionality. The default search budget (5% of the indexed points) makes it approximate; raise it for recall. Past about 10% recall plateaus and query time does not fall. No Manhattan.

Super class

[AnnIndex](#) -> BallTreeIndex

Active bindings

search_budget Integer or NULL. Points inspected per query.

Methods

Public methods:

- [BallTreeIndex\\$new\(\)](#)
- [BallTreeIndex\\$clone\(\)](#)

BallTreeIndex\$new(): Build the index.

Usage:

```
BallTreeIndex$new(  
  data,  
  metric = c("euclidean", "sqeuclidean", "cosine"),  
  search_budget = NULL,  
  seed = 42L,  
  precision = c("float", "double")  
)
```

Arguments:

data Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; [load_ann_index\(\)](#) uses that path.

metric String. One of c("euclidean", "sqeuclidean", "cosine").

search_budget Integer or NULL. Points inspected per query. NULL uses 5% of the indexed points. Can be changed after the build.

seed Integer. Fixes the pivot choice at each split.

precision String. "float" stores the data as f32, "double" as f64.

BallTreeIndex\$clone(): The objects of this class are cloneable with this method.

Usage:

```
BallTreeIndex$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Omohundro, ICSI Technical Report, 1989

Examples

```
x <- generate_clustered_data(1000L, 16L)$data  
idx <- BallTreeIndex$new(x)  
res <- idx$predict(x[1:5, ], k = 10L)
```

ExhaustiveIndex	<i>Exhaustive (brute-force) index</i>
-----------------	---------------------------------------

Description

Exact search: every query is compared to every indexed sample, SIMD accelerated and multi-threaded. The ground truth to measure the approximate indices against, and faster than you might think up to a few hundred thousand samples. Large query batches go through a blocked GEMM path, which on macOS runs on Apple Accelerate, so it can outrun the thread count set by `ann_set_threads()`.

Super class

`AnnIndex` -> ExhaustiveIndex

Methods

Public methods:

- `ExhaustiveIndex$new()`
- `ExhaustiveIndex$clone()`

`ExhaustiveIndex$new()`: Build the index.

Usage:

```
ExhaustiveIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine", "manhattan"),
  precision = c("float", "double")
)
```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; `load_ann_index()` uses that path.

`metric` String. One of `c("euclidean", "sqeuclidean", "cosine", "manhattan")`.

`precision` String. "float" stores the data as f32, "double" as f64.

`ExhaustiveIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExhaustiveIndex$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- ExhaustiveIndex$new(x)
res <- idx$predict(x[1:5, ], k = 10L)
```

`generate_cell_embeddings`*Generate synthetic cell embeddings*

Description

Foundation-model cell embeddings in the style of Geneformer or scGPT. A large shared mean offset puts every cell inside an anisotropy cone, a few axis-aligned rogue dimensions dominate dot products, each cell type varies in its own low-rank subspace, and a per-cell scale stands in for library size. The nastiest of the four generators.

Usage

```
generate_cell_embeddings(n, dim, n_clusters = 25L, seed = 42L)
```

Arguments

<code>n</code>	Integer. Number of samples.
<code>dim</code>	Integer. Number of features.
<code>n_clusters</code>	Integer. Number of clusters.
<code>seed</code>	Integer. Random seed.

Value

A list with:

- `data` - Numeric matrix, $n \times \text{dim}$.
- `labels` - Integer vector of 1-based cluster labels.

Examples

```
dat <- generate_cell_embeddings(1000L, 64L, n_clusters = 8L)
```

`generate_clustered_data`*Generate clustered synthetic data*

Description

Separated Gaussian clusters, with a fifth of the points on thin bridges between neighbouring clusters so the boundaries aren't trivially clean. The baseline of the four generators.

All generators are the ones behind the Rust crate's benchmark tables. They draw in f32 like the crate's gridsearch examples and the Python bindings, so the same seed gives the same points in all three.

Usage

```
generate_clustered_data(n, dim, n_clusters = 25L, seed = 42L)
```

Arguments

n	Integer. Number of samples.
dim	Integer. Number of features.
n_clusters	Integer. Number of clusters.
seed	Integer. Random seed.

Value

A list with:

- data - Numeric matrix, n x dim.
- labels - Integer vector of 1-based cluster labels.

Examples

```
dat <- generate_clustered_data(1000L, 16L, n_clusters = 5L)
table(dat$labels)
```

generate_correlated_data

Generate correlated synthetic data

Description

Well-separated clusters, each an arbitrarily oriented ellipsoid, plus a globally shared off-axis subspace that correlates the features. None of the structured variance lines up with the coordinate axes, and bridges between neighbouring clusters keep the blobs reachable for graph indices.

Usage

```
generate_correlated_data(  
  n,  
  dim,  
  n_clusters = 25L,  
  cor_strength = 0.5,  
  seed = 42L  
)
```

Arguments

n	Integer. Number of samples.
dim	Integer. Number of features.
n_clusters	Integer. Number of clusters.
cor_strength	Numeric. Share of the structured variance in the shared off-axis subspace, between 0 (all cluster-local) and 1 (all shared). 0.5 is the value behind the crate's benchmark tables.
seed	Integer. Random seed.

Value

A list with:

- data - Numeric matrix, n x dim.
- labels - Integer vector of 1-based cluster labels.

Examples

```
dat <- generate_correlated_data(1000L, 32L, n_clusters = 5L)
```

```
generate_low_rank_data
```

Generate low-rank synthetic data

Description

Cell types on an `intrinsic_dim`-dimensional manifold, isometrically embedded in `dim` ambient dimensions with a touch of noise. Types are grouped into lineages, with curved differentiation trajectories running between types of the same lineage.

Usage

```
generate_low_rank_data(
  n,
  dim,
  n_clusters = 25L,
  intrinsic_dim = 16L,
  seed = 42L
)
```

Arguments

n	Integer. Number of samples.
dim	Integer. Number of features.
n_clusters	Integer. Number of clusters.
intrinsic_dim	Integer. True dimensionality of the manifold. Must not exceed dim.
seed	Integer. Random seed.

Value

A list with:

- data - Numeric matrix, n x dim.
- labels - Integer vector of 1-based cluster labels.

Examples

```
dat <- generate_low_rank_data(1000L, 64L, intrinsic_dim = 8L)
```

HnswIndex	<i>HNSW index</i>
-----------	-------------------

Description

Hierarchical navigable small world graph. The usual first choice: high recall at low query latency. Raise `ef_search` for recall at query time, `ef_construction` for a better graph at build time.

Nodes are inserted in parallel, so two builds with the same seed give slightly different graphs. Need bit-identical results? Build with `ann_set_threads(1L)`.

Super class

[AnnIndex](#) -> HnswIndex

Active bindings

`ef_search` Integer. Beam width at query time.

Methods**Public methods:**

- [HnswIndex\\$new\(\)](#)
- [HnswIndex\\$clone\(\)](#)

`HnswIndex$new()`: Build the index.

Usage:

```
HnswIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine", "manhattan"),
  m = 16L,
  ef_construction = 200L,
  ef_search = 50L,
  seed = 42L,
  precision = c("float", "double"),
  .verbose = FALSE
)
```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; `load_ann_index()` uses that path.

`metric` String. One of c("euclidean", "sqeuclidean", "cosine", "manhattan").

`m` Integer. Edges per node on the upper layers, $2 * m$ on layer 0. 16 suits most data; 32 to 48 helps in high dimensions.

`ef_construction` Integer. Candidate list width during the build. Better graph, slower build, no cost at query time.

`ef_search` Integer. Beam width at query time. Raised to `k` internally if smaller. Can be changed after the build.

`seed` Integer. Fixes the layer assignment.

`precision` String. "float" stores the data as f32, "double" as f64.

`.verbose` Boolean. Print build progress from Rust.

`HnswIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
HnswIndex$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Malkov & Yashunin, IEEE TPAMI, 2020

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- HnswIndex$new(x, metric = "cosine")
idx$ef_search <- 100L
res <- idx$predict(x[1:5, ], k = 10L)
```

IvfIndex

IVF index

Description

Inverted file over k-means Voronoi cells. Cheap to build and easy to tune: `nlist` sets how finely the space is cut, `nprobe` how many cells a query visits. The smallest of the unquantised approximate indices. No Manhattan.

Super class

`AnnIndex` -> `IvfIndex`

Active bindings

nprobe Integer or NULL. Cells visited per query.

Methods

Public methods:

- [IvfIndex\\$new\(\)](#)
- [IvfIndex\\$clone\(\)](#)

`IvfIndex$new()`: Build the index.

Usage:

```
IvfIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine"),
  nlist = NULL,
  nprobe = NULL,
  kmeans_iters = NULL,
  kmeans_balanced = FALSE,
  seed = 42L,
  precision = c("float", "double"),
  .verbose = FALSE
)
```

Arguments:

data Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; [load_ann_index\(\)](#) uses that path.

metric String. One of c("euclidean", "sqeuclidean", "cosine").

nlist Integer or NULL. Number of Voronoi cells. NULL uses sqrt(n).

nprobe Integer or NULL. Cells visited per query. NULL uses sqrt(nlist). Can be changed after the build.

kmeans_iters Integer or NULL. Lloyd iterations. NULL uses 30.

kmeans_balanced Boolean. Reseed starved centroids each iteration.

seed Integer. Fixes the k-means initialisation.

precision String. "float" stores the data as f32, "double" as f64.

.verbose Boolean. Print build progress from Rust.

`IvfIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
IvfIndex$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Jégou, Douze & Schmid, IEEE TPAMI, 2011

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- IvfIndex$new(x)
idx$nprobe <- 10L
res <- idx$predict(x[1:5, ], k = 10L)
```

KdTreeIndex

*kd forest index***Description**

Forest of randomised kd spill-trees. Same trade as [AnnoyIndex](#) with axis-aligned splits instead of random hyperplanes: more trees means better recall and a larger index. Points near a split land in both children, which lets one descent recover neighbours a hard split would separate. The one tree index here that supports Manhattan.

Super class

[AnnIndex](#) -> KdTreeIndex

Active bindings

search_budget Integer or NULL. Candidates inspected per query.

Methods**Public methods:**

- [KdTreeIndex\\$new\(\)](#)
- [KdTreeIndex\\$clone\(\)](#)

KdTreeIndex\$new(): Build the index.

Usage:

```
KdTreeIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine", "manhattan"),
  n_trees = 25L,
  search_budget = NULL,
  seed = 42L,
  precision = c("float", "double")
)
```

Arguments:

data Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; [load_ann_index\(\)](#) uses that path.

metric String. One of c("euclidean", "sqeuclidean", "cosine", "manhattan").

n_trees Integer. Number of trees in the forest.
search_budget Integer or NULL. Candidates inspected per query. NULL uses $k * n_trees * 20$.
 Can be changed after the build.
seed Integer. Fixes the split dimensions.
precision String. "float" stores the data as f32, "double" as f64.

KdTreeIndex\$clone(): The objects of this class are cloneable with this method.

Usage:

```
KdTreeIndex$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Muja & Lowe, IEEE TPAMI, 2014

Examples

```

x <- generate_clustered_data(1000L, 16L)$data
idx <- KdTreeIndex$new(x)
res <- idx$predict(x[1:5, ], k = 10L)

```

KmknnIndex

kMkNN index

Description

Exact search over a k-means partition, pruning whole clusters by the triangle inequality. Same answers as [ExhaustiveIndex](#); whether it is faster depends on how well the data clusters, so time both. No Manhattan.

Super class

[AnnIndex](#) -> KmknnIndex

Methods

Public methods:

- [KmknnIndex\\$new\(\)](#)
- [KmknnIndex\\$clone\(\)](#)

KmknnIndex\$new(): Build the index.

Usage:

```
KmknnIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine"),
  nlist = NULL,
  kmeans_iters = NULL,
  kmeans_balanced = FALSE,
  seed = 42L,
  precision = c("float", "double"),
  .verbose = FALSE
)
```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted;

`load_ann_index()` uses that path.

`metric` String. One of c("euclidean", "sqeuclidean", "cosine").

`nlist` Integer or NULL. Number of k-means clusters. NULL uses $\sqrt{n}$.

`kmeans_iters` Integer or NULL. Lloyd iterations. NULL uses 30.

`kmeans_balanced` Boolean. Reseed starved centroids each iteration.

`seed` Integer. Fixes the k-means initialisation.

`precision` String. "float" stores the data as f32, "double" as f64.

`.verbose` Boolean. Print build progress from Rust.

`KmknnIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
KmknnIndex$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Wang, IEEE ICDMW, 2012

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- KmknnIndex$new(x)
res <- idx$predict(x[1:5, ], k = 10L)
```

knn_recall

Recall of an approximate kNN result

Description

Fraction of the true k nearest neighbours that the approximate search found, averaged over queries. Order within a row does not matter.

Usage

```
knn_recall(truth, approx, k = NULL)
```

Arguments

truth	Integer matrix. Queries x k true neighbour indices, e.g. the idx element from an ExhaustiveIndex .
approx	Integer matrix. Queries x k approximate neighbour indices.
k	Optional integer. Only the first k columns of both are compared. Defaults to ncol(approx).

Value

Numeric between 0 and 1.

load_ann_index	<i>Load an index written by \$save()</i>
----------------	--

Description

Reads the index bundle and its R-side parameters (metric, search knobs) back into the matching [AnnIndex](#) subclass. The float width is read from the bundle. Bundles are tied to the on-disk format version of the underlying Rust crate; a format bump means rebuilding.

Usage

```
load_ann_index(dir)
```

Arguments

dir	String. Directory written by \$save().
-----	--

Value

An [AnnIndex](#) subclass, ready to query.

LshIndex

*LSH index***Description**

Multi-probe locality-sensitive hashing over random projections. By far the cheapest index to build here, and the weakest on recall for the query time it costs; the one index whose query can come out slower than brute force. Fewer `bits_per_hash` widens the buckets, more `num_tables` trades memory for recall. No Manhattan.

Super class

[AnnIndex](#) -> LshIndex

Active bindings

`n_probe` Integer or NULL. Buckets probed per table.

`max_candidates` Integer or NULL. Cap on candidates scored.

Methods**Public methods:**

- [LshIndex\\$new\(\)](#)
- [LshIndex\\$clone\(\)](#)

`LshIndex$new()`: Build the index.

Usage:

```
LshIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine"),
  num_tables = 8L,
  bits_per_hash = 12L,
  slot_bits = NULL,
  n_probe = NULL,
  max_candidates = NULL,
  seed = 42L,
  precision = c("float", "double")
)
```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted;

[load_ann_index\(\)](#) uses that path.

`metric` String. One of `c("euclidean", "sqeuclidean", "cosine")`.

`num_tables` Integer. Independent hash tables.

`bits_per_hash` Integer. Total bits in a bucket code.

slot_bits Integer or NULL. Bits each quantised projection contributes. NULL uses 1 for cosine and 2 otherwise.

n_probe Integer or NULL. Buckets probed per table. NULL uses one per projection. Can be changed after the build.

max_candidates Integer or NULL. Cap on candidates scored per query. NULL means no cap. Can be changed after the build.

seed Integer. Fixes the random projections.

precision String. "float" stores the data as f32, "double" as f64.

LshIndex\$clone(): The objects of this class are cloneable with this method.

Usage:

```
LshIndex$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Lv et al., VLDB, 2007

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- LshIndex$new(x)
res <- idx$predict(x[1:5, ], k = 10L)
```

NNDescentIndex

NN-Descent index

Description

Builds the kNN graph directly by iterative local join. Want the kNN graph of the data itself (UMAP, clustering)? \$extract_knn() hands back the converged graph without any search, far cheaper than \$query_self(). It reads the post-pruning graph, so leave diversify_prob at 0 if extraction is the point.

Super class

[AnnIndex](#) -> NNDescentIndex

Active bindings

ef_search Integer or NULL. Beam width at query time.

k_graph Integer. Degree of the built graph. Read-only.

Methods

Public methods:

- [NNDescentIndex\\$new\(\)](#)
- [NNDescentIndex\\$extract_knn\(\)](#)
- [NNDescentIndex\\$clone\(\)](#)

`NNDescentIndex$new()`: Build the index.

Usage:

```
NNDescentIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine", "manhattan"),
  k_graph = 15L,
  delta = 0.001,
  diversify_prob = 0,
  max_iter = NULL,
  max_candidates = NULL,
  n_trees = NULL,
  ef_search = NULL,
  seed = 42L,
  precision = c("float", "double"),
  .verbose = FALSE
)
```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; [load_ann_index\(\)](#) uses that path.

`metric` String. One of `c("euclidean", "sqeuclidean", "cosine", "manhattan")`.

`k_graph` Integer. Neighbours per node in the graph being built.

`delta` Numeric. Convergence threshold: descent stops once the fraction of updated edges falls below it.

`diversify_prob` Numeric between 0 and 1. Probability of pruning an occluded edge after descent. 0 disables pruning.

`max_iter` Integer or NULL. Iteration cap. NULL uses `max(round(log2(n)), 5)`.

`max_candidates` Integer or NULL. Neighbours sampled per node per local join. NULL uses `min(k_graph, 60)`.

`n_trees` Integer or NULL. Random projection trees for the seed graph. NULL uses `min(5 + round(n^0.25), 12)`.

`ef_search` Integer or NULL. Beam width at query time. NULL uses `clamp(2 * k, 50, 200)`. Can be changed after the build.

`seed` Integer. Fixes the seed graph and the sampling.

`precision` String. "float" stores the data as f32, "double" as f64.

`.verbose` Boolean. Print build progress from Rust.

`NNDescentIndex$extract_knn()`: The converged kNN graph over the indexed data, read straight off the index. No search involved.

Usage:

```
NNDescentIndex$extract_knn(k = NULL, include_self = TRUE, return_dist = TRUE)
```

Arguments:

`k` Integer or NULL. Row length, including the sample itself when `include_self = TRUE`. NULL uses the graph degree `k_graph`.

`include_self` Boolean. Put each sample first at distance 0, as `$query_self()` does.

`return_dist` Boolean. Return the distances as well.

Returns: A list with `idx` and `dist`, as for `$predict()`, with one row per indexed sample. Rows the graph cannot fill are NA padded.

`NNDescentIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NNDescentIndex$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Dong, Moses & Li, WWW, 2011

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- NNDescentIndex$new(x, k_graph = 15L)
knn <- idx$extract_knn()
```

NsgIndex

NSG index

Description

Navigating spreading-out graph. Builds an NN-Descent kNN graph of degree `knn_k` first, then refines it into a sparse monotonic graph. Smallest of the graph indices; pays for it with a double build.

Super class

[AnnIndex](#) -> NsgIndex

Active bindings

`ef_search` Integer or NULL. Beam width at query time.

Methods

Public methods:

- [NsgIndex\\$new\(\)](#)
- [NsgIndex\\$clone\(\)](#)

`NsgIndex$new()`: Build the index.

Usage:

```
NsgIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine", "manhattan"),
  r = 32L,
  l_build = 100L,
  c = 500L,
  knn_k = 64L,
  ef_search = NULL,
  seed = 42L,
  precision = c("float", "double"),
  .verbose = FALSE
)
```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted;

`load_ann_index()` uses that path.

`metric` String. One of `c("euclidean", "sqeuclidean", "cosine", "manhattan")`.

`r` Integer. Maximum out-degree of the refined graph.

`l_build` Integer. Candidate list width while refining.

`c` Integer. Candidate pool size per node before pruning.

`knn_k` Integer. Degree of the NN-Descent graph built first. Wants to be comfortably above `r`.

`ef_search` Integer or NULL. Beam width at query time. NULL uses 100. Can be changed after the build.

`seed` Integer. Fixes the initial graph and navigating node.

`precision` String. "float" stores the data as f32, "double" as f64.

`.verbose` Boolean. Print build progress from Rust.

`NsgIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NsgIndex$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Fu et al., VLDB, 2019

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- NsgIndex$new(x)
res <- idx$predict(x[1:5, ], k = 10L)
```

predict.AnnIndex	<i>Query an index</i>
------------------	-----------------------

Description

S3 wrapper around the `$predict()` method of an [AnnIndex](#).

Usage

```
## S3 method for class 'AnnIndex'
predict(object, newdata, k = 15L, ...)
```

Arguments

object	An AnnIndex .
newdata	Numeric matrix or data.frame. Queries x features.
k	Integer. Number of neighbours.
...	Passed on to <code>\$predict()</code> : <code>return_dist</code> , <code>.verbose</code> .

Value

A list with `idx` and `dist`. See [AnnIndex](#).

RnnDescentIndex	<i>Relative NN-Descent index</i>
-----------------	----------------------------------

Description

Builds and prunes a navigable graph in one pass, skipping the separate refinement step [NsgIndex](#) pays for. `r` caps the out-degree, `ef_search` is the recall knob. On Gaussian benchmark data it tops out around 0.97 recall, so check it reaches what you need.

Super class

[AnnIndex](#) -> RnnDescentIndex

Active bindings

`ef_search` Integer or NULL. Beam width at query time.
`k_search` Integer or NULL. Neighbours expanded per hop.

Methods

Public methods:

- [RnnDescentIndex\\$new\(\)](#)
- [RnnDescentIndex\\$clone\(\)](#)

`RnnDescentIndex$new()`: Build the index.

Usage:

```
RnnDescentIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine", "manhattan"),
  s = 20L,
  r = 96L,
  t1 = 4L,
  t2 = 15L,
  n_trees = NULL,
  ef_search = NULL,
  k_search = NULL,
  seed = 42L,
  precision = c("float", "double"),
  .verbose = FALSE
)
```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; [load_ann_index\(\)](#) uses that path.

`metric` String. One of `c("euclidean", "sqeuclidean", "cosine", "manhattan")`.

`s` Integer. Neighbours sampled per node per local join.

`r` Integer. Maximum out-degree after pruning.

`t1` Integer. Outer iterations.

`t2` Integer. Inner descent iterations per outer one.

`n_trees` Integer or NULL. Random projection trees for the seed graph. NULL uses $\min(5 + n^{0.25} / 2, 16)$.

`ef_search` Integer or NULL. Beam width at query time. NULL uses 100. Can be changed after the build.

`k_search` Integer or NULL. Neighbours expanded per hop. NULL uses 32, capped at `r`. Can be changed after the build.

`seed` Integer. Fixes the seed graph and the sampling.

`precision` String. "float" stores the data as f32, "double" as f64.

`.verbose` Boolean. Print build progress from Rust.

`RnnDescentIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
RnnDescentIndex$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Ono & Matsui, ACM MM, 2023

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- RnnDescentIndex$new(x)
res <- idx$predict(x[1:5, ], k = 10L)
```

SoarIndex

SOAR index

Description

IVF with spilling: every point also lands in a second cell, picked by a rule that accounts for the residual it carries in its first. Buys recall at a given nprobe over [IvfIndex](#), but scans about twice the candidates per probe, so compare the two on query time, not on nprobe. No Manhattan.

Super class

[AnnIndex](#) -> SoarIndex

Active bindings

nprobe Integer or NULL. Cells visited per query.

Methods

Public methods:

- [SoarIndex\\$new\(\)](#)
- [SoarIndex\\$clone\(\)](#)

`SoarIndex$new()`: Build the index.

Usage:

```
SoarIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine"),
  nlist = NULL,
  nprobe = NULL,
  rule = NULL,
  rule_param = NULL,
  kmeans_iters = NULL,
  kmeans_balanced = FALSE,
  seed = 42L,
  precision = c("float", "double"),
  .verbose = FALSE
)
```

Arguments:

data Numeric matrix or data.frame. Samples x features. An index pointer is also accepted; `load_ann_index()` uses that path.

metric String. One of c("euclidean", "sqeuclidean", "cosine").

nlist Integer or NULL. Number of Voronoi cells. NULL uses `sqrt(n)`.

nprobe Integer or NULL. Cells visited per query. NULL uses `sqrt(nlist)`. Can be changed after the build.

rule String or NULL. Secondary assignment rule, one of c("nearest", "shifted", "orthogonal"). NULL picks orthogonal for cosine and shifted otherwise.

rule_param Numeric or NULL. mu for the shifted rule, lambda for the orthogonal one. NULL uses 0.5 and 1.0 respectively.

kmeans_iters Integer or NULL. Lloyd iterations. NULL uses 30.

kmeans_balanced Boolean. Reseed starved centroids each iteration.

seed Integer. Fixes the k-means initialisation.

precision String. "float" stores the data as f32, "double" as f64.

.verbose Boolean. Print build progress from Rust.

`SoarIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SoarIndex$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Sun et al., NeurIPS, 2023

Examples

```
x <- generate_clustered_data(1000L, 16L)$data
idx <- SoarIndex$new(x)
res <- idx$predict(x[1:5, ], k = 10L)
```

subsample_queries	<i>Subsample queries from a dataset</i>
-------------------	---

Description

Draws rows from `x` and adds light Gaussian noise (sd 0.05). Querying an index with the rows it was built from flattens it: every query has an exact hit at distance zero. This puts the queries near the data instead of on it. The data is cast to f32 first, as in the Python bindings.

Usage

```
subsample_queries(x, n, seed = 42L)
```

Arguments

<code>x</code>	Numeric matrix, samples x features.
<code>n</code>	Integer. Number of rows to draw, capped at <code>nrow(x)</code> .
<code>seed</code>	Integer. Random seed for the draw and the noise.

Value

Numeric matrix with `min(n, nrow(x))` rows and `ncol(x)` columns.

Examples

```
dat <- generate_clustered_data(1000L, 16L)$data
query <- subsample_queries(dat, 100L)
```

VamanaIndex

Vamana index

Description

The flat graph from DiskANN: out-degree `r`, pruned in two passes with the relaxed-neighbour rule. Builds slower than [HnswIndex](#) at its cheapest setting, though the gap closes once you match on recall, and the index is a few per cent smaller.

Super class

[AnnIndex](#) -> VamanaIndex

Active bindings

`ef_search` Integer or NULL. Beam width at query time.

Methods**Public methods:**

- [VamanaIndex\\$new\(\)](#)
- [VamanaIndex\\$clone\(\)](#)

`VamanaIndex$new()`: Build the index.

Usage:

```
VamanaIndex$new(
  data,
  metric = c("euclidean", "sqeuclidean", "cosine", "manhattan"),
  r = 48L,
  l_build = 100L,
  alpha_pass1 = 1,
```

```

    alpha_pass2 = 1.2,
    ef_search = NULL,
    seed = 42L,
    precision = c("float", "double")
  )

```

Arguments:

`data` Numeric matrix or data.frame. Samples x features. An index pointer is also accepted;

`load_ann_index()` uses that path.

`metric` String. One of c("euclidean", "sqeuclidean", "cosine", "manhattan").

`r` Integer. Maximum out-degree.

`l_build` Integer. Candidate list width during the build.

`alpha_pass1` Numeric. Relaxed-neighbour factor on the first pruning pass. 1 is the plain rule.

`alpha_pass2` Numeric. Same on the second pass. Above 1 keeps longer edges, which keeps the graph navigable from far away.

`ef_search` Integer or NULL. Beam width at query time. NULL uses 75. Can be changed after the build.

`seed` Integer. Fixes the entry point and pruning order.

`precision` String. "float" stores the data as f32, "double" as f64.

`VamanaIndex$clone()`: The objects of this class are cloneable with this method.

Usage:

```
VamanaIndex$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Subramanya et al., NeurIPS, 2019

Examples

```

x <- generate_clustered_data(1000L, 16L)$data
idx <- VamanaIndex$new(x)
res <- idx$predict(x[1:5, ], k = 10L)

```

Index

`ann_get_threads`, [2](#)
`ann_set_threads`, [3](#)
`ann_set_threads()`, [8](#)
`AnnIndex`, [3](#), [5](#), [6](#), [8](#), [12](#), [13](#), [15](#), [16](#), [18–20](#), [22](#),
[24](#), [26](#), [28](#)
`AnnoyIndex`, [3](#), [5](#), [15](#)

`BallTreeIndex`, [6](#)

`ExhaustiveIndex`, [3](#), [8](#), [16](#), [18](#)

`generate_cell_embeddings`, [9](#)
`generate_clustered_data`, [9](#)
`generate_correlated_data`, [10](#)
`generate_low_rank_data`, [11](#)

`HnswIndex`, [3](#), [12](#), [28](#)

`IvfIndex`, [13](#), [26](#)

`KdTreeIndex`, [15](#)
`KmknnIndex`, [16](#)
`knn_recall`, [17](#)

`load_ann_index`, [18](#)
`load_ann_index()`, [3–5](#), [7](#), [8](#), [13–15](#), [17](#), [19](#),
[21](#), [23](#), [25](#), [27](#), [29](#)
`LshIndex`, [19](#)

`NNDescentIndex`, [20](#)
`NsgIndex`, [22](#), [24](#)

`predict.AnnIndex`, [24](#)

`RnnDescentIndex`, [24](#)

`saveRDS()`, [3](#)
`SoarIndex`, [26](#)
`subsample_queries`, [27](#)

`VamanaIndex`, [28](#)