

Package ‘RcppHNSW’

October 12, 2022

Title 'Rcpp' Bindings for 'hnsplib', a Library for Approximate Nearest Neighbors

Version 0.4.1

Description 'Hnsplib' is a C++ library for Approximate Nearest Neighbors. This package provides a minimal R interface by relying on the 'Rcpp' package. See <https://github.com/nmslib/hnsplib> for more on 'hnsplib'. 'hnsplib' is released under Version 2.0 of the Apache License.

License GPL (>= 3)

URL <https://github.com/jlmelville/rcpphnsw>

BugReports <https://github.com/jlmelville/rcpphnsw/issues>

Encoding UTF-8

Imports methods, Rcpp (>= 0.11.3)

LinkingTo Rcpp

RoxygenNote 7.2.0

Suggests testthat, covr

NeedsCompilation yes

Author James Melville [aut, cre],
Aaron Lun [ctb],
Samuel Grangeaud [ctb],
Dmitriy Selivanov [ctb],
Yuxing Liao [ctb]

Maintainer James Melville <jlmelville@gmail.com>

Repository CRAN

Date/Publication 2022-07-18 07:20:02 UTC

R topics documented:

RcppHnsw-package	2
hnsw_build	2
hnsw_knn	3
hnsw_search	5

Index

7

RcppHsw-package	<i>Rcpp bindings for the hswlib C++ library for approximate nearest neighbors.</i>
-----------------	--

Description

hswlib is a library implementing the Hierarchical Navigable Small World method for approximate nearest neighbor search.

Details

Details about hswlib are available at the reference listed below.

Author(s)

James Melville for the R interface; Yury Malkov for hswlib itself.

Maintainer: James Melville <jlmeville@gmail.com>

References

<https://github.com/nmslib/hswlib>

Malkov, Y. A., & Yashunin, D. A. (2016). Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *arXiv preprint arXiv:1603.09320*.

hsw_build	<i>Build an hswlib nearest neighbor index</i>
-----------	---

Description

Build an hswlib nearest neighbor index

Usage

```
hsw_build(
  X,
  distance = "euclidean",
  M = 16,
  ef = 200,
  verbose = FALSE,
  progress = "bar",
  n_threads = 0,
  grain_size = 1
)
```

Arguments

<code>X</code>	a numeric matrix of data to add. Each of the <code>n</code> rows is an item in the index.
<code>distance</code>	Type of distance to calculate. One of: <ul style="list-style-type: none"> • "l2" Squared L2, i.e. squared Euclidean. • "euclidean" Euclidean. • "cosine" Cosine. • "ip" Inner product: $1 - \sum(a_i * b_i)$, i.e. the cosine distance where the vectors are not normalized. This can lead to negative distances and other non-metric behavior.
<code>M</code>	Controls the number of bi-directional links created for each element during index construction. Higher values lead to better results at the expense of memory consumption. Typical values are 2 - 100, but for most datasets a range of 12 - 48 is suitable. Can't be smaller than 2.
<code>ef</code>	Size of the dynamic list used during construction. A larger value means a better quality index, but increases build time. Should be an integer value between 1 and the size of the dataset.
<code>verbose</code>	If TRUE, log messages to the console.
<code>progress</code>	If "bar" (the default), also log a progress bar when verbose = TRUE. There is a small but noticeable overhead (a few percent of run time) to tracking progress. Set progress = NULL to turn this off. Has no effect if verbose = FALSE.
<code>n_threads</code>	Maximum number of threads to use. The exact number is determined by grain_size.
<code>grain_size</code>	Minimum amount of work to do (rows in <code>X</code> to add) per thread. If the number of rows in <code>X</code> isn't sufficient, then fewer than <code>n_threads</code> will be used. This is useful in cases where the overhead of context switching with too many threads outweighs the gains due to parallelism.

Value

an instance of a HnswL2, HnswCosine or HnswIp class.

Examples

```

irism <- as.matrix(iris[, -5])
ann <- hnsw_build(irism)
iris_nn <- hnsw_search(irism, ann, k = 5)

```

hnsw_knn

Find Nearest Neighbors and Distances

Description

A k-nearest neighbor algorithm using the hnsplib library (<https://github.com/nmslib/hnswlib>).

Usage

```

hsw_knn(
  X,
  k = 10,
  distance = "euclidean",
  M = 16,
  ef_construction = 200,
  ef = 10,
  verbose = FALSE,
  progress = "bar",
  n_threads = 0,
  grain_size = 1
)

```

Arguments

X	a numeric matrix of data to search Each of the n rows is an item in the index.
k	Number of neighbors to return.
distance	Type of distance to calculate. One of: <ul style="list-style-type: none"> • "l2" Squared L2, i.e. squared Euclidean. • "euclidean" Euclidean. • "cosine" Cosine. • "ip" Inner product: $1 - \sum(a_i * b_i)$, i.e. the cosine distance where the vectors are not normalized. This can lead to negative distances and other non-metric behavior.
M	Controls the number of bi-directional links created for each element during index construction. Higher values lead to better results at the expense of memory consumption. Typical values are 2 - 100, but for most datasets a range of 12 - 48 is suitable. Can't be smaller than 2.
ef_construction	Size of the dynamic list used during construction. A larger value means a better quality index, but increases build time. Should be an integer value between 1 and the size of the dataset.
ef	Size of the dynamic list used during search. Higher values lead to improved recall at the expense of longer search time. Can take values between k and the size of the dataset and may be greater or smaller than ef_construction. Typical values are 100 - 2000.
verbose	If TRUE, log messages to the console.
progress	If "bar" (the default), also log a progress bar when verbose = TRUE. There is a small but noticeable overhead (a few percent of run time) to tracking progress. Set progress = NULL to turn this off. Has no effect if verbose = FALSE.
n_threads	Maximum number of threads to use. The exact number is determined by grain_size.
grain_size	Minimum amount of work to do (rows in X to add or search for) per thread. If the number of rows in X isn't sufficient, then fewer than n_threads will be used. This is useful in cases where the overhead of context switching with too many threads outweighs the gains due to parallelism.

Value

a list containing:

- `idx` an n by k matrix containing the nearest neighbor indices.
- `dist` an n by k matrix containing the nearest neighbor distances.

Every item in the dataset is considered to be a neighbor of itself, so the first neighbor of item i should always be i itself. If that isn't the case, then any of M , `ef_construction` and `ef` may need increasing.

Hnswlib Parameters

Some details on the parameters used for index construction and search, based on https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md:

- `M` Controls the number of bi-directional links created for each element during index construction. Higher values lead to better results at the expense of memory consumption, which is around $M * 8-10$ bytes per bytes per stored element. High intrinsic dimensionalities will require higher values of M . A range of 2 - 100 is typical, but 12 - 48 is ok for most use cases.
- `ef_construction` Size of the dynamic list used during construction. A larger value means a better quality index, but increases build time. Should be an integer value between 1 and the size of the dataset. A typical range is 100 - 2000. Beyond a certain point, increasing `ef_construction` has no effect. A sufficient value of `ef_construction` can be determined by searching with `ef = ef_construction`, and ensuring that the recall is at least 0.9.
- `ef` Size of the dynamic list used during index search. Can differ from `ef_construction` and be any value between k (the number of neighbors sought) and the number of elements in the index being searched.

References

Malkov, Y. A., & Yashunin, D. A. (2016). Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *arXiv preprint arXiv:1603.09320*.

Examples

```
iris_nn_data <- hnsw_knn(as.matrix(iris[, -5]), k = 10)
```

hnsw_search

Search an hnswlib nearest neighbor index

Description

Search an hnswlib nearest neighbor index

Usage

```
hnsw_search(
  X,
  ann,
  k,
  ef = 10,
  verbose = FALSE,
  progress = "bar",
  n_threads = 0,
  grain_size = 1
)
```

Arguments

<code>X</code>	A numeric matrix of data to search for neighbors.
<code>ann</code>	an instance of a <code>HnswL2</code> , <code>HnswCosine</code> or <code>HnswIp</code> class.
<code>k</code>	Number of neighbors to return. This can't be larger than the number of items that were added to the index <code>ann</code> . To check the size of the index, call <code>ann\$size()</code> .
<code>ef</code>	Size of the dynamic list used during search. Higher values lead to improved recall at the expense of longer search time. Can take values between <code>k</code> and the size of the dataset. Typical values are 100 - 2000.
<code>verbose</code>	If <code>TRUE</code> , log messages to the console.
<code>progress</code>	If "bar" (the default), also log a progress bar when <code>verbose = TRUE</code> . There is a small but noticeable overhead (a few percent of run time) to tracking progress. Set <code>progress = NULL</code> to turn this off. Has no effect if <code>verbose = FALSE</code> .
<code>n_threads</code>	Maximum number of threads to use. The exact number is determined by <code>grain_size</code> .
<code>grain_size</code>	Minimum amount of work to do (rows in <code>X</code> to search) per thread. If the number of rows in <code>X</code> isn't sufficient, then fewer than <code>n_threads</code> will be used. This is useful in cases where the overhead of context switching with too many threads outweighs the gains due to parallelism.

Value

a list containing:

- `idx` an `n` by `k` matrix containing the nearest neighbor indices.
- `dist` an `n` by `k` matrix containing the nearest neighbor distances.

Every item in the dataset is considered to be a neighbor of itself, so the first neighbor of item `i` should always be `i` itself. If that isn't the case, then any of `M`, `ef_construction` and `ef` may need increasing.

Examples

```
irism <- as.matrix(iris[, -5])
ann <- hnsw_build(irism)
iris_nn <- hnsw_search(irism, ann, k = 5)
```

Index

[hns_build](#), [2](#)
[hns_knn](#), [3](#)
[hns_search](#), [5](#)
[HnswCosine \(RcppHnsw-package\)](#), [2](#)
[HnswIp \(RcppHnsw-package\)](#), [2](#)
[HnswL2 \(RcppHnsw-package\)](#), [2](#)

[Rcpp_HnswCosine-class](#)
 ([RcppHnsw-package](#)), [2](#)
[Rcpp_HnswIp-class \(RcppHnsw-package\)](#), [2](#)
[Rcpp_HnswL2-class \(RcppHnsw-package\)](#), [2](#)
[RcppHnsw-package](#), [2](#)