

Package: RcppHNSW (via r-universe)

September 15, 2024

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

Version 0.6.9000

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>

Imports methods, Rcpp (>= 0.11.3)

Suggests covr, testthat

LinkingTo Rcpp

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

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

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

RemoteRef HEAD

RemoteSha 0e41d91a6cdceb66d3292c051902ed4ddab7b858

Contents

RcppHnsw-package	2
hnsw_build	2
hnsw_knn	4
hnsw_search	6
Index	8

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

Description

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

Details

Details about hnsplib are available at the reference listed below.

Author(s)

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

Maintainer: James Melville jlmelville@gmail.com

References

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

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*.

See Also

Useful links:

- <https://github.com/jlmelville/rcpphnsw>
- Report bugs at <https://github.com/jlmelville/rcpphnsw/issues>

hnsw_build	<i>Build an hnsplib nearest neighbor index</i>
------------	--

Description

Build an hnsplib nearest neighbor index

Usage

```

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

```

Arguments

X	A numeric matrix of data to search for neighbors. If <code>byrow = TRUE</code> (the default) then each row of X is an item to be searched. Otherwise, each item should be stored in the columns of X.
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	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.
verbose	If TRUE, log messages to the console.
progress	defunct and has no effect.
n_threads	Maximum number of threads to use. The exact number is determined by <code>grain_size</code> .
grain_size	Minimum amount of work to do (rows in X to add) per thread. If the number of rows in X 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.
byrow	if TRUE (the default), this indicates that the items in X to be indexed are stored in each row. Otherwise, the items are stored in the columns of X. Storing items in each column reduces the overhead of copying data to a form that can be indexed by the hnsw library.

Value

an instance of an HnswEuclidean, 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/hnsplib>).

Usage

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

Arguments

- | | |
|----------|--|
| X | A numeric matrix of n items to search for neighbors. If byrow = TRUE (the default) then each row of X stores an item to be searched. Otherwise, each item should be stored in the columns of X. |
| 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	defunct and has no effect.
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.
byrow	if TRUE (the default), this indicates that the items to be processed in X are stored in each row of X. Otherwise, the items are stored in the columns of X. Storing items in each column reduces the overhead of copying data to a form that can be used by the hnsw library. Note that if byrow = FALSE, any matrices returned from this function will also store the items by column.

Value

a list containing:

- `idx` a matrix containing the nearest neighbor indices.
- `dist` a matrix containing the nearest neighbor distances.

The dimensions of the matrices respect the storage (row or column-based) of X as indicated by the `byrow` parameter. If `byrow = TRUE` (the default) each row of `idx` and `dist` contain the neighbor information for the item passed in the equivalent row of X, i.e. the dimensions are $n \times k$ where n is the number of items in X. If `byrow = FALSE`, then each column of `idx` and `dist` contain the neighbor information for the item passed in the equivalent column of X, i.e. the dimensions are $k \times n$.

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` or `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 hnsplib nearest neighbor index

Description

Search an hnsplib nearest neighbor index

Usage

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

Arguments

<code>X</code>	A numeric matrix of data to search for neighbors. If <code>byrow = TRUE</code> (the default) then each row of <code>X</code> is an item to be searched. Otherwise, each item should be stored in the columns of <code>X</code> .
<code>ann</code>	an instance of an <code>HnswEuclidean</code> , <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>	defunct and has no effect.
<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 (items in <code>X</code> to search) per thread. If the number of items 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.
<code>byrow</code>	if <code>TRUE</code> (the default), this indicates that the items to be searched in <code>X</code> are stored in each row of <code>X</code> . Otherwise, the items are stored in the columns of <code>X</code> . Storing items in each column reduces the overhead of copying data to a form that can be searched by the <code>hns</code> library. Note that if <code>byrow = FALSE</code> , any matrices returned from this function will also store the items by column.

Value

a list containing:

- `idx` a matrix containing the nearest neighbor indices.
- `dist` a matrix containing the nearest neighbor distances.

The dimensions of the matrices respect the storage (row or column-based) of `X` as indicated by the `byrow` parameter. If `byrow = TRUE` (the default) each row of `idx` and `dist` contain the neighbor information for the item passed in the equivalent row of `X`, i.e. the dimensions are $n \times k$ where n is the number of items in `X`. If `byrow = FALSE`, then each column of `idx` and `dist` contain the neighbor information for the item passed in the equivalent column of `X`, i.e. the dimensions are $k \times n$.

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` or `ef` may need increasing.

Examples

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

Index

hnsw_build, [2](#)
hnsw_knn, [4](#)
hnsw_search, [6](#)
HnswCosine (RcppHnsw-package), [2](#)
HnswEuclidean (RcppHnsw-package), [2](#)
HnswIp (RcppHnsw-package), [2](#)
HnswL2 (RcppHnsw-package), [2](#)

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