# Package 'RcppAnnoy'

October 28, 2022

**Type** Package

**Title** 'Rcpp' Bindings for 'Annoy', a Library for Approximate Nearest Neighbors

**Version** 0.0.20

**Date** 2022-10-27

**Author** Dirk Eddelbuettel

**Maintainer** Dirk Eddelbuettel <edd@debian.org>

**Description** 'Annoy' is a small C++ library for Approximate Nearest Neighbors written for efficient memory usage as well an ability to load from / save to disk. This package provides an R interface by relying on the 'Rcpp' package, exposing the same interface as the original Python wrapper to 'Annoy'. See <https://github.com/spotify/annoy> for more on 'Annoy'. 'Annoy' is released under Version 2.0 of the Apache License. Also included is a small Windows port of 'mmap' which is released under the MIT license.

**License** GPL (>= 2)

**Depends** R (>= 3.1)

**Imports** methods, Rcpp

**LinkingTo** Rcpp

**Suggests** tinytest

**URL** https://github.com/eddelbuettel/rcppannoy, https://dirk.eddelbuettel.com/code/rcpp.annoy.html

**BugReports** https://github.com/eddelbuettel/rcppannoy/issues

**NeedsCompilation** yes

**RoxygenNote** 7.1.1

**Repository** CRAN

**Date/Publication** 2022-10-27 23:37:56 UTC

# R topics documented:



---

RcppAnnoy-package *Rcpp bindings for the Annoy C++ library for approximate nearest neighbors.*

---

## Description

Annoy is a small library written to provide fast and memory-efficient nearest neigbor lookup from a possibly static index which can be shared across processes.

## Details

Details about Annoy are available at the reference listed below.

## Author(s)

Dirk Eddelbuettel for the R interface; Erik Bernhardsson for Annoy itself.

Maintainer: Dirk Eddelbuettel <edd@debian.org>

## References

https://github.com/spotify/annoy

## Examples

```
# Optional simple examples of the most important functions
```

---

AnnoyIndex *Approximate Nearest Neighbors with Annoy*

---

## Description

Annoy is a small library written to provide fast and memory-efficient nearest neighbor lookup from a possibly static index which can be shared across processes.

## Usage

```
a <- new(AnnoyEuclidean, vectorsz)

a$setSeed(0)
a$setVerbose(0)

a$addItem(i, dv)

a$getNItems()

a$getItemsVector(i)
a$getDistance(i, j)

a$build(n_trees)

a$getNNsByItem(i, n)
a$getNNsByItemList(i, n, search_k, include_distances)

a$getNNsByVector(v, n)
a$getNNsByVectorList(v, n, search_k, include_distances)

a$save(fn)
a$load(fn)
a$unload()
```

## Details

`new(Class, vectorsz)` Create a new Annoy instance of type `Class` where `Class` is on of the following: `AnnoyEuclidean`, `AnnoyAngular`, `AnnoyManhattan`, `AnnoyHamming`. `vectorsz` denotes the length of the vectors that the Annoy instance will be indexing.

`$addItem(i, v)` Adds item `i` (any nonnegative integer) with vector `v`. Note that it will allocate memory for `max(i) + 1` items.

`$build(n_trees)` Builds a forest of `n_trees` trees. More trees gives higher precision when querying. After calling `build`, no more items can be added.

`$save(fn)` Saves the index to disk as filename `fn`. After saving, no more items can be added.

`$load(fn)` Loads (mmaps) an index from filename `fn` on disk.

`$unload()` Unloads index.

`$getDistance(i, j)` Returns the distance between items `i` and `j`

`$getNNsByItem(i, n)` Returns the `n` closest items as an integer vector of indices.

`$getNNsByVector(v, n)` Same as `$getNNsByItem`, but queries by vector `v` rather than index `i`.

`$getNNsByItemList(i, n, search_k = -1, include_distances = FALSE)` Returns the `n` closest items to item `i` as a list. During the query it will inspect up to `search_k` nodes which defaults to `n_trees * n` if not provided. `search_k` gives you a run-time tradeoff between better accuracy and speed. If you set `include_distances` to `TRUE`, it will return a length 2 list with elements `"item"` & `"distance"`. The `"item"` element contains the `n` closest items as an integer vector of indices. The optional `"distance"` element contains the corresponding distances to `"item"` as a numeric vector.

`$getNNsByVectorList(i, n, search_k = -1, include_distances = FALSE)` Same as `$getNNsByItemList`, but queries by vector `v` rather than index `i`

`$getItemsVector(i)` Returns the vector for item `i` that was previously added.

`$getNItems()` Returns the number of items in the index.

`$setVerbose()` If `1` then messages will be printed during processing. If `0` then messages will be suppressed during processing.

`$setSeed()` Set random seed for annoy (integer).

## Examples

```
library(RcppAnnoy)

# BUILDING ANNOY INDEX ------------------------------------------------------
vector_size <- 10
a <- new(AnnoyEuclidean, vector_size)

a$setSeed(42)

# Turn on verbose status messages (0 to turn off)
a$setVerbose(1)

# Load 100 random vectors into index
for (i in 1:100) a$addItem(i - 1, runif(vector_size)) # Annoy uses zero indexing

# Display number of items in index
a$getNItems()

# Retrieve item at postition 0 in index
a$getItemsVector(0)

# Calculate distance between items at postitions 0 & 1 in index
a$getDistance(0, 1)

# Build forest with 50 trees
a$build(50)


# PERFORMING ANNOY SEARCH ---------------------------------------------------

# Retrieve 5 nearest neighbors to item 0
# Returned as integer vector of indices
a$getNNsByItem(0, 5)

# Retrieve 5 nearest neighbors to item 0
# search_k = -1 will invoke default search_k value of n_trees * n
# Return results as list with an element for distance
a$getNNsByItemList(0, 5, -1, TRUE)

# Retrieve 5 nearest neighbors to item 0
# search_k = -1 will invoke default search_k value of n_trees * n
# Return results as list without an element for distance
```

```
a$getNNsByItemList(0, 5, -1, FALSE)


v <- runif(vector_size)
# Retrieve 5 nearest neighbors to vector v
# Returned as integer vector of indices
a$getNNsByVector(v, 5)

# Retrieve 5 nearest neighbors to vector v
# search_k = -1 will invoke default search_k value of n_trees * n
# Return results as list with an element for distance
a$getNNsByVectorList(v, 5, -1, TRUE)

# Retrieve 5 nearest neighbors to vector v
# search_k = -1 will invoke default search_k value of n_trees * n
# Return results as list with an element for distance
a$getNNsByVectorList(v, 5, -1, TRUE)


# SAVING/LOADING ANNOY INDEX ------------------------------------------------

# Create a tempfile, replace with a local file to keep
treefile <- tempfile(pattern="annoy", fileext="tree")

# Save annoy tree to disk
a$save(treefile)

# Load annoy tree from disk
a$load(treefile)

# Unload index from memory
a$unload()
```

---

`getAnnoyVersion` *Get the Annoy library version*

---

## Description

Get the version of the Annoy C++ library that RcppAnnoy was compiled with.

## Usage

```
getAnnoyVersion(compact = FALSE)
```

## Arguments

| | |
|---|---|
| `compact` | Logical scalar indicating whether a compact `package_version` should be returned. |

## Value

An integer vector containing the major, minor and patch version numbers; or if `compact=TRUE`, a `package_version` object.

## Author(s)

Aaron Lun

---

`getArchictectureStatus` *Report CPU Architecture and Compiler*

---

## Description

Report CPU Architecture and Compiler

## Usage

```
getArchictectureStatus()
```

## Value

A constant direct created at compile-time describing the extent of AVX instructions (512 bit, 128 bit, or none) and compiler use where currently recognised are MSC (unlikely for R), GCC, Clang, or 'other'.

# Index