Package 'BiocNeighbors'

April 15, 2019

Version 1.0.0

Date 2018-10-15

Title Nearest Neighbor Detection for Bioconductor Packages

Depends R (>= 3.5), BiocParallel

Imports Rcpp, S4Vectors, stats, methods

Suggests testthat, BiocStyle, knitr, rmarkdown, FNN, RcppAnnoy

biocViews Clustering, Classification

Description Implements exact and approximate methods for nearest neighbor detection, in a framework that allows them to be easily switched within Bioconductor packages or workflows. The exact algorithm is implemented using pre-clustering with the k-means algorithm, as described by Wang (2012). This is faster than conventional kd-trees for neighbor searching in higher (> 20) dimensional data. The approximate method uses the Annoy algorithm. Functions are also provided to search for all neighbors within a given distance. Parallelization is achieved for all methods using the BiocParallel framework.

License GPL-3

LinkingTo Rcpp, RcppAnnoy

VignetteBuilder knitr

SystemRequirements C++11

RoxygenNote 6.1.0

git_url https://git.bioconductor.org/packages/BiocNeighbors

git_branch RELEASE_3_8

git_last_commit e252fc0

git_last_commit_date 2018-10-30

Date/Publication 2019-04-15

Author Aaron Lun [aut, cre, cph]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

R topics documented:

`AnnoyIndex` *The AnnoyIndex class*

Description

A class to hold indexing structures for the Annoy algorithm for approximate nearest neighbor identification.

Usage

```
AnnoyIndex(path, dim, NAMES=NULL)

AnnoyIndex_path(x)

## S4 method for signature 'AnnoyIndex'
show(object)

## S4 method for signature 'AnnoyIndex'
dim(x)

## S4 method for signature 'AnnoyIndex'
dimnames(x)
```

Arguments

`path`	A string specifying the path to the index file.
`dim`	An integer vector of length 2, specifying the dimensions of the data used to construct the index.
`NAMES`	A character vector of sample names or `NULL`.
`x, object`	A AnnoyIndex object.

Details

The AnnoyIndex class holds the indexing structure required to run the KMKNN algorithm. Users should never need to call the constructor explicitly, but should generate instances of AnnoyIndex classes with `buildAnnoy`.

Value

The `AnnoyIndex` constructor will return an instance of the AnnoyIndex class.

`AnnoyIndex_path` will return the path to the index file.

`dim` will return the dimensions of the data set used to create the index.

`dimnames` will return a list of length 2 where the first element is the supplied `NAMES`.

Author(s)

Aaron Lun

See Also

`buildAnnoy`

Examples

```
example(buildAnnoy)

dim(out)
str(AnnoyIndex_path(out))
```

`AnnoyParam` *The AnnoyParam class*

Description

A class to hold parameters for the Annoy algorithm for approximate nearest neighbor identification.

Usage

```
AnnoyParam(ntrees=50, directory=tempdir())

AnnoyParam_ntrees(x)

AnnoyParam_directory(x)

## S4 method for signature 'AnnoyParam'
show(object)
```

Arguments

`ntrees`	Integer scalar, number of trees to use for index generation.
`directory`	String specifying the directory in which to save the index.
`x, object`	A AnnoyParam object.

Details

The AnnoyParam class holds any parameters associated with running the Annoy algorithm. Currently, this relates to building of the index - see `buildAnnoy` for details.

Value

The `AnnoyParam` constructor will return an instance of the AnnoyParam class.

`AnnoyParam_ntrees` will return the number of trees as an integer scalar.

`AnnoyParam_directory` will return the directory as a string.

Author(s)

Aaron Lun

See Also

`buildAnnoy`

Examples

```
(out <- AnnoyParam())

AnnoyParam_ntrees(out)
AnnoyParam_directory(out)
```

BiocNeighborIndex *The BiocNeighborIndex class*

Description

A virtual class for indexing structures of different nearest-neighbor search algorithms.

Details

The BiocNeighborIndex class is a virtual base class on which other index objects are built. There are 2 concrete subclasses:

`KmknnIndex`: exact nearest-neighbor search with the KMKNN algorithm.

`AnnoyIndex`: approximate nearest-neighbor search with the Annoy algorithm.

These objects hold indexing structures for a given data set - see the associated documentation pages for more details. It also retains information about the dimensionality of the input data as well as the sample names.

Methods

`show(object)`: Display the class and dimensions of a BiocNeighborIndex `object`.

`dimnames(x)`: Display the dimension names of a BiocNeighborIndex `x`.

Author(s)

Aaron Lun

See Also

KmknnIndex and AnnoyIndex for direct constructors.

buildNNIndex for construction on an actual data set.

findKNN and queryKNN for dispatch.

BiocNeighborParam *The BiocNeighborParam class*

Description

A virtual class for specifying the type of nearest-neighbor search algorithm and associated parameters.

Details

The BiocNeighborParam class is a virtual base class on which other parameter objects are built. There are 2 concrete subclasses:

KmknnParam: exact nearest-neighbor search with the KMKNN algorithm.

AnnoyParam: approximate nearest-neighbor search with the Annoy algorithm.

These objects hold parameters specifying how each algorithm should be run on an arbitrary data set. See the associated documentation pages for more details.

Methods

`show(object)`: Display the class of a BiocNeighborParam `object`.

Author(s)

Aaron Lun

See Also

KmknnParam and AnnoyParam for constructors.

buildNNIndex, findKNN and queryKNN for dispatch.

buildAnnoy *Build an Annoy index*

Description

Build an Annoy index and save it to file in preparation for a nearest-neighbors search.

Usage

```
buildAnnoy(X, ntrees=50, directory=tempdir(),
    fname=tempfile(tmpdir=directory, fileext=".idx"))
```

Arguments

X	A numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
ntrees	Integer scalar specifying the number of trees, where more trees provide greater accuracy at the cost of more computational work.
directory	String containing the path to the directory in which to save the index file.
fname	String containing the path to the index file.

Details

This function is automatically called by findAnnoy and related functions. However, it can be called directly by the user to save time if multiple queries are to be performed to the same X.

It is advisable to change directory to a location that is amenable to parallel read operations on HPC file systems. Of course, if index files are manually constructed, the user is also responsible for their clean-up after all calculations are completed.

Value

A AnnoyIndex object containing:

- path, a string containing the path to the index file.
- dim, an integer vector of length 2 containing dim(X) for later reference.
- NAMES, a character vector or NULL equal to rownames(X).

Author(s)

Aaron Lun

See Also

See AnnoyIndex for details on the output class.

See findAnnoy and queryAnnoy for dependent functions.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
out <- buildAnnoy(Y)
out
```

`buildKmknn` *Pre-cluster points with k-means*

Description

Perform k-means clustering in preparation for a KMKNN nearest-neighbors search.

Usage

```
buildKmknn(X, ...)
```

Arguments

`X`	A numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
`...`	Further arguments to pass to `kmeans`.

Details

This function is automatically called by `findKmknn` and related functions. However, it can be called directly by the user to save time if multiple queries are to be performed to the same `X`.

Value

A KmknnIndex object containing:

- `data`, a numeric matrix with points in the *columns* and dimensions in the rows, i.e., transposed relative to the input. Points have also been reordered to improve data locality during the nearest-neighbor search. Specifically, points in the same cluster are contiguous and ordered by increasing distance from the cluster center.
- `clusters`, itself a list containing:
  - `centers`, a numeric matrix of cluster center coordinates where each column corresponds to a cluster.
  - `info`, another list of length equal to the number of clusters. Each entry corresponds to a column of `centers` (let's say cluster j) and is a list of length 2. The first element is the zero-index of the first cell in the *output* `X` that is assigned to j. The second element is the distance of each point in the cluster from the cluster center.
- `order`, an integer vector specifying how rows in `X` have been reordered in columns of `data`.
- `NAMES`, a character vector or `NULL` equal to `rownames(X)`.

Author(s)

Aaron Lun

See Also

See `kmeans` for optional arguments.

See `KmknnIndex` for details on the output class.

See `findKmknn`, `queryKmknn` and `findNeighbors` for dependent functions.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
out <- buildKmknn(Y)
out
```

`buildNNIndex` *Build a nearest-neighbor index*

Description

Build indices for nearest-neighbor searching with different algorithms.

Usage

```
buildNNIndex(..., BNPARAM=NULL)
```

Arguments

`...`	Further arguments to be passed to individual methods. The mandatory argument for all methods is `X`, a numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
`BNPARAM`	A BiocNeighborParam object specifying the type of index to be constructed. This defaults to a KmknnParam object if `NULL`.

Details

Supplying a KmknnParam object as `BNPARAM` will dispatch to `buildKmknn`.

Supplying an AnnoyParam object as `BNPARAM` will dispatch to `buildAnnoy`.

Value

A BiocNeighborIndex object containing indexing structures for each algorithm.

Author(s)

Aaron Lun

See Also

`buildKmknn` and `buildAnnoy` for specific methods.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
(k.out <- buildNNIndex(Y))
(a.out <- buildNNIndex(Y, BNPARAM=AnnoyParam()))
```

findAnnoy *Find approximate nearest neighbors*

Description

Use the Annoy algorithm to identify approximate nearest neighbors from a dataset.

Usage

```
findAnnoy(X, k, get.index=TRUE, get.distance=TRUE, BPPARAM=SerialParam(),
    precomputed=NULL, subset=NULL, ...)
```

Arguments

X	A numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
k	A positive integer scalar specifying the number of nearest neighbors to retrieve.
get.index	A logical scalar indicating whether the indices of the nearest neighbors should be recorded.
get.distance	A logical scalar indicating whether distances to the nearest neighbors should be recorded.
BPPARAM	A BiocParallelParam object indicating how the search should be parallelized.
precomputed	An AnnoyIndex object from running buildAnnoy on X.
subset	A vector indicating the rows of X for which the nearest neighbors should be identified.
...	Further arguments to be passed to buildAnnoy if precomputed=NULL.

Details

This function uses the Annoy method (Approximate Nearest Neighbours Oh Yeah) by Erik Bernhardsson to identify approximate k-nearest neighbors in high-dimensional data. Briefly, a tree is constructed by using a random hyperplane to split the points at each internal node. For a given input data point, all points in the same leaf node are defined as a set of potential nearest neighbors. This is repeated across multiple trees, and the union of all such sets is searched to identify the actual nearest neighbors.

By default, nearest neighbors are identified for all data points within X. If subset is specified, nearest neighbors are only detected for the points in the subset. This yields the same result as (but is more efficient than) subsetting the output matrices after running findKNN with subset=NULL.

Turning off get.index or get.distance will not return the corresponding matrices in the output. This may provide a slight speed boost when these returned values are not of interest. Using BPPARAM will also split the search across multiple workers, which should increase speed proportionally (in theory) to the number of cores.

If the function is to be called multiple times with the same X (e.g., with different subset), it may be faster to call buildAnnoy once externally, and pass the returned object to findAnnoy via the precomputed argument. This avoids unnecessary re-indexing and can provide a substantial speed-up. Note that when precomputed is supplied, the value of X is completely ignored.

Value

A list is returned containing:

- index, if get.index=TRUE. This is an integer matrix where each row corresponds to a point (denoted here as i) in X. The row for i contains the row indices of X that are the nearest neighbors to point i, sorted by increasing distance from i.
- distance, if get.distance=TRUE. This is a numeric matrix where each row corresponds to a point (as above) and contains the sorted distances of the neighbors from i.

If subset is not NULL, each row of the above matrices refers to a point in the subset, in the same order as supplied in subset.

Author(s)

Aaron Lun

References

Bernhardsson E (2018). Annoy. https://github.com/spotify/annoy

See Also

See buildAnnoy to construct the index files ahead of time.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
out <- findAnnoy(Y, k=25)
head(out$index)
head(out$distance)
```

---

findKmknn *Find nearest neighbors*

Description

Use the KMKNN (K-means for k-nearest neighbors) algorithm to identify nearest neighbors from a dataset.

Usage

```
findKmknn(X, k, get.index=TRUE, get.distance=TRUE, BPPARAM=SerialParam(),
    precomputed=NULL, subset=NULL, raw.index=FALSE, ...)
```

Arguments

X	A numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
k	A positive integer scalar specifying the number of nearest neighbors to retrieve.
get.index	A logical scalar indicating whether the indices of the nearest neighbors should be recorded.

get.distance	A logical scalar indicating whether distances to the nearest neighbors should be recorded.
BPPARAM	A BiocParallelParam object indicating how the search should be parallelized.
precomputed	A KmknnIndex object returned from running buildKmknn on X.
subset	A vector indicating the rows of X for which the nearest neighbors should be identified.
raw.index	A logical scalar indicating whether column indices to the reordered data in precomputed should be directly returned.
...	Further arguments to pass to buildKmknn if precomputed=NULL.

Details

This function uses the method proposed by Wang (2012) to quickly identify k-nearest neighbors in high-dimensional data. Briefly, data points are rapidly clustered into N clusters using k-means clustering in buildKmknn, where N is the square root of the number of points. This clustering is then used to speed up the nearest neighbor search across X, exploiting the triangle inequality between cluster centers, the query point and each point in the cluster to narrow the search space.

By default, nearest neighbors are identified for all data points within X. If subset is specified, nearest neighbors are only detected for the points in the subset. This yields the same result as (but is more efficient than) subsetting the output matrices after running findKmknn with subset=NULL.

Turning off get.index or get.distance will not return the corresponding matrices in the output. This may provide a slight speed boost when these returned values are not of interest. Using BPPARAM will also split the search across multiple workers, which should increase speed proportionally (in theory) to the number of cores.

If the function is to be called multiple times with the same X (e.g., with different subset), it may be faster to call buildKmknn once externally, and pass the returned object to findKmknn via the precomputed argument. This avoids unnecessary repeated k-means clustering and can provide a substantial speed-up. Note that when precomputed is supplied, the value of X is completely ignored.

Currently, only Euclidean distances are supported, but support may be added for other distance types depending on demand. It remains to be seen whether the speed-up achieved with k-means is still applicable to alternative distance metrics.

Note that the code here was originally derived from an implementation in the **cydar** package (Lun *et al.*, 2017).

Value

A list is returned containing:

- index, if get.index=TRUE. This is an integer matrix where each row corresponds to a point (denoted here as i) in X. The row for i contains the row indices of X that are the nearest neighbors to point i, sorted by increasing distance from i.
- distance, if get.distance=TRUE. This is a numeric matrix where each row corresponds to a point (as above) and contains the sorted distances of the neighbors from i.

If subset is not NULL, each row of the above matrices refers to a point in the subset, in the same order as supplied in subset.

If raw.index=TRUE, the values in index refer to *columns* of KmknnIndex_clustered_data(precomputed).

Ties and random seeds

In general, this function is fully deterministic, despite the use of a stochastic `kmeans` step in `buildKmknn`. The only exception occurs when there are tied distances to neighbors, at which point the order and/or identity of the k-nearest neighboring points is not well-defined.

A warning will be raised if ties are detected among the `k+1` nearest neighbors, as this indicates that the order/identity is arbitrary. Specifically, ties are detected when a larger distance is less than `(1 + 1e-10)`-fold of the smaller distance. This criterion tends to be somewhat conservative in the sense that it will warn users even if there is no problem (i.e., the distances are truly different). However, more accurate detection is difficult to achieve due to the vagaries of numerical precision across different machines.

In the presence of ties, the output will depend on the ordering of points in the `buildKmknn` output. Users should set the seed to guarantee consistent (albeit arbitrary) results across different runs of the function. Note, however, that the exact selection of tied points depends on the numerical precision of the system. Thus, even after setting a seed, there is no guarantee that the results will be reproducible across machines (especially Windows)!

Returning raw indices

Advanced users can also set `raw.index=TRUE`, which yields results equivalent to running `findKmknn` on `t(PRE)` directly, where `PRE` is the output of `KmknnIndex_clustered_data(precomputed)`. With this setting, the indices in the output `index` matrix refer to *columns* of `PRE`. Similarly, the `subset` argument is assumed to refer to columns of `PRE`.

This setting may be more convenient when the reordered data in `precomputed` is used elsewhere, e.g., for plotting. With `raw.index=TRUE`, users avoid the need to switch between the original ordering and that from `buildKmknn`. Of course, it is also the user's responsibility to be aware of the reordering in downstream applications.

Author(s)

Aaron Lun

References

Wang X (2012). A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. *Proc Int Jt Conf Neural Netw*, 43, 6:2351-2358.

Lun ATL, Richard AC, Marioni JC (2017). Testing for differential abundance in mass cytometry data. *Nat. Methods*, 14, 7:707-709.

See Also

`buildKmknn` to build the index structure ahead of time.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
out <- findKmknn(Y, k=25)
head(out$index)
head(out$distance)
```

`findKNN` *Find k-nearest neighbors*

Description

Find the k-nearest neighbors for each point in a data set, using exact or approximate algorithms.

Usage

```
findKNN(..., BNINDEX=NULL, BNPARAM=NULL)
```

Arguments

`...`	Further arguments to pass to specific methods, including: • `X`, a numeric data matrix where rows are points and columns are dimensions. • `k`, an integer scalar for the number of nearest neighbors. This is the only strictly mandatory parameter. • `subset`, a vector specifying the subset of points in `X` to search. • `get.index`, a logical scalar indicating whether to return row indices of the neighbors. • `get.distance`, a logical scalar indicating whether to return distances to neighbors. • `BPPARAM`, a BiocParallelParam class for parallelization.
`BNINDEX`	A BiocNeighborIndex object, or `NULL`.
`BNPARAM`	A BiocNeighborParam object, or `NULL` if `BININDEX` is supplied.

Details

The class of `BNINDEX` and `BNPARAM` will determine whether dispatch is performed to `findKmknn` or `findAnnoy`. Only one of these arguments needs to be defined to resolve dispatch. However, if both are defined, they cannot specify different algorithms.

If `BNINDEX` is supplied, `X` does not need to be specified. In fact, any value of `X` will be ignored as all necessary information for the search is already present in `BNINDEX`.

Value

A list is returned containing:

- `index`, if `get.index=TRUE`. This is an integer matrix where each row corresponds to a point (denoted here as i) in `X`. The row for i contains the row indices of `X` that are the nearest neighbors to point i, sorted by increasing distance from i.
- `distance`, if `get.distance=TRUE`. This is a numeric matrix where each row corresponds to a point (as above) and contains the sorted distances of the neighbors from i.

If `subset` is not `NULL`, each row of the above matrices refers to a point in the subset, in the same order as supplied in `subset`.

Author(s)

Aaron Lun

See Also

findKmknn and findAnnoy for specific methods.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
str(k.out <- findKNN(Y, k=10))
str(a.out <- findKNN(Y, k=10, BNPARAM=AnnoyParam()))

k.dex <- buildKmknn(Y)
str(k.out2 <- findKNN(Y, k=10, BNINDEX=k.dex, BNPARAM=NULL))
str(k.out3 <- findKNN(Y, k=10, BNINDEX=k.dex, BNPARAM=KmknnParam()))

a.dex <- buildAnnoy(Y)
str(a.out2 <- findKNN(Y, k=10, BNINDEX=a.dex, BNPARAM=NULL))
str(a.out3 <- findKNN(Y, k=10, BNINDEX=a.dex, BNPARAM=AnnoyParam()))
```

---

findNeighbors *Find all neighbors*

Description

Find all neighboring data points within a certain distance with the KMKNN algorithm.

Usage

```
findNeighbors(X, threshold, get.index=TRUE, get.distance=TRUE,
    BPPARAM=SerialParam(), precomputed=NULL, subset=NULL,
    raw.index=FALSE, ...)
```

Arguments

X	A numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
threshold	A positive numeric scalar specifying the maximum distance at which a point is considered a neighbor.
get.index	A logical scalar indicating whether the indices of the neighbors should be recorded.
get.distance	A logical scalar indicating whether distances to the neighbors should be recorded.
BPPARAM	A BiocParallelParam object indicating how the search should be parallelized.
precomputed	A KmknnIndex object from running buildKmknn on X.
subset	A vector indicating the rows of X for which the neighbors should be identified.
raw.index	A logial scalar indicating whether raw column indices to precomputed$data should be returned.
...	Further arguments to pass to buildKmknn if precomputed=NULL.

Details

This function uses the same algorithm described in findKmknn to identify all points in X that within threshold of each point in X. For Euclidean distances, this is equivalent to identifying all points in a hypersphere centered around the point of interest.

By default, a search is performed for each data point in X, but it can be limited to a specified subset of points with subset. This yields the same result as (but is more efficient than) subsetting the output matrices after running findNeighbors with subset=NULL.

Turning off get.index or get.distance may provide a slight speed boost when these returned values are not of interest. Using BPPARAM will also split the search by query points, which usually provides a linear increase in speed.

If multiple queries are to be performed to the same X, it may be beneficial to use buildKmknn directly and pass the result to precomputed. In such cases, it is also possible to set raw.index=TRUE to obtain indices of neighbors in the reordered data set in precomputed, though this will change both the nature of the output index and the interpretation of subset - see ?findKmknn for details.

Value

A list is returned containing:

- index, if get.index=TRUE. This is a list of integer vectors where each entry corresponds to a point (denoted here as i) in X. The vector for i contains the set of row indices of all points in X that lie within threshold of point i. Points in each vector are not ordered, and i will always be included in its own set.
- distance, if get.distance=TRUE. This is a list of numeric vectors where each entry corresponds to a point (as above) and contains the distances of the neighbors from i. Elements of each vector in distance match to elements of the corresponding vector in index.

If subset is not NULL, each row of the above matrices refers to a point in the subset, in the same order as supplied in subset.

If raw.index=TRUE, the values in index refer to *columns* of KmknnIndex_clustered_data(precomputed).

Author(s)

Aaron Lun

See Also

buildKmknn to build an index ahead of time.

Examples

```
Y <- matrix(runif(100000), ncol=20)
out <- findNeighbors(Y, threshold=1)
```

KmknnIndex *The KmknnIndex class*

Description

A class to hold indexing structures for the KMKNN algorithm for exact nearest neighbor identification.

Usage

```
KmknnIndex(data, centers, info, order, NAMES=NULL)

KmknnIndex_clustered_data(x)

KmknnIndex_cluster_centers(x)

KmknnIndex_cluster_info(x)

KmknnIndex_clustered_order(x)

## S4 method for signature 'KmknnIndex'
show(object)

## S4 method for signature 'KmknnIndex'
dim(x)

## S4 method for signature 'KmknnIndex'
dimnames(x)
```

Arguments

data	A numeric matrix with data points in columns and dimensions in rows.
centers	A numeric matrix with clusters in columns and dimensions in rows.
info	A list of statistics for each cluster.
order	An integer vector of length equal to ncol(data), specifying the order of observations.
NAMES	A character vector of sample names or NULL.
x, object	A KmknnIndex object.

Details

The KmknnIndex class holds the indexing structure required to run the KMKNN algorithm. Users should never need to call the constructor explicitly, but should generate instances of KmknnIndex classes with buildKmknn.

Value

The KmknnIndex constructor will return an instance of the KmknnIndex class.

KmknnIndex_clustered_data and related getters will return the corresponding slots of object.

dim will return the dimensions of t(data).

dimnames will return a list of length 2 where the first element is the supplied NAMES.

Author(s)

Aaron Lun

See Also

buildKmknn

Examples

```
example(buildKmknn)

dim(out)

str(KmknnIndex_clustered_data(out))
str(KmknnIndex_cluster_centers(out))
str(KmknnIndex_clustered_order(out))
```

KmknnParam *The KmknnParam class*

Description

A class to hold parameters for the KMKNN algorithm for exact nearest neighbor identification.

Usage

```
KmknnParam(...)

KmknnParam_kmeans_args(x)

## S4 method for signature 'KmknnParam'
show(object)
```

Arguments

`...`	Arguments to be passed to kmeans.
`x, object`	A KmknnParam object.

Details

The KmknnParam class holds any parameters associated with running the KMKNN algorithm. Currently, this relates to tuning of the k-means step - see buildKmknn for details.

Value

The `KmknnParam` constructor will return an instance of the KmknnParam class.

The `KmknnParam_kmeans_args` function will return a list of named arguments, used in `...` to construct `object`.

Author(s)

Aaron Lun

See Also

buildKmknn

Examples

```
(out <- KmknnParam(iter.max=100))

KmknnParam_kmeans_args(out)
```

---

queryAnnoy *Query nearest neighbors*

Description

Use the Annoy algorithm to query a dataset for nearest neighbors of points in another dataset.

Usage

```
queryAnnoy(X, query, k, get.index=TRUE, get.distance=TRUE, BPPARAM=SerialParam(),
    precomputed=NULL, transposed=FALSE, subset=NULL, ...)
```

Arguments

X	A numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
query	A numeric matrix of query points, containing different data points in the rows but the same number and ordering of dimensions in the columns.
k	A positive integer scalar specifying the number of nearest neighbors to retrieve.
get.index	A logical scalar indicating whether the indices of the nearest neighbors should be recorded.
get.distance	A logical scalar indicating whether distances to the nearest neighbors should be recorded.
BPPARAM	A BiocParallelParam object indicating how the search should be parallelized.
precomputed	An AnnoyIndex object from running buildAnnoy on X.
transposed	A logical scalar indicating whether the query is transposed, in which case query is assumed to contain dimensions in the rows and data points in the columns.
subset	A vector indicating the rows of query (or columns, if transposed=TRUE) for which the nearest neighbors should be identified.
...	Further arguments to be passed to buildAnnoy if precomputed=NULL.

Details

This function uses the same algorithm described in findAnnoy to identify points in X that are nearest neighbors of each point in query. This requires both X and query to have the same number of dimensions. Moreover, the upper bound for k is set at the number of points in X.

By default, nearest neighbors are identified for all data points within query. If subset is specified, nearest neighbors are only detected for the query points in the subset. This yields the same result as

(but is more efficient than) subsetting the output matrices after running queryKNN on the full query (i.e., with subset=NULL).

If transposed=TRUE, this function assumes that query is already transposed, which saves a bit of time by avoiding an unnecessary transposition. Turning off get.index or get.distance may also provide a slight speed boost when these returned values are not of interest. Using BPPARAM will also split the search by query points across multiple processes.

If multiple queries are to be performed to the same X, it may be beneficial to use buildAnnoy directly to precompute the clustering. Note that when precomputed is supplied, the value of X is ignored.

Value

A list is returned containing:

- index, if get.index=TRUE. This is an integer matrix where each row corresponds to a point (denoted here as i) in query. The row for i contains the row indices of X that are the nearest neighbors to point i, sorted by increasing distance from i.
- distance, if get.distance=TRUE. This is a numeric matrix where each row corresponds to a point (as above) and contains the sorted distances of the neighbors from i.

If subset is not NULL, each row of the above matrices refers to a point in the subset, in the same order as supplied in subset.

Author(s)

Aaron Lun

See Also

buildAnnoy to build the index beforehand.

findAnnoy to identify k-NNs within X.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
Z <- matrix(rnorm(20000), ncol=20)
out <- queryAnnoy(Y, query=Z, k=25)
head(out$index)
head(out$distance)
```

---

queryKmknn *Query nearest neighbors*

Description

Use the KMKNN algorithm to query a dataset for nearest neighbors of points in another dataset.

Usage

```
queryKmknn(X, query, k, get.index=TRUE, get.distance=TRUE, BPPARAM=SerialParam(),
    precomputed=NULL, transposed=FALSE, subset=NULL, raw.index=FALSE, ...)
```

Arguments

X	A numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
query	A numeric matrix of query points, containing different data points in the rows but the same number and ordering of dimensions in the columns.
k	A positive integer scalar specifying the number of nearest neighbors to retrieve.
get.index	A logical scalar indicating whether the indices of the nearest neighbors should be recorded.
get.distance	A logical scalar indicating whether distances to the nearest neighbors should be recorded.
BPPARAM	A BiocParallelParam object indicating how the search should be parallelized.
precomputed	A KmknnIndex object from running buildKmknn on X.
transposed	A logical scalar indicating whether the query is transposed, in which case query is assumed to contain dimensions in the rows and data points in the columns.
subset	A vector indicating the rows of query (or columns, if transposed=TRUE) for which the nearest neighbors should be identified.
raw.index	A logical scalar indicating whether column indices to the reordered data in precomputed should be directly returned.
...	Further arguments to pass to buildKmknn if precomputed=NULL.

Details

This function uses the same algorithm described in findKmknn to identify points in X that are nearest neighbors of each point in query. This requires both X and query to have the same number of dimensions. Moreover, the upper bound for k is set at the number of points in X.

By default, nearest neighbors are identified for all data points within query. If subset is specified, nearest neighbors are only detected for the query points in the subset. This yields the same result as (but is more efficient than) subsetting the output matrices after running queryKmknn on the full query (i.e., with subset=NULL).

If transposed=TRUE, this function assumes that query is already transposed, which saves a bit of time by avoiding an unnecessary transposition. Turning off get.index or get.distance may also provide a slight speed boost when these returned values are not of interest. Using BPPARAM will also split the search by query points across multiple processes.

If multiple queries are to be performed to the same X, it may be beneficial to use buildKmknn directly to precompute the clustering. Note that when precomputed is supplied, the value of X is ignored. Advanced users can also set raw.index=TRUE, which returns indices of neighbors in the reordered data set in precomputed. This may be useful when dealing with multiple queries to a common precomputed object.

See comments in ?findKmknn regarding the warnings when tied distances are observed.

Value

A list is returned containing:

- index, if get.index=TRUE. This is an integer matrix where each row corresponds to a point (denoted here as i) in query. The row for i contains the row indices of X that are the nearest neighbors to point i, sorted by increasing distance from i.

- distance, if get.distance=TRUE. This is a numeric matrix where each row corresponds to a point (as above) and contains the sorted distances of the neighbors from i.

If subset is not NULL, each row of the above matrices refers to a point in the subset, in the same order as supplied in subset.

If raw.index=TRUE, the values in index refer to *columns* of KmknnIndex_clustered_data(precomputed).

Author(s)

Aaron Lun

See Also

buildKmknn, findKmknn

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
Z <- matrix(rnorm(20000), ncol=20)
out <- queryKmknn(Y, query=Z, k=25)
head(out$index)
head(out$distance)
```

---

queryKNN *Query k-nearest neighbors*

Description

Find the k-nearest neighbors in one data set for each point in another query data set, using exact or approximate algorithms.

Usage

```
queryKNN(..., BNINDEX=NULL, BNPARAM=NULL)
```

Arguments

...	Further arguments to pass to specific methods, including: • X, a numeric data matrix where rows are points and columns are dimensions. • query, a numeric query matrix where rows are points and columns are dimensions. • k, an integer scalar for the number of nearest neighbors. • subset, a vector specifying the subset of points in X to search. • get.index, a logical scalar indicating whether to return row indices of the neighbors. • get.distance, a logical scalar indicating whether to return distances to neighbors. • BPPARAM, a BiocParallelParam object for parallelization. • transposed, a logical scalar indicating whether query is transposed, i.e., with columns as points. Of these, only query and k are strictly mandatory.
BNINDEX	A BiocNeighborIndex object, or NULL.
BNPARAM	A BiocNeighborParam object, or NULL if BININDEX is supplied.

Details

The class of BNINDEX and BNPARAM will determine whether dispatch is performed to queryKmknn or queryAnnoy. Only one of these arguments needs to be defined to resolve dispatch. However, if both are defined, they cannot specify different algorithms.

If BNINDEX is supplied, X does not need to be specified. In fact, any value of X will be ignored as all necessary information for the search is already present in BNINDEX.

Value

A list is returned containing:

- index, if get.index=TRUE. This is an integer matrix where each row corresponds to a point (denoted here as i) in query. The row for i contains the row indices of X that are the nearest neighbors to point i, sorted by increasing distance from i.
- distance, if get.distance=TRUE. This is a numeric matrix where each row corresponds to a point (as above) and contains the sorted distances of the neighbors from i.

If subset is not NULL, each row of the above matrices refers to a point in the subset, in the same order as supplied in subset.

Author(s)

Aaron Lun

See Also

queryKmknn and queryAnnoy for specific methods.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
Z <- matrix(rnorm(10000), ncol=20)
str(k.out <- queryKNN(Y, Z, k=10))
str(a.out <- queryKNN(Y, Z, k=10, BNPARAM=AnnoyParam()))

k.dex <- buildKmknn(Y)
str(k.out2 <- queryKNN(Y,Z,  k=10, BNINDEX=k.dex, BNPARAM=NULL))
str(k.out3 <- queryKNN(Y,Z,  k=10, BNINDEX=k.dex, BNPARAM=KmknnParam()))

a.dex <- buildAnnoy(Y)
str(a.out2 <- queryKNN(Y,Z,  k=10, BNINDEX=a.dex, BNPARAM=NULL))
str(a.out3 <- queryKNN(Y,Z,  k=10, BNINDEX=a.dex, BNPARAM=AnnoyParam()))
```

---

queryNeighbors *Query neighbors*

Description

Find all neighboring data points within a certain distance of a query point with the KMKNN algorithm.

Usage

```
queryNeighbors(X, query, threshold, get.index=TRUE, get.distance=TRUE,
    BPPARAM=SerialParam(), precomputed=NULL, transposed=FALSE, subset=NULL,
    raw.index=FALSE, ...)
```

Arguments

X	A numeric matrix where rows correspond to data points and columns correspond to variables (i.e., dimensions).
query	A numeric matrix of query points, containing different data points in the rows but the same number and ordering of dimensions in the columns.
threshold	A positive numeric scalar specifying the maximum distance at which a point is considered a neighbor.
get.index	A logical scalar indicating whether the indices of the neighbors should be recorded.
get.distance	A logical scalar indicating whether distances to the neighbors should be recorded.
BPPARAM	A BiocParallelParam object indicating how the search should be parallelized.
precomputed	A KmknnIndex object from running buildKmknn on X.
transposed	A logical scalar indicating whether the query is transposed, in which case query is assumed to contain dimensions in the rows and data points in the columns.
subset	A vector indicating the rows of query (or columns, if transposed=TRUE) for which the neighbors should be identified.
raw.index	A logial scalar indicating whether raw column indices to precomputed$data should be returned.
...	Further arguments to pass to buildKmknn if precomputed=NULL.

Details

This function uses the same algorithm described in findKmknn to identify points in X that are neighbors (i.e., within a distance threshold) of each point in query. This requires both X and query to have the same number of dimensions.

By default, neighbors are identified for all data points within query. If subset is specified, neighbors are only detected for the query points in the subset. This yields the same result as (but is more efficient than) subsetting the output matrices after running queryNeighbors on the full query (i.e., with subset=NULL).

If transposed=TRUE, this function assumes that query is already transposed, which saves a bit of time by avoiding an unnecessary transposition. Turning off get.index or get.distance may also provide a slight speed boost when these returned values are not of interest. Using BPPARAM will also split the search by query points across multiple processes.

If multiple queries are to be performed to the same X, it may be beneficial to use buildKmknn directly to precompute the clustering. Advanced users can also set raw.index=TRUE, which returns indices of neighbors in the reordered data set in precomputed. This may be useful when dealing with multiple queries to a common precomputed object.

Value

A list is returned containing:

- index, if get.index=TRUE. This is a list of integer vectors where each entry corresponds to a point (denoted here as i) in query. The vector for i contains the set of row indices of all points in X that lie within threshold of point i. Points in each vector are not ordered, and i will always be included in its own set.
- distance, if get.distance=TRUE. This is a list of numeric vectors where each entry corresponds to a point (as above) and contains the distances of the neighbors from i. Elements of each vector in distance match to elements of the corresponding vector in index.

If subset is not NULL, each row of the above matrices refers to a point in the subset, in the same order as supplied in subset.

If raw.index=TRUE, the values in index refer to *columns* of KmknnIndex_clustered_data(precomputed).

Author(s)

Aaron Lun

See Also

buildKmknn to build an index ahead of time.

Examples

```
Y <- matrix(rnorm(100000), ncol=20)
Z <- matrix(rnorm(20000), ncol=20)
out <- queryNeighbors(Y, query=Z, threshold=1)
head(out$index)
head(out$distance)
```

Index