Package 'eiR'

April 5, 2014

Type Package

Title Accelerated similarity searching of small molecules

Version 1.2.0

Date 2013-10-9

Author Kevin Horan

Maintainer Kevin Horan <khoran@cs.ucr.edu>

Suggests RCurl,snow,BiocStyle

Description The eiR package provides utilities for accelerated structure similarity searching of very large small molecule data sets using an embedding and indexing approach.

License Artistic-2.0

System Requirements GSL (>=1.14) http://www.gnu.org/software/gsl/

Depends R (>= 2.10.0), ChemmineR (>= 2.13.8), methods, DBI

biocViews Infrastructure, DataImport, Clustering, Bioinformatics,Proteomics

Imports snow, tools, snowfall, RUnit, methods,ChemmineR,RCurl,digest,BiocGenerics

R topics documented:

addTransform *Add Transform*

Description

New descriptor types can be added using the addTransform function. These transforms are basically just ways to read descriptors from compound definitions, and to convert descriptors between string and object form. This conversion is required because descriptors are stored as strings in the SQL database, but are used by the rest of the program as objects.

There are two main components that need to be added. The addTransform function takes the name of the transform and two functions, toString, and toObject. These have slightly different meanings depending on the component you are adding. The first component to add is a transform from a chemical compound format, such as SDF, to a descriptor format, such as atom pair (AP), in either string or object form. The toString function should take any kind of chemical compound source, such an SDF file, an SDF object or an SDFset, and output a string representation of the descriptors. Since this function can be written in terms of other functions that will be defined, you can usually accept the default value of this function. The toObject function should take the same kind of input, but output the descriptors as an object. The actual return value is a list containing the names of the compounds (in the names field), and the actual descriptor objects (in the descriptors field).

The second component to add is a transform that converts between string and object representations of descriptors. In this case the toString function takes descriptors in object form and returns a string representation for each. The toObject function performs the inverse operation. It takes descriptors in string form and returns them as objects. The objects returned by this function will be exactly what is handed to the distance function, so you need to make sure that the two match each other.

Usage

```
addTransform(descriptorType, compoundFormat = NULL, toString = NULL, toObject)
```

Arguments

descriptorType	The name of the type of the descritor being added.
compoundFormat	The format of the compound data the descriptor will be extracted from.
toString	A function with three arguments, the data, an SQL connection object, and a directory name. The last two are optional and can be set to a default value of NULL if not used in the body of the function. If this parameter is NULL and compoundFormat is not NULL, then a default function will be used for this value.
toObject	A function with three arguments, the data, an SQL connection object, and a directory name. The last two are optional and can be set to a default value of NULL if not used in the body of the function. If compoundFormat is not NULL, then the return value of this function should be a list with the fields "names" and "descriptors", containing the compound names and descriptor objects, respectivly. If compoundFormat is NULL, then the return value should be a collection

of descriptor objects, in whatever format the distance function for this descrptor type requires.

Value

No value returned.

Author(s)

Kevin Horan

See Also

setDefaultDistance

Examples

```
# adding support for atompair (ap) descriptors extracted from
# sdf formmatted data.

    #first component
    addTransform("ap-example","sdf-example",
        # Any sdf source -> APset
        toObject = function(input,conn=NULL,dir="."){
            sdfset=if(is.character(input) && file.exists(input)){
                read.SDFset(input)
            }else if(inherits(input,"SDFset")){
                input
            }else{
           stop(paste("unknown type for input, or filename does not exist. type found:",class(input)))
            }
            list(names=sdfid(sdfset),descriptors=sdf2ap(sdfset))
        }
    )

    #second component
    addTransform("ap-example",
        # APset -> string,
        toString = function(apset,conn=NULL,dir="."){
            unlist(lapply(ap(apset), function(x) paste(x,collapse=", ")))
        },
        # string or list -> AP set list
        toObject= function(v,conn=NULL,dir="."){
            if(inherits(v,"list") || length(v)==0)
                return(v)

            as( if(!inherits(v,"APset")){
                    names(v)=as.character(1:length(v));
                    read.AP(v,type="ap",isFile=FALSE)
                } else v,
                "list")
        }
```

```
)
```

`eiAdd` *Add new compounds*

Description

Add additional compounds to and existing database

Usage

```
eiAdd(r,d,refIddb,additions,dir=".",format="sdf",conn=defaultConn(dir),
descriptorType="ap",distance=getDefaultDist(descriptorType),
updateByName = FALSE)
```

Arguments

`r`	The number of references used to build the database you wish to query agains.
`d`	The number of dimensions used to build the database you wish to query agains.
`refIddb`	An Iddb formatted file containing the reference IDs of the database you wish to append to. This should almost always be the file: run-r-d/<long random string>.cdb. The `refIddb` value should also be returned by eiMakeDb.
`additions`	The compounds to add. This can be either a file in sdf format, or and SDFset object.
`dir`	The directory where the "data" directory lives. Defaults to the current directory.
`format`	The format of the data given in `additions`. Currently only "sdf" is supported.
`conn`	Database connection to use.
`descriptorType`	The format of the descriptor. Currently supported values are "ap" for atom pair, and "fp" for fingerprint.
`distance`	The distance function to be used to compute the distance between two descriptors. A default function is provided for "ap" and "fp" descriptors.
`updateByName`	If true we make the assumption that all compounds, both in the existing database and the given dataset, have unique names. This function will then avoid re-adding existing, identical compounds, and will update existing compounds with a new definition if a new compound definition with an existing name is given. If false, we allow duplicate compound names to exist in the database, though not duplicate definitions. So identical compounds will not be re-added, but if a new version of an existing compound is added it will not update the existing one, it will add the modified one as a completely new compound with a new compound id.

Details

New Compounds can be added to an existing database, however, the reference compounds cannot be changed. This will also update the matrix file in the run/job directory with the new compounds.

Author(s)

Kevin Horan

See Also

eiMakeDb eiPerformanceTest eiQuery

Examples

```
library(snow)
r<- 50
d<- 40

#initialize
data(sdfsample)
dir=file.path(tempdir(),"add")
dir.create(dir)
eiInit(sdfsample[1:99],dir=dir)

#create compound db
refIddb=eiMakeDb(r,d,numSamples=20,dir=dir)


#find compounds similar two each query
eiAdd(r,d,refIddb,sdfsample[100],dir=dir)
```

---

eiCluster *Cluster compounds*

Description

Uses Jarvis-Patrick clustering to cluster the compound database using the LSH algorithm to quickly find nearest neighbors.

Usage

```
eiCluster(r,d,K,minNbrs, dir=".",cutoff=NULL,
 descriptorType="ap",distance=getDefaultDist(descriptorType),
 conn=defaultConn(dir), W = 1.39564, M=19,L=10,T=30,type="cluster",linkage="single")
```

Arguments

r	The number of references used to build the database you wish to query against.
d	The number of dimensions used to build the database you wish to query against.
K	The number of neighbors to consider for each compound.

minNbrs	The minimum number of neighbors that two comopunds must have in common in order to be joined.
dir	The directory where the "data" directory lives. Defaults to the current directory.
descriptorType	The format of the descriptor. Currently supported values are "ap" for atom pair, and "fp" for fingerprint.
distance	The distance function to be used to compute the distance between two descriptors. A default function is provided for "ap" and "fp" descriptors.
cutoff	Distance cutoff value. Compounds having a distance larger this this value will not be included in the nearest neighbor table. Note that this is a distance value, not a similarity value, as is often used in other ChemmineR functions.
conn	Database connection to use.
W	Tunable LSH parameter. See LSHKIT page for details. http://lshkit.sourceforge.net/dd/d2a/mplsh-tune_8cpp.html
M	Tunable LSH parameter. See LSHKIT page for details. http://lshkit.sourceforge.net/dd/d2a/mplsh-tune_8cpp.html
L	Number of hash tables
T	Number of probes
type	If "cluster", returns a clustering, else, if "matrix", returns a list in the format expected by the jarvisPatrick function in ChemmineR. This list contains the nearest neighbor matrix along with the similarity matrix. This allows one to quickly try different cutoff values without having to re-compute the whole similarity matrix each time. Note that since we are returning similarity values here instead of distance values, this will only work if the given distance function returns a value between 0 and 1. This is true of the default funtions.
linkage	Can be one of "single", "average", or "complete", for single linkage, average linkage and complete linkage merge requirements, respectively. In the context of Jarvis-Patrick, average linkage means that at least half of the pairs between the clusters under consideration must pass the merge requirement. Similarly, for complete linkage, all pairs must pass the merge requirement. Single linkage is the normal case for Jarvis-Patrick and just means that at least one pair must meet the requirement.

Details

The jarvis patrick clustering algorithm takes a set of items, a distance function, and two parameters, K, and minNbrs. For each item, it find the K nearest neighbors of that item. Normally this requires computing the distance between every pair of items. However, using Locality Sensative Hashing (LSH), the set of nearst neighbors can be found in near constant time. Once the nearest neighbor matrix is computed, the algorithm makes one pass through the items and merges all pairs that have at least minNbrs neighbors in common.

Although not required, it is avisable to specify a cutoff value. This is the maximum distance two items can have from each other and still be considered to be neighbors. It is thus possible for an item to end up with less than K neighbors if less than K items are close enough to it. If a cutoff is not specified, it is possible for highly un-related items to be listed as neighbors of another item simply

because nothing else was nearby. This can lead to items being joined into clusters with which they have no true connection.

The `type` parameter can be used to return a list which can be used to call the `jarvisPatrick` function in ChemmineR directly. The advantage of this is that it will contain the similarity matrix which can then be used to quickly set different cutoff values (using `trimNeighbors`) whithout having to re-compute the similarity matrix. Note that this requires that the given distance function return a value between 0 and 1 so it can be converted to a similarity function.

Value

If `type` is "cluster", returns a clustering. This will be a vector in which the names are the compound names, and the values are the cluster labels. Otherwise, if `type` is "matrix", returns a list with the following components:

`indexes`	index values of nearest neighbors, for each item.
`names`	The database compound id of each item in the set.
`similarities`	The similarity values of each neighbor to the item for that row. Each similarity values corresponds to the id number in the same position in the indexes entry

If there are not `K` neibhbors for a compound, that row will be padded with NAs.

Author(s)

Kevin Horan

Examples

```
library(snow)
   r<- 50
   d<- 40

   #initialize
   data(sdfsample)
   dir=file.path(tempdir(),"cluster")
   dir.create(dir)
   eiInit(sdfsample,dir=dir)

   #create compound db
   eiMakeDb(r,d,numSamples=20,dir=dir, cl=makeCluster(1,type="SOCK",outfile=""))

eiCluster(r,d,K=5,minNbrs=2,cutoff=0.5,dir=dir)
```

`eiInit` *Initialize a compound database*

Description

Takes the raw compound database in whatever format the given measure supports and creates a "data" directory.

Usage

```
eiInit(inputs,dir=".",format="sdf",descriptorType="ap",append=FALSE,
conn=defaultConn(dir,create=TRUE), updateByName = FALSE, cl = NULL, connSource = NULL)
```

Arguments

`inputs`	Either a filename of a file in `format` format, or an SDFset. This can also be a vector of filenames and if `cl` is also specified and if you database supports it (SQLite does not), it will load these file in parallel on the cluster.
`dir`	The directory where the "data" directory lives. Defaults to the current directory.
`format`	The format of the data in `inputs`. Currenly only "sdf" and "smiles" is supported.
`descriptorType`	The format of the descriptor. Currently supported values are "ap" for atom pair, and "fp" for fingerprint.
`append`	If true the given compounds will be added to an existing database and the <data-dir>/Main.iddb file will be udpated with the new compound id numbers. This should not normally be used directly, use `eiAdd` instead to add new compounds to a database.
`conn`	Database connection to use.
`updateByName`	If true we make the assumption that all compounds, both in the existing database and the given dataset, have unique names. This function will then avoid re-adding existing, identical compounds, and will update existing compounds with a new definition if a new compound definition with an existing name is given. If false, we allow duplicate compound names to exist in the database, though not duplicate definitions. So identical compounds will not be re-added, but if a new version of an existing compound is added it will not update the existing one, it will add the modified one as a completely new compound with a new compound id.
`cl`	A SNOW cluster can be given here to run this function in parrallel.
`connSource`	A function returning a new database connection. Note that it is not suffient to return a reference to an existing connection, it must be a distinct, new connection. This is needed for cluster operations that make use of the database as they will each need to craete a new connection. If not given, certain parts of this function will not be parrallelized. This function can also be used to setup the envrionment on the cluster worker nodes. For example, you might need to re-load libraries like RSQLite and such.

Details

EiInit can take either an SDFset, or a filename. SDF and SMILES is supported by default. It might complain if your SDF file does not follow the SDF specification. If this happens, you can create an SDFset with the `read.SDFset` command and then use that instead of the filename.

EiInit will create a folder called 'data'. Commands should always be executed in the folder containing this directory (ie, the parent directory of "data"), or else specify the location of that directory with the `dir` option.

Value

A directory called "data" will have been created in the current working directory. The generated compound ids of the given compounds will be returned. These can be used to reference a compound or set of compounds in other functions, such as `eiQuery`.

Author(s)

Kevin Horan

See Also

`eiMakeDb` `eiPerformanceTest` `eiQuery`

Examples

```
data(sdfsample)
dir=file.path(tempdir(),"init")
dir.create(dir)
eiInit(sdfsample,dir=dir)
```

`eiMakeDb` *Create an embedded database*

Description

Uses the initalized compound data to create an embedded compound databbase with `r` reference compounds in `d` dimensions.

Usage

```
eiMakeDb(refs,d,descriptorType="ap",distance=getDefaultDist(descriptorType),
dir=".",numSamples=cdbSize(dir)*0.1,conn=defaultConn(dir),
cl=makeCluster(1,type="SOCK",outfile=""),connSource=NULL)
```

Arguments

refs	The reference compounds to use to build the database you wish to query against. Refs can be one of three things. It can be a filename of an iddb file giving the index values of the reference compounds to use, it can be vector of index values, or it can be a scalar value giving the number of randomly selected references to use.
d	The number of dimensions used to build the database you wish to query against.
descriptorType	The format of the descriptor. Currently supported values are "ap" for atom pair, and "fp" for fingerprint.
distance	The distance function to be used to compute the distance between two descriptors. A default function is provided for "ap" and "fp" descriptors.
dir	The directory where the "data" directory lives. Defaults to the current directory.
numSamples	The number of non-reference samples to be chosen now to be used later by the eiPerformanceTest function.
conn	Database connection to use.
cl	A SNOW cluster can be given here to run this function in parrallel.
connSource	A function returning a new database connection. Note that it is not suffient to return a reference to an existing connection, it must be a distinct, new connection. This is needed for cluster operations that make use of the database as they will each need to craete a new connection. If not given, certain parts of this function will not be parrallelized. This function can also be used to setup the envrionment on the cluster worker nodes. For example, you might need to re-load libraries like RSQLite and such.

Details

This function will embedd compounds from the data directory in another space which allows for more efficient searching. The main two parameters are r and d. r is the number of reference compounds to use and d is the dimension of the embedding space. We have found in practice that setting d to around 100 works well. r should be large enough to "represent" the full compound database. Note that an r by r matrix will be constructed during the course of execution, so r should be less than about 46,000 to avoid overflowing an integer. Since this is the longest running step, a SNOW cluster can be provided to parallelize the task.

To help tune these values, eiMakeDb will pick numSamples non-reference samples which can later be used by the eiPerformanceTest function.

eiMakdDb does its job in a job folder, named after the number of reference compounds and the number of embedding dimensions. For example, using 300 reference compounds to generate a 100-dimensional embedding (r=300, d=100) will result in a job folder called run-300-100. The embedding result is the file matrix.<r>.<d>. In the above example, the output would be run-300-100/matrix.300.100.

Value

Creates files in dir ("run-r-d" by default). The return value is the name of the refIddb file, which needs to be given to other functions such as eiQuery or eiAdd.

Author(s)

Kevin Horan

See Also

eiInit eiPerformanceTest eiQuery

Examples

```
library(snow)

r<- 50
d<- 40

#initialize
data(sdfsample)
dir=file.path(tempdir(),"makedb")
dir.create(dir)
eiInit(sdfsample,dir=dir)

#create compound db
refIddb=eiMakeDb(r,d,numSamples=20,dir=dir,
   cl=makeCluster(1,type="SOCK",outfile=""))
```

---

eiPerformanceTest *Test the performance of LSH search*

Description

Tests the performance of embedding and LSH.

Usage

```
eiPerformanceTest(r,d,distance=getDefaultDist(descriptorType),descriptorType="ap",
conn=defaultConn(dir),dir=".",K=200, W = 1.39564, M=19,L=10,T=30)
```

Arguments

r	The number of references used to build the database you wish to query agains.
d	The number of dimensions used to build the database you wish to query agains.
distance	The distance function to be used to compute the distance between two descriptors. A default function is provided for "ap" and "fp" descriptors.
descriptorType	The format of the descriptor. Currently supported values are "ap" for atom pair, and "fp" for fingerprint.
conn	Database connection to use.
dir	The directory where the "data" directory lives. Defaults to the current directory.

K	Number of search results to use for LSH performance test.
W	Tunable LSH parameter. See LSHKIT page for details. http://lshkit.sourceforge.net/dd/d2a/mplsh-tune_8cpp.html
M	Tunable LSH parameter. See LSHKIT page for details. http://lshkit.sourceforge.net/dd/d2a/mplsh-tune_8cpp.html
L	Number of hash tables
T	Number of probes

Details

This will perform two different tests. The first tests the embedding results in similarity search. The way this works is by approximating 1,000 random similarity searches (determined by data/test_queries.iddb) by nearest neighbor search using the coordinates from the embedding results. The search results are then compared to the reference search results (chemical-search.results.gz).

The comparison results are summarized in two types of files. The first type lists the recall for different k values, k being the number of numbers to retrieve. These files are named as "recall-ratio-k". For example, if the recall is 70 compound search - 70 of the 100 results are among the real top-100 compounds - then the value at line 100 is 0.7. Several relaxation ration are used, each generating a file in this form. For instance, recall.ratio-10 is the file listing the recalls when relaxation ratio is 10. The other file, recall.csv, lists recalls of different relaxation ratios in one file by limiting to selected k value. In this CSV file, the rows correspond to different relaxation ratios, and the columns are different k values. You will be able to pick an appropriate relaxation ratio for the k values you are interested in.

The second test measures the performance of the Locality Sensitive Hash (LSH). The results for lsh-assisted search will be in run-r-d/indexed.performance. It's a 1,000-line files of recall values. Each line corresponds to one test query. LSH search performance is highly sensitive to your LSH parameters (K, W, M, L, T). The default parameters are listed in the man page for `eiPerformanceTest`. When you have your embedding result in a matrix file, you should follow instruction on http://lshkit.sourceforge.net/dd/d2a/mplsh-tune_8cpp.html to find the best values for these parameters.

Value

No value is returned. Creates files in `dir`/run-r-d.

Author(s)

Kevin Horan

See Also

`eiInit` `eiMakeDb` `eiQuery`

Examples

```
library(snow)

r<- 50
```

```
d<- 40

#initialize
data(sdfsample)
dir=file.path(tempdir(),"perf")
dir.create(dir)
eiInit(sdfsample,dir=dir)

#create compound db
eiMakeDb(r,d,numSamples=20,dir=dir,
   cl=makeCluster(1,type="SOCK",outfile=""))

eiPerformanceTest(r,d,dir=dir,K=22)
```

`eiQuery` *Perform a query on an embedded database*

Description

Finds similar compounds for each query.

Usage

```
eiQuery(r,d,refIddb,queries,format="sdf",
dir=".",descriptorType="ap",distance=getDefaultDist(descriptorType),conn=defaultConn(dir),
asSimilarity=FALSE, K=200, W = 1.39564, M=19,L=10,T=30,lshData=NULL,
mainIds = readIddb(file.path(dir, Main)))
```

Arguments

`r`	The number of references used to build the database you wish to query against.
`d`	The number of dimensions used to build the database you wish to query against.
`refIddb`	An Iddb formatted file containing the reference IDs of the database you wish to query against. This should almost always be the file: run-r-d/<long random string>.cdb. The `refIddb` value should also be returned by eiMakeDb.
`queries`	This can be either an SDFset, or a file containg 1 or more query compounds.
`format`	The format in which the queries are given. Valid values are: "sdf" when `queries` is either a filename of an sdf file, or and SDFset object; "compound_id" when `queries` is a list of id numbers; and "name", when `queries` is a list of compound names, as returned by `cid(apset)`.
`dir`	The directory where the "data" directory lives. Defaults to the current directory.
`descriptorType`	The format of the descriptor. Currently supported values are "ap" for atom pair, and "fp" for fingerprint.
`distance`	The distance function to be used to compute the distance between two descriptors. A default function is provided for "ap" and "fp" descriptors. The Tanimoto function is used by default.

conn	Database connection to use.
asSimilarity	If true, return similarity values instead of distance values. This only works in the given distance function returns values between 0 and 1. This is true for the default atom pair and finger print distance functions.
K	The number of results to return.
W	Tunable LSH parameter. See LSHKIT page for details. http://lshkit.sourceforge.net/dd/d2a/mplsh-tune_8cpp.html
M	Tunable LSH parameter. See LSHKIT page for details. http://lshkit.sourceforge.net/dd/d2a/mplsh-tune_8cpp.html
L	Number of hash tables
T	Number of probes
lshData	A pointer returned by loadLSHData. The LSH data is generally the largest chunk of data that must be held in memory while performing a query. Since it remains the same across queries it makes sense to pre-load the is data once when doing multiple queries. If this value is NULL the LSH data will be loaded internally and then released before eiQuery returns.
mainIds	A vector of all id numbers in the current database. This is mainly provided as an option here to avoid having to re-read the id list multiple times when executing several queries. If not supplied it will read it in itself.

Details

This function identifies the database by the r, d, and refIddb parameters. The queries can be given in a few different formats, see the queries parameter for details. The LSH algorithm is used to quickly identify compounds similar to the queries. This function must use a distance function rather than a similarity function. However, if the distance function given returns values between 0 and 1, then the asSimilarity parameter may be used to return similarity values rather than distance values.

Value

Returns a data frame with columns 'query', 'target', 'target_ids', and 'distance'. 'query' and 'target' are the compound names and distance is the distance between them, as computed by the given distance function.'target_ids' is the compound id of the target. Query namess are repeated for each matching target found. If asSimilarity is true then instead of a "distance" column there will be a "similarity" column.

Author(s)

Kevin Horan

See Also

eiInit eiMakeDb eiPerformanceTest

Examples

```
library(snow)
r<- 50
d<- 40

#initialize
data(sdfsample)
dir=file.path(tempdir(),"query")
dir.create(dir)
eiInit(sdfsample,dir=dir)

#create compound db
refIddb=eiMakeDb(r,d,numSamples=20,dir=dir,
   cl=makeCluster(1,type="SOCK",outfile=""))

#find compounds similar two each query
results = eiQuery(r,d,refIddb,sdfsample[1:2],K=15,dir=dir)
```

`example_compounds` *Example Compounds*

Description

122 compounds in SDF format, stored as a list. Each element of the list is one line of text. This is just used in some unit tests.

Format

The format is: chr [1:12222] "3540" " OpenBabel06051210572D" "" ...

`freeLSHData` *Free LSH Data*

Description

Free the memory allocated by `loadLSHData`.

Usage

```
freeLSHData(lshData)
```

Arguments

`lshData`	A pointer returned by `loadLSHData`.

Value

No return value.

Author(s)

Kevin Horan

See Also

loadLSHData eiQuery

Examples

```
## Not run:
lshData = loadLSHData(r,d)
eiQuery(r,d,refIddb,c("650002","650003"), format="name",K=15,lshData=lshData)
eiQuery(r,d,refIddb,c("650004","650005"), format="name",K=15,lshData=lshData)
freeLSHData(lshData)

## End(Not run)
```

---

loadLSHData *Load LSH Data*

Description

Load the LSH index and data. If many queries are going to be performed it is advantageous to load this object first and then hand it to eiQuery via the lshData parameter for each query.

If the data needs to be freed you can call the freeLSHData function.

Usage

```
loadLSHData(r, d, W = NA, M = NA, L = NA, K = NA, T = NA, dir = ".", matrixFile = NULL)
```

Arguments

r	The number of references used to build the database you wish to query against.
d	The number of dimensions used to build the database you wish to query against.
W	See eiQuery
M	See eiQuery
L	See eiQuery
K	See eiQuery
T	See eiQuery
dir	The directory where the "data" directory lives. Defaults to the current directory.
matrixFile	The path to the matrix file. If not specified it will look for it in the default spot.

Value

Returns a pointer to the allocated data. This should only be passed to other functions with an `lshData` parameter, such as `eiQuery`

Author(s)

Kevin Horan

See Also

`freeLSHData` `eiQuery`

Examples

```
## Not run:
lshData = loadLSHData(r,d)
eiQuery(r,d,refIddb,c("650002","650003"), format="name",K=15,lshData=lshData)
eiQuery(r,d,refIddb,c("650004","650005"), format="name",K=15,lshData=lshData)
freeLSHData(lshData)

## End(Not run)
```

---

setDefaultDistance *Set the default distance function for a descriptor type*

Description

Set the default distance function for a descriptor type. This is the distance function that will be used if none is given for a particular function call.

Usage

```
setDefaultDistance(descriptorType, distance)
```

Arguments

`descriptorType`	The type of the descriptor to set a distance function for. Built-in values are "ap" and "fp". Additional values can be set as well.
`distance`	A distance function taking two descriptor objects (as returned by toObject in a descriptor transform, see `\ ink{addTransform}` for details), and returning a distance value.

Value

No return value.

Author(s)

Kevin Horan

See Also

addTransform

Examples

```
setDefaultDistance("ap", function(d1,d2) 1-cmp.similarity(d1,d2) )
```

Index