Package 'largeVis'

May 8, 2017

Type Package

Title High-Quality Visualizations of Large, High-Dimensional Datasets

Version 0.2.1

Author Amos B. Elberg

Maintainer Amos Elberg <amos.elberg@gmail.com>

Description Implements the largeVis algorithm (see Tang, et al. (2016) <DOI:10.1145/2872427.2883041>) for visualizing very large high-dimensional datasets. Also very fast search for approximate nearest neighbors; outlier detection; and optimized implementations of the HDBSCAN*, DBSCAN and OPTICS clustering algorithms; plotting functions for visualizing the above.

License GPL-3

LazyData TRUE

RoxygenNote 6.0.1

Depends R (>= 3.0.2), Matrix

Imports ggplot2 (>= 2.1.0)

LinkingTo Rcpp (>= 0.12.4),RcppProgress (>= 0.2.1),RcppArmadillo (>= 0.7.200.2.0),testthat(>= 1.0.2)

Suggests ggforce, testthat, knitr, rmarkdown, png, dbscan

URL https://github.com/elbamos/largeVis

BugReports https://github.com/elbamos/largeVis/issues

NeedsCompilation yes

BuildVignettes TRUE

VignetteBuilder knitr

SystemRequirements C++11

Repository CRAN

Date/Publication 2017-05-08 12:01:23 UTC

R topics documented:

`largeVis-package` *largeVis: high-quality visualizations for large, high-dimensionality datasets*

Description

This is an implementation of the `largeVis` algorithm by Tang et al., and related functions and algorithms.

Details

`largeVis` estimates a low-dimensional embedding for high-dimensional data, where the distance between vertices in the low-dimensional space is proportional to the distance between them in the high-dimensional space. The algorithm works in 4 phases:

- Estimate candidate nearest-neighbors for each vertex by building `n.trees` random projection trees.
- Estimate `K` nearest-neighbors for each vertex by visiting each vertex' 2d-degree neighbors (its neighbors' neighbors). This is repeated `max.iter` times. Note that the original paper suggested a `max.iter` of 1, however a larger number may be appropriate for some datasets if the algorithm has trouble finding K neighbors for every vertex.
- Estimate $p_{j|i}$, the conditional probability that each edge found in the previous step is actually to a nearest neighbor of each of its nodes.
- Using stochastic gradient descent, estimate an embedding for each vertex in the low-dimensional space.

The nearest-neighbor search functionality is also available as a separate function, where it offers an extremely fast approximate nearest-neighbor search. (See the Benchmarks vignette for details.)

The package also includes implementations of the HDBSCAN, DBSCAN, and OPTICS clustering algorithms, and LOF outlier detection, optimized to use data generated by running `largeVis`.

References

Jian Tang, Jingzhou Liu, Ming Zhang, Qiaozhu Mei. Visualizing Large-scale and High-dimensional Data. R. Campello, D. Moulavi, and J. Sander, Density-Based Clustering Based on Hierarchical Density Estimates In: Advances in Knowledge Discovery and Data Mining, Springer, pp 160-172. 2013 Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jorg Sander (1999). OPTICS: Ordering Points To Identify the Clustering Structure. ACM SIGMOD international conference on Management of data. ACM Press. pp. 49-60. Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu (1996). Evangelos Simoudis, Jiawei Han, Usama M. Fayyad, eds. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226-231. ISBN 1-57735-004-9.

See Also

Useful links:

- https://github.com/elbamos/largeVis
- Report bugs at https://github.com/elbamos/largeVis/issues

`as.dendrogram.hdbscan` *as.dendrogram.hdbscan*

Description

Convert an hdbscan object into a dendrogram compatible with the `stats` package.

Usage

```
## S3 method for class 'hdbscan'
as.dendrogram(object, includeNodes = FALSE, ...)
```

Arguments

`object`	An `hdbscan` object.
`includeNodes`	Whether individual nodes should be included in the dedrogram. Can cause a substantial increase in the size of the object.
`...`	For compatibility with `as.dendrogram`, and currently ignored.

Value

A dendrogram object, where nodes have the following attributes:

'leaf' As in dendrogram.

'members' As in dendrogram.

'size' The number of nodes underneath the cluster.

'height' The core distance at which the cluster or node was merged.

'probability' The probability that the leaf is a true member of its assigned cluster.

'GLOSH' The leaf's GLOSH outlier score.

'stability' The node's determined stability, taking into account child-node stabilities. Missing for leaves.

'selected' Whether the node was selected as a cluster. Missing for leaves. Note that when a node is selected, all points under child branches are assigned to the same cluster.

'cluster' The cluster number, for reference against the hdbscan object.

Note

The hdbscan algorithm works by first building a hierarchy based on a minimal spanning tree, then consolidating nodes according to rules in the algorithm. The algorithm then selects some of the consolidated nodes as clusters, deselecting others. For example, if Node A has children B and C, the algorithm might select A, and then all points under A, B, and C would be assigned to the same cluster. Or, it might deselect A, and select B and C instead. In that case, the nodes under B would be assigned to one cluster, the nodes under C to a different cluster, and nodes under A but not B or C would not be assigned to any cluster. This function returns a dendrogram of the middle stage, the hierarchy of consolidated nodes. Whether a node was selected as as cluster is an attribute of each node.

This function remains experimental in terms of finding the best way to represent an hdbscan object in a dendrogram.

Examples

```
data(iris)
vis <- largeVis(t(iris[,1:4]), K = 20, sgd_batches = 1, threads = 1)
hdbscanobj <- hdbscan(vis, minPts = 10, K = 5)
plot(as.dendrogram(hdbscanobj))
```

---

buildEdgeMatrix *Build an nearest-neighbor graph weighted by distance.*

Description

Build an nearest-neighbor graph weighted by distance.

Convert an edge matrix to a dist object.

Usage

```
buildEdgeMatrix(data, neighbors = NULL, distance_method = "Euclidean",
  threads = NULL, verbose = getOption("verbose", TRUE), ...)

## S3 method for class 'edgematrix'
as.dist(m, diag = FALSE, upper = FALSE)
```

Arguments

data	A matrix with a number of columns equal to the number of columns in ‘x‘
neighbors	An adjacency matrix of the type produced by randomProjectionTreeSearch. If NULL, randomProjectionTreeSearch will be run with parameters given by ....
distance_method	One of "Euclidean" or "Cosine"
threads	The number of threads to use in calculating distance; set automatically if NULL (the default).
verbose	Verbosity
...	Additional parameters passed to randomProjectionTreeSearch if neighbors is NULL.
m	An ‘edgematrix‘ object.
diag	logical value indicating whether the diagonal of the distance matrix should be printed by print.dist.
upper	logical value indicating whether the upper triangle of the distance matrix should be printed by print.dist.

Value

An ‘edgematrix‘ object consisting of the elements of a sparse matrix, with the distance method stored in attribute method.

A dist object.

Note

This method converts the otherwise sparse edge matrix into a dense dist object, where any distances absent from the edge matrix are represented as NA.

---

buildWijMatrix *buildWijMatrix*

Description

Rescale the weights in an edge matrix to match a given perplexity.

Usage

```
buildWijMatrix(x, threads = NULL, perplexity = 50)

## S3 method for class 'edgematrix'
buildWijMatrix(x, threads = NULL, perplexity = 50)

## S3 method for class 'TsparseMatrix'
buildWijMatrix(x, threads = NULL, perplexity = 50)

## S3 method for class 'CsparseMatrix'
buildWijMatrix(x, threads = NULL, perplexity = 50)
```

Arguments

x	An edgematrix, either an 'edgematrix' object or a sparse matrix.
threads	The maximum number of threads to spawn. Determined automatically if NULL (the default).
perplexity	Given perplexity.

Value

A list with the following components:

'dist' An [N,K] matrix of the distances to the nearest neighbors.

'id' An [N,K] matrix of the node indexes of the neartest neighbors. Note that this matrix is 1-indexed, unlike most other matrices in this package.

'k' The number of nearest neighbors.

---

distance *Calculate pairwise Euclidean or angular distances efficiently*

Description

This function is a wrapper around a C++ function that calculates pairwise distances in a memory- and CPU-efficient manner.

Usage

```
distance(x, i, j, distance_method, threads = NULL, verbose)

## S3 method for class 'matrix'
distance(x, i, j, distance_method = "Euclidean",
  threads = NULL, verbose = getOption("verbose", TRUE))

## S3 method for class 'CsparseMatrix'
distance(x, i, j, distance_method = "Euclidean",
```

```
  threads = NULL, verbose = getOption("verbose", TRUE))

## S3 method for class 'TsparseMatrix'
distance(x, i, j, distance_method = "Euclidean",
  threads = NULL, verbose = getOption("verbose", TRUE))
```

Arguments

x	A (potentially sparse) matrix, where examples are columns and features are rows.
i	0-indexed vector of column indices.
j	0-indexed vector of column indices.
distance_method	One of "Euclidean" or "Cosine."
threads	The maximum number of threads to spawn. Determined automatically if NULL (the default).
verbose	Verbosity.

Details

The Euclidean or angular distances between columns in ‘x‘ identified by parameters ‘i‘ and ‘j‘ are calculated and returned.

Value

A vector of the distances between the columns in ‘x‘ indexed by ‘i‘ and ‘j‘, with attribute `method` giving the `distance_method`.

---

ggManifoldMap *Visualize an embedding by ggplotting with images*

Description

Identical to manifoldMap, but adds images to an existing `ggplot2` object or creates one.

Usage

```
ggManifoldMap(ggObject = NULL, x, n = nrow(x), images, scale = 1)
```

Arguments

ggObject	a `ggplot` object. If not provided, a new `ggplot` object with `geom_blank` will be created.
x	A `largeVis` object or [N,D] matrix of coordinates.
n	The number of images to sample.
images	The images. A 3-D or 4-D array.
scale	Proportion to scale the images to.

Details

See manifoldMap. Note that this function can be considerably slower to display than manifoldMap. It therefore should only be used if other features of ggplot2 are required.

If the objects in the list are matrix objects, or the array is 3-dimensional, the images will be treated as greyscale. If there is an additional dimension, it must have a length of 3 and be RGB color layers.

Value

A ggplot object.

---

gplot *gplot*

Description

Plot an hdbscan object, using ggplot. The plot is primarily intended for diagnostic purposes, but can be useful to undertand how clusters were generated.

Usage

```
gplot(x, coords, text = FALSE, distances = FALSE)
```

Arguments

x	An hdbscan object.
coords	Coordinates for the points clustered in x.
text	If TRUE, include on the plot labels for each node's index. If "parent", the labels will instead be the index number of the node's cluster.
distances	If TRUE, draw circles representing the core distances around each point.

Details

Point color corresponds to clusters, with outliers as the NA color. Alpha corresponds to the node's GLOSH score. The segments on the plot correspond to the connections on the minimum spanning tree. Segment alpha corresponds to λ_p.

If the parameter text is set to TRUE or "parent", the nodes will be labelled with the node index number or cluster index number, respectively.

Value

A ggplot object

Examples

```
## Not run:
library(largeVis)
# The aggregation dataset can be downloaded from https://github.com/elbamos/clusteringdatasets
data(Aggregation)
dat <- as.matrix(Aggregation[, 1:2])
aggregateVis <- largeVis(dat, K = 10, tree_threshold = 100,
                         max_iter = 5, sgd_batches = 1)
clusters <- hdbscan(aggregateVis, verbose = FALSE)
gplot(clusters, dat)

## End(Not run)
```

`hdbscan` *hdbscan*

Description

Implemenation of the hdbscan algorithm.

Usage

```
hdbscan(edges, neighbors = NULL, minPts = 20, K = 5, threads = NULL,
  verbose = getOption("verbose", TRUE))
```

Arguments

`edges`	An edge matrix of the type returned by `buildEdgeMatrix` or, alternatively, a `largeVis` object.
`neighbors`	An adjacency matrix of the type returned by `randomProjectionTreeSearch`. Must be specified unless `edges` is a `largeVis` object.
`minPts`	The minimum number of points in a cluster.
`K`	The number of points in the core neighborhood. (See details.)
`threads`	Maximum number of threads. Determined automatically if `NULL` (the default). It is unlikely that this parameter should ever need to be adjusted. It is only available to make it possible to abide by the CRAN limitation that no package use more than two cores.
`verbose`	Verbosity.

Details

The hyperparameter `K` controls the size of core neighborhoods. The algorithm measures the density around a point as 1 / the distance between that point and its kth nearest neighbor. A low value of `K` is similar to clustering nearest neighbors rather than based on density. A high value of `K` may cause the algorithm to miss some (usually contrived) clustering patterns, such as where clusters are made up of points arranged in lines to form shapes.

The function must be provided sufficient nearest-neighbor data for whatever is specified for k. If k = 5, for example, the edge matrix (and neighbor matrix, if specified) must contain data on at least 5 neighbors for each point. This should not be problematic in typical use in connection with `largeVis`, which is ordinarily run with a far higher k-value than hdbscan.

Value

An object of type `hdbscan` with the following fields:

'clusters' A vector of the cluster membership for each vertex. Outliers are given `NA`

'probabilities' A vector of the degree of each vertex' membership. This is calculated by standardizing each vertex' $lambda_p$ against the $lambda_birth$ and $lambda_deaeth$ of the cluster.

'glosh' A vector of GLOSH outlier scores for each node assigned to a cluster. NA for nodes not in a cluster.

'tree' The minimum spanning tree used to generate the clustering.

'hierarchy' A representation of the condensed cluster hierarchy.

'call' The call.

The hierarchy describes the complete post-condensation structure of the tree:

'nodemembership' The cluster ID of the vertex's immediate parent, after condensation.

'lambda' λ_p for each vertex.

'parent' The cluster ID of each cluster's parent.

'stability' The cluster's stability, taking into account child-node stabilities.

'selected' Whether the cluster was selected.

'coredistances' The core distance determined for each vertex.

'lamba_birth' λ_b for each cluster.

'lambda_deaeth' λ_d for each cluster.

Note

This is not precisely the `HDBSCAN` algorithm because it relies on the nearest neighbor data generated by the `LargeVis` algorithm. In particular, `HDBSCAN` assumes that all points can be connected in a single minimal-spanning tree. This implementation uses a minimal-spanning forest, because the graph may not be fully connected depending on the amount of nearest-neighbor data provided. If, for example, the data has distinct clusters where no member of some cluster is a nearest neighbor of a point in any other cluster, which can easily happen, the algorithm will generate distinct trees for each disconnected set of points. In testing, this improved the performance of the algorithm.

Do not rely on the content of the `probabilities` field for outliers. A future version will hopefully provide some measure of outlier factor.

References

R. Campello, D. Moulavi, and J. Sander, Density-Based Clustering Based on Hierarchical Density Estimates In: Advances in Knowledge Discovery and Data Mining, Springer, pp 160-172. 2013

See Also

https://github.com/lmcinnes/hdbscan

Examples

```
## Not run:
library(largeVis)
library(clusteringdatasets)  # See https://github.com/elbamos/clusteringdatasets
data(spiral)
dat <- as.matrix(spiral[, 1:2])
neighbors <- randomProjectionTreeSearch(t(dat), K = 10, tree_threshold = 100,
                                        max_iter = 5, threads = 1)
edges <- buildEdgeMatrix(t(dat), neighbors)
clusters <- hdbscan(edges, neighbors = neighbors, verbose = FALSE, threads = 1)

# Calling largeVis while setting sgd_batches to 1 is
# the simplest way to generate the data structures neeeded for hdbscan
spiralVis <- largeVis(t(dat), K = 10, tree_threshold = 100, max_iter = 5,
                      sgd_batches = 1, threads = 1)
clusters <- hdbscan(spiralVis, verbose = FALSE, threads = 1)
# The gplot function helps to visualize the clustering
largeHighDimensionalDataset <- matrix(rnorm(50000), ncol = 50)
vis <- largeVis(largeHighDimensionalDataset)
clustering <- hdbscan(vis)
gplot(clustering, t(vis$coords))

## End(Not run)
```

`largeVis` *Apply the LargeVis algorithm for visualizing large high-dimensional datasets.*

Description

Apply the LargeVis algorithm for visualizing large high-dimensional datasets.

Usage

```
largeVis(x, dim = 2, K = 50, n_trees = 50, tree_threshold = max(10,
  min(nrow(x), ncol(x))), max_iter = 1, distance_method = "Euclidean",
  perplexity = max(50, K/3), save_neighbors = TRUE, save_edges = TRUE,
  threads = NULL, verbose = getOption("verbose", TRUE), ...)
```

Arguments

`x`	A matrix, where the features are rows and the examples are columns.
`dim`	The number of dimensions in the output
`K`	The number of nearest-neighbors to use in computing the kNN graph

n_trees	See randomProjectionTreeSearch. The default is set at 50, which is the number used in the examples in the original paper.
tree_threshold	See randomProjectionTreeSearch. By default, this is the number of features in the input set.
max_iter	See randomProjectionTreeSearch.
distance_method	One of "Euclidean" or "Cosine." See randomProjectionTreeSearch.
perplexity	See buildWijMatrix.
save_neighbors	Whether to include in the output the adjacency matrix of nearest neighbors.
save_edges	Whether to include in the output the distance matrix of nearest neighbors.
threads	The maximum number of threads to spawn. Determined automatically if NULL (the default). It is unlikely that this parameter should ever need to be adjusted. It is only available to make it possible to abide by the CRAN limitation that no package use more than two cores.
verbose	Verbosity
...	Additional arguments passed to projectKNNs.

Value

A 'largeVis' object with the following slots:

'knns' If save_neighbors=TRUE, An [N,K] 0-indexed integer matrix, which is an adjacency list of each vertex' identified nearest neighbors. If the algorithm failed to find K neighbors, the matrix is padded with NA's. Note that this matrix is not identical to the output from randomProjectionTreeSearch: missing neighbors are NA's rather than -1's, and the matrix is transposed.

'edges' If save_edges=TRUE, a [N,N] sparse matrix of distances between nearest neighbors.

'wij' A sparse [N,N] matrix where each cell represents w_{ij}.

'call' The call.

'coords' A [D,N] matrix of the embedding of the dataset in the low-dimensional space.

References

Jian Tang, Jingzhou Liu, Ming Zhang, Qiaozhu Mei. Visualizing Large-scale and High-dimensional Data.

Examples

```
# iris
data(iris)
dat <- as.matrix(iris[,1:4])
visObject <- largeVis(dat, max_iter = 20, K = 10, sgd_batches = 10000, threads = 1)
plot(t(visObject$coords))

## Not run:
# mnist
```

```
# Note: The MNIST dataset may be obtained using the deepnet package.
load("./mnist.Rda")
dat <- mnist$images
dim(dat) <- c(42000, 28 * 28)
dat <- (dat / 255) - 0.5
dat <- t(dat)
visObject <- largeVis(dat, n_trees = 50, tree_th = 200, K = 50)
plot(t(visObject$coords))

## End(Not run)
```

`lof` *Local Outlier Factor Score*

Description

Calculate the Local Outlier Factor (LOF) score for each data point given knowledge of k-Nearest Neighbors.

Usage

```
lof(edges)
```

Arguments

`edges`	An 'edgematrix' of the type produced by `buildEdgeMatrix`.

Value

A vector of LOF values for each data point.

References

Based on code in the `dbscan` package.

`lv_dbscan` *lv_dbscan*

Description

Implementation of the DBSCAN algorithm using largeVis datastructures.

Usage

```
lv_dbscan(edges, neighbors, eps = Inf, minPts = nrow(neighbors - 1),
  verbose = getOption("verbose", TRUE))
```

Arguments

edges	An ‘edgematrix‘ object. Alternatively, a largeVis object, in which case edges and neighbors will be taken from the edges and knns parameters, respectively.
neighbors	An adjacency matrix of the type produced by randomProjectionTreeSearch
eps	See dbscan.
minPts	See dbscan.
verbose	Vebosity level.

Details

The DBSCAN algorithm attempts to find clusters of a minimum density given by eps. This implementation leverages the nearest neighbor data assembled by largeVis.

Value

A dbscan object.

References

Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu (1996). Evangelos Simoudis, Jiawei Han, Usama M. Fayyad, eds. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226-231. ISBN 1-57735-004-9.

---

lv_optics *lv_optics*

Description

Experimental implementation of the OPTICS algorithm.

Usage

```
lv_optics(edges, neighbors, eps = Inf, minPts = nrow(neighbors), eps_cl, xi,
  useQueue = TRUE, verbose = getOption("verbose", TRUE))
```

Arguments

edges	A weighted graph of the type produced by buildEdgeMatrix. Alternatively, a largeVis object, in which case edges and neighbors will be taken from the edges and knns parameters, respectively.
neighbors	An adjacency matrix of the type produced by randomProjectionTreeSearch
eps	See optics.
minPts	See optics.
eps_cl	See optics.

xi	See optics.
useQueue	Whether to process points in order of core distance. (See note.)
verbose	Vebosity level.

Details

This is an implementation of the OPTICS algorithm that attempts to leverage the largeVis nearest-neighbor search.

This implementation does not use the OPTICS neighbor-search strategy, in favor of using the pre-calculated neighbor matrix produced incidentally by largeVis. It is therefore a variant of OPTICS rather than an implementation of the original, and the results vary slightly from those obtained by the implementations in ELKI and the dbscan package.

Value

An optics object.

Note

The useQueue parameter controls the order in which points that have not yet been visisted are processed. If FALSE, points are processed in order of rows. If TRUE, they are processed in ascending order of core distance. FALSE is more compatible with the implementations in the dbscan package and in the ELKI Java clustering package. TRUE may produce preferrable results.

References

Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jorg Sander (1999). OPTICS: Ordering Points To Identify the Clustering Structure. ACM SIGMOD international conference on Management of data. ACM Press. pp. 49-60.

---

manifoldMap *Visualize an embedding by plotting with images*

Description

Makes a plot of n images sampled from images, positions at coordinates given by x.

Usage

```
manifoldMap(x, n = nrow(x), images, scale = 1, ...)
```

Arguments

x	A largeVis object or [N,D] matrix of coordinates.
n	The number of images to sample.
images	The images. A 3-D or 4-D array.
scale	Proportion to scale the images to.
...	Addiitional parameters passed to plot.

Details

The images can be passed in either as a list or a 3- or 4-dimensional array. The first dimension is `n`.

If the objects in the list are `matrix` objects, or the array is 3-dimensional, the images will be treated as greyscale. If there is an additional dimension, it must have a length of 3 and be RGB color layers.

References

Andrej Karpapthy. t-SNE Visualization of CNN Codes.

See Also

`ggManifoldMap`

Examples

```
## Not run:
load("mnist.Rda")
load("mnistcoordinates.Rda")

flip <- function(x) apply(x,2,rev)
rotate <- function(x) t(flip(x))

mnistimages <- apply(mnist$images,
   MARGIN=1,
   FUN = function(x) as.array(rotate(flip(x))))
mnistimages <- t(mnistimages)
dim(mnistimages) <- c(42000, 28, 28)

manifoldMap(coords,
   1000,
   scale = 0.07,
   mnistimages)

## End(Not run)
```

---

manifoldMapStretch *manifoldMapStretch*

Description

A manifold map that fills the full extent of the plot.

Usage

```
manifoldMapStretch(x, f, size_x = 500, size_y = 500, image_size = 50, ...)
```

Arguments

x	A [N,D] matrix of coordinates.
f	A function that, called with the index number of a row of x, returns an R object representing an image. See the example.
size_x	The width of the requested plot, in pixels.
size_y	The height of the requested plot, in pixels.
image_size	The size to plot each image; each is plotted as a square.
...	Additional parameters passed to plot.

Details

Ported from http://cs.stanford.edu/people/karpathy/cnnembed/. Each position is filled with its nearest neighbor.

Note

This function is experimental.

Examples

```
## Not run:
# Demonstration of f
load(system.file("extdata", "faces.Rda", package="largeVis"))

imagepaths <- paste("pathtoimages",
   faceLabels[,1], sub("png", "jpg", faceLabels[,2]), sep = "/")

manifoldMapStretch(as.matrix(faceCoords[,1:2]),
   f = function(x) jpeg::readJPEG(imagePaths[x]),
   size_x = 5000, size_y = 5000, image_size = 100)

## End(Not run)
```

---

neighborsToVectors *A utility function to convert a k-NN graph to a pair of 0-indexed vectors of indices.*

Description

In the returned list, the nodes indexed by 'j' are the identified nearest neighbors of the nodes indexed by 'i'. In other words, if 'i = c(0,0,0,1,1,1)' and 'j = c(1,2,3,2,3,4)', then nodes '1, 2 & 3' are nearest neighbors of node 0, but node 0 is not a nearest neighbor of node 1.

Usage

```
neighborsToVectors(x)
```

Arguments

x	A '[K,N]' matrix of indices of the nearest neighbors of each vertex. 0-indexed.

Value

A list with fields:

i The slowly-varying indices of x

j The quickly-varying indices of x

---

projectKNNs *Project a distance matrix into a lower-dimensional space.*

Description

Takes as input a sparse matrix of the edge weights connecting each node to its nearest neighbors, and outputs a matrix of coordinates embedding the inputs in a lower-dimensional space.

Usage

```
projectKNNs(wij, dim = 2, sgd_batches = NULL, M = 5, gamma = 7,
  alpha = 1, rho = 1, coords = NULL, useDegree = FALSE,
  momentum = NULL, seed = NULL, threads = NULL,
  verbose = getOption("verbose", TRUE))
```

Arguments

wij	A symmetric sparse matrix of edge weights, in C-compressed format, as created with the Matrix package.
dim	The number of dimensions for the projection space.
sgd_batches	The number of edges to process during SGD. Defaults to a value set based on the size of the dataset. If the parameter given is between 0 and 1, the default value will be multiplied by the parameter.
M	The number of negative edges to sample for each positive edge.
gamma	The strength of the force pushing non-neighbor nodes apart.
alpha	Hyperparameter used in the default distance function, $1/(1+\alpha\dot{\|}\|y_i-y_j\|\|^2)$. The function relates the distance between points in the low-dimensional projection to the likelihood that the two points are nearest neighbors. Increasing α tends to push nodes and their neighbors closer together; decreasing α produces a broader distribution. Setting α to zero enables the alternative distance function. α below zero is meaningless.
rho	Initial learning rate.
coords	An initialized coordinate matrix.
useDegree	Whether to use vertex degree to determine weights in negative sampling (if TRUE), or the sum of the vertex's edges (the default). See Notes.

momentum	If not NULL (the default), SGD with momentum is used, with this multiplier, which must be between 0 and 1. Note that momentum can drastically speed-up training time, at the cost of additional memory consumed.
seed	Random seed to be passed to the C++ functions; sampled from hardware entropy pool if NULL (the default). Note that if the seed is not NULL (the default), the maximum number of threads will be set to 1 in phases of the algorithm that would otherwise be non-deterministic.
threads	The maximum number of threads to spawn. Determined automatically if NULL (the default).
verbose	Verbosity

Details

The algorithm attempts to estimate a dim-dimensional embedding using stochastic gradient descent and negative sampling.

The objective function is:

$$O = \sum_{(i,j) \in E} w_{ij} (\log f(||p(e_{ij} = 1||) + \sum_{k=1}^{M} E_{jk\ P_n(j)} \gamma \log(1 - f(||p(e_{ij_k} - 1||)))$$

where $f()$ is a probabilistic function relating the distance between two points in the low-dimensional projection space, and the probability that they are nearest neighbors.

The default probabilistic function is $1/(1 + \alpha \dot{||}x||^2)$. If α is set to zero, an alternative probabilistic function, $1/(1 + \exp(x^2))$ will be used instead.

Note that the input matrix should be symmetric. If any columns in the matrix are empty, the function will fail.

Value

A dense [N,D] matrix of the coordinates projecting the w_ij matrix into the lower-dimensional space.

Note

If specified, seed is passed to the C++ and used to initialize the random number generator. This will not, however, be sufficient to ensure reproducible results, because the initial coordinate matrix is generated using the R random number generator. To ensure reproducibility, call set.seed before calling this function, or pass it a pre-allocated coordinate matrix.

The original paper called for weights in negative sampling to be calculated according to the degree of each vertex, the number of edges connecting to the vertex. The reference implementation, however, uses the sum of the weights of the edges to each vertex. In experiments, the difference was imperceptible with small (MNIST-size) datasets, but the results seems aesthetically preferrable using degree. The default is to use the edge weights, consistent with the reference implementation.

Examples

```
## Not run:
data(CO2)
CO2$Plant <- as.integer(CO2$Plant)
CO2$Type <- as.integer(CO2$Type)
CO2$Treatment <- as.integer(CO2$Treatment)
co <- scale(as.matrix(CO2))
# Very small datasets often produce a warning regarding the alias table.  This is safely ignored.
suppressWarnings(vis <- largeVis(t(co), K = 20, sgd_batches = 1, threads = 2))
suppressWarnings(coords <- projectKNNs(vis$wij, threads = 2))
plot(t(coords))

## End(Not run)
```

`randomProjectionTreeSearch` *Find approximate k-Nearest Neighbors using random projection tree search.*

Description

A fast and accurate algorithm for finding approximate k-nearest neighbors.

Usage

```
randomProjectionTreeSearch(x, K = 150, n_trees = 50,
  tree_threshold = max(10, nrow(x)), max_iter = 1,
  distance_method = "Euclidean", seed = NULL, threads = NULL,
  verbose = getOption("verbose", TRUE))

## S3 method for class 'matrix'
randomProjectionTreeSearch(x, K = 150, n_trees = 50,
  tree_threshold = max(10, nrow(x)), max_iter = 1,
  distance_method = "Euclidean", seed = NULL, threads = NULL,
  verbose = getOption("verbose", TRUE))

## S3 method for class 'CsparseMatrix'
randomProjectionTreeSearch(x, K = 150, n_trees = 50,
  tree_threshold = max(10, nrow(x)), max_iter = 1,
  distance_method = "Euclidean", seed = NULL, threads = NULL,
  verbose = getOption("verbose", TRUE))

## S3 method for class 'TsparseMatrix'
randomProjectionTreeSearch(x, K = 150, n_trees = 50,
  tree_threshold = max(10, nrow(x)), max_iter = 1,
  distance_method = "Euclidean", seed = NULL, threads = NULL,
  verbose = getOption("verbose", TRUE))
```

Arguments

x	A (potentially sparse) matrix, where examples are columnns and features are rows.
K	How many nearest neighbors to seek for each node.
n_trees	The number of trees to build.
tree_threshold	The threshold for creating a new branch. The paper authors suggest using a value equivalent to the number of features in the input set.
max_iter	Number of iterations in the neighborhood exploration phase.
distance_method	One of "Euclidean" or "Cosine."
seed	Random seed passed to the C++ functions. If seed is not NULL (the default), the maximum number of threads will be set to 1 in phases that would be non-determinstic otherwise.
threads	The maximum number of threads to spawn. Determined automatically if NULL (the default).
verbose	Whether to print verbose logging using the progress package.

Details

Note that the algorithm does not guarantee that it will find K neighbors for each node. A warning will be issued if it finds fewer neighbors than requested. If the input data contains distinct partitionable clusters, try increasing the tree_threshold to increase the number of returned neighbors.

Value

A [K, N] matrix of the approximate K nearest neighbors for each vertex.

---

sgdBatches *sgdBatches*

Description

Calculate the default number of batches for a given number of vertices and edges.

Usage

```
sgdBatches(N, E = 150 * N/2)
```

Arguments

N	Number of vertices.
E	Number of edges.

Details

The formula used is the one used by the `LargeVis` reference implementation. This is substantially less than the recommendation $E * 10000$ in the original paper.

Value

The recommended number of sgd batches.

Examples

```
# Observe that increasing K has no effect on processing time
N <- 70000 # MNIST
K <- 10:250
plot(K, sgdBatches(rep(N, length(K)), N * K / 2))

# Observe that processing time scales linarly with N
N <- c(seq(from = 1, to = 10000, by = 100), seq(from = 10000, to = 10000000, by = 1000))
plot(N, sgdBatches(N))
```

Index