Package ‘diffusionMap’

October 13, 2022

Type Package

Title Diffusion Map

Version 1.2.0

Description Implements diffusion map method of data parametrization, including creation and visualization of diffusion map, clustering with diffusion K-means and regression using adaptive regression model. Richards (2009) <doi:10.1088/0004-637X/691/1/32>.

Depends R (>= 2.10)

Imports scatterplot3d, graphics, igraph, Matrix, stats

License GPL-3

URL https://github.com/rcannood/diffusionMap

RoxygenNote 6.1.1

Encoding UTF-8

LazyData true

NeedsCompilation no

Author Joseph Richards [aut] (joeyrichar), Robrecht Cannoodt [aut, cre] (<https://orcid.org/0000-0003-3641-729X>, rcannood)

Maintainer Robrecht Cannoodt <rcannood@gmail.com>

Repository CRAN

Date/Publication 2019-09-10 22:50:18 UTC

R topics documented:

`adapreg` *Adaptive Regression*

Description

Non-parametric adaptive regression method for diffusion map basis.

Usage

```
adapreg(D, y, mmax = min(50, length(y)), fold = NULL, nfolds = 10,
  nrep = 5)
```

Arguments

`D`	n-by-n pairwise distance matrix for a data set with n points, or alternatively output from the dist() function
`y`	vector of responses to model
`mmax`	maximum model size to consider
`fold`	vector of integers of size n specifying the k-fold cross-validation allocation. Default does nfolds-fold CV by sample(1:nfolds,length(y),replace=T)
`nfolds`	number of folds to do CV. If fold is supplied, nfolds is ignored
`nrep`	number of times optimization algorithm is run (with random initializations). Higher nrep allows algorithm to avoid getting stuck in local minima

Details

Fits an adaptive regression model leaving as free parameters both the diffusion map localness parameter, epsilon, and the size of the regression model, m. The adaptive regression model is the expansion of the response function on the first m diffusion map basis functions.

This routine searches for the optimal (epsilon,m) by minimizing the cross-validation risk (CV MSE) of the regression estimate. The function uses `optimize()` to search over an appropriate range of epsilon and calls the function `adapreg.m()` to find the optimal m for each epsilon.

Default uses 10-fold cross-validation to choose the optimal model size. User may also supply a vector of fold allocations. For instance, sample(1:10,length(y),replace=T) does 10-fold CV while 1:length(y) performs leave-one-out CV.

Value

The returned value is a list with components

`mincvrisk`	minimum cross-validation risk for the adaptive regression model for the given epsilon
`mopt`	size of the optimal regression model. If mopt == mmax, it is advised to increase mmax.
`epsopt`	optimal value of epsilon used in diffusion map construction
`y.hat`	predictions of the response, y-hat, for the optimal model
`coeff`	coefficients of the optimal model

References

Richards, J. W., Freeman, P. E., Lee, A. B., and Schafer, C. M., (2009), ApJ, 691, 32

See Also

`diffuse()`,`adapreg.m()`

Examples

```
library(scatterplot3d)
## trig function on circle
t=seq(-pi,pi,.01)
x=cbind(cos(t),sin(t))
y = cos(3*t) + rnorm(length(t),0,.1)
tcol = topo.colors(32)
colvec = floor((y-min(y))/(max(y)-min(y))*32); colvec[colvec==0] = 1
scatterplot3d(x[,1],x[,2],y,color=tcol[colvec],pch=20,
  main="Cosine function supported on circle",angle=55,
  cex.main=2,col.axis="gray",cex.symbols=2,cex.lab=2,
  xlab=expression("x"[1]),ylab=expression("x"[2]),zlab="y")

D = as.matrix(dist(x))
# do 10-fold cross-validation to optimize (epsilon, m):
AR = adapreg(D,y, mmax=5,nfolds=2,nrep=2)
print(paste("optimal model size:",AR$mopt,"; optimal epsilon:",
  round(AR$epsopt,4),"; min. CV risk:",round(AR$mincvrisk,5)))
plot(y,AR$y.hat,ylab=expression(hat("y")),cex.lab=1.5,cex.main=1.5,
  main="Predictions")
abline(0,1,col=2,lwd=2)
```

`adapreg.m` *Adaptive Regression*

Description

Non-parametric adaptive regression method for diffusion map basis.

Usage

```
adapreg.m(epsilon, D, y, mmax = min(50, length(y)), fold = NULL,
  nfolds = 10, objfun = FALSE)
```

Arguments

`epsilon`	diffusion map kernel parameter
`D`	n-by-n pairwise distance matrix for a data set with n points, or alternatively output from the dist() function
`y`	vector of responses to model
`mmax`	maximum model size to consider
`fold`	vector of integers of size n specifying the k-fold cross-validation allocation. Default does nfolds-fold CV by sample(1:nfolds,length(y),replace=T)
`nfolds`	number of folds to do CV. If fold is supplied, nfolds is ignored
`objfun`	if the function is to be passed into an optimization routine (such as minimize()), then this needs to be set to TRUE, so that only the minimal CV risk is returned

Details

Fits an adaptive regression model using the estimated diffusion map coordinates of a data set, while holding epsilon fixed and optimizing over m. The adaptive regression model is the expansion of the response function on the first m diffusion map basis functions.

For a given epsilon value, this routine finds the optimal m by minimizing the cross-validation risk (CV MSE) of the regression estimate. To optimize over (epsilon,m), use the function `adapreg()`.

Default uses 10-fold cross-validation to choose the optimal model size. User may also supply a vector of fold allocations. For instance, sample(1:10,length(y),replace=T) does 10-fold CV while 1:length(y) does leave-one-out CV.

Value

The returned value is a list with components

`mincvrisk`	minimum cross-validation risk for the adaptive regression model for the given epsilon
`mopt`	size of the optimal regression model. If mopt equals mmax, it is advised to increase mmax.

`cvrisk`	vector of CV risk estimates for model sizes from 1:mmax
`epsilon`	value of epsilon used in diffusion map construction
`y.hat`	predictions of the response, y-hat, for the optimal model
`coeff`	coefficients of the optimal model

If objfun is set to TRUE, then the returned value is the minimum cross-validation risk for the adaptive regression model for the given epsilon.

References

Richards, J. W., Freeman, P. E., Lee, A. B., and Schafer, C. M., (2009), ApJ, 691, 32

See Also

`diffuse()`,`adapreg()`

Examples

```
library(stats)
library(scatterplot3d)
## trig function on circle
t=seq(-pi,pi,.01)
x=cbind(cos(t),sin(t))
y = cos(3*t) + rnorm(length(t),0,.1)
tcol = topo.colors(32)
colvec = floor((y-min(y))/(max(y)-min(y))*32); colvec[colvec==0] = 1
scatterplot3d(x[,1],x[,2],y,color=tcol[colvec],pch=20,
  main="Cosine function supported on circle",angle=55,
  cex.main=2,col.axis="gray",cex.symbols=2,cex.lab=2,
  xlab=expression("x"[1]),ylab=expression("x"[2]),zlab="y")

D = as.matrix(dist(x))
# leave-one-out cross-validation:
AR = adapreg.m(.01,D,y,fold=1:length(y))
print(paste("optimal model size:",AR$mopt,"; min. CV risk:",
  round(AR$mincvrisk,4)))
par(mfrow=c(2,1),mar=c(5,5,4,1))
plot(AR$cvrisks,typ='b',xlab="Model size",ylab="CV risk",
  cex.lab=1.5,cex.main=1.5,main="CV risk estimates")
plot(y,AR$y.hat,ylab=expression(hat("y")),cex.lab=1.5,cex.main=1.5,
  main="Predictions")
abline(0,1,col=2,lwd=2)

## swiss roll data
N=2000
t = (3*pi/2)*(1+2*runif(N));  height = runif(N);
X = cbind(t*cos(t), height, t*sin(t))
X = scale(X) + matrix(rnorm(N*3,0,0.05),N,3)
tcol = topo.colors(32)
colvec = floor((t-min(t))/(max(t)-min(t))*32); colvec[colvec==0] = 1
scatterplot3d(X,pch=18,color=tcol[colvec],xlab=expression("x"[1]),
  ylab=expression("x"[2]),zlab=expression("x"[3]),cex.lab=1.5,
  main="Swiss Roll, Noise = 0.05",cex.main=1.5,xlim=c(-2,2),
  ylim=c(-2,2),zlim=c(-2,2),col.axis="gray")

D = as.matrix(dist(X))
# 10-fold cross-validation:
AR = adapreg.m(.2,D,t,mmax=25,nfolds=5)
print(paste("optimal model size:",AR$mopt,"; min. CV risk:",
  round(AR$mincvrisk,4)))
par(mfrow=c(2,1),mar=c(5,5,4,1))
plot(AR$cvrisks,typ='b',xlab="Model size",ylab="CV risk",
  cex.lab=1.5,cex.main=1.5,main="CV risk estimates")
plot(t,AR$y.hat,ylab=expression(hat("t")),cex.lab=1.5,cex.main=1.5,
  main="Predictions")
abline(0,1,col=2,lwd=2)
```

`annulus` *Annulus toy data set*

Description

The annulus data frame has 1000 rows and 2 columns. 500 data points are from the noisy annulus and 500 data points reside within the annulus.

Usage

```
annulus
```

Format

Data are in two dimensions.

`Chainlink` *Chainlink toy clustering data set*

Description

The Chainlink data frame has 1000 rows and 3 columns. The data are of two interlocking 3-dimensional rings. 500 data points are from one ring and 500 from the other ring.

Usage

```
Chainlink
```

Format

The data are in 3 dimensions, C1, C2, and C3.

Source

http://www.uni-marburg.de/fb12/datenbionik/data?language_sync=1

References

Ultsch, A.: Clustering with SOM: U*C, In Proc. Workshop on Self-Organizing Maps, Paris, France, (2005) , pp. 75-82

---

diffuse *Compute diffusion map coordinates from pair-wise distances.*

Description

Uses the pair-wise distance matrix for a data set to compute the diffusion map coefficients. Computes the Markov transition probability matrix, and its eigenvalues and left & right eigenvectors. Returns a 'dmap' object.

Usage

```
diffuse(D, eps.val = epsilonCompute(D), neigen = NULL, t = 0,
  maxdim = 50, delta = 10^-5)
```

Arguments

`D`	n-by-n pairwise distance matrix for a data set with n points, or alternatively output from the dist() function
`eps.val`	epsilon parameter for the diffusion weight matrix, exp(-D$^2$/(eps.val)). Default is to use the epsilon corresponding to the median distance to the 0.01*n nearest neighbor
`neigen`	number of dimensions of final diffusion map representation. Default uses number of dimensions corresponding to a 95% drop-off in eigenvalue multiplier.
`t`	optional time-scale parameter in the diffusion map. The (recommended) default uses multiscale geometry.
`maxdim`	the maximum number of diffusion map dimensions returned if 95% drop-off is not attained.
`delta`	sparsity cut-off for the symmetric graph Laplacian. Default of 10^-5 is used. Higher value induces more sparsity in Laplacian (and faster computations)

Details

Diffusion map is a powerful tool for data parametrization that exploits the natural geometry of a data set. Diffusion map uses local interactions between data points, propogated to larger scales, to construct a global representation of the data.

The parameter eps.val controls the degree of localness in the diffusion weight matrix. For most statisitical inference problems using diffusion map, results should be optimized over eps.val. Generally a good starting point is to pick eps.val as $2*$med.knn$^2$, where med.knn is the median distance to the kth nearest neighbor, and k is chosen 1-2% of n. The default uses 1% of n.

Computation of the diffusion map coordinates requires singular value decomposition of the normalized graph Laplacian. This operation is optimized for speed by exploiting the sparseness of the graph Laplacian and by using ARPACK for fast matrix decomposition. Increasing the sparseness parameter, delta, will speed up the algorithm.

Value

The returned value is an object of 'class' 'diffuse'.

The function 'plot' is used to plot the diffusion coordinates in 1, 2, or 3 dimensions. The function 'print' displays the computed eigen-multipliers and the value of epsilon used.

An object of class 'dmap' is a list containing the following components:

`X`	matrix of n diffusion map coordinates, entered column-wise (does not include the trivial coordinate)
`phi0`	trivial left eigenvector of Markov matrix (stationary distribution of Markov random walk) in diffusion map construction
`eigenvals`	eigen-values of the svd of the symmetric graph Laplacian
`eigenmult`	eigen-multipliers of the diffusion map
`psi`	right eigenvectors of the Markov matrix (first row is the trivial right eigenvector)
`phi`	left eigenvectors of the Markov matrix (first row is the trivial left eigenvector)
`neigen`	number of diffusion map dimensions used
`epsilon`	the value of epsilon used

References

Coifman, R. R., & Lafon, S., (2006), Appl. Comput. Harmon. Anal., 21, 5

Lafon, S., & Lee, A., (2006), IEEE Trans. Pattern Anal. and Mach. Intel., 28, 1393

Richards, J. W., Freeman, P. E., Lee, A. B., Schafer, C. M., (2009), ApJ, 691, 32

Examples

```
library(stats)
## example with noisy spiral
n=2000
t=runif(n)^.7*10
al=.15;bet=.5;
x1=bet*exp(al*t)*cos(t)+rnorm(n,0,.1)
y1=bet*exp(al*t)*sin(t)+rnorm(n,0,.1)
plot(x1,y1,pch=20,main="Noisy spiral")
D = dist(cbind(x1,y1))
dmap = diffuse(D,neigen=10) # compute diffusion map
par(mfrow=c(2,1))
plot(t,dmap$X[,1],pch=20,axes=FALSE,xlab="spiral parameter",ylab="1st diffusion coefficient")
```

```
box()
plot(1:10,dmap$eigenmult,typ='h',xlab="diffusion map dimension",ylab="eigen-multipliers")

## example with annulus data set
data(annulus)
plot(annulus,main="Annulus Data",pch=20,cex=.7)
D = dist(annulus) # use Euclidean distance
dmap = diffuse(D,eps.val=.1) # compute diffusion map & plot
print(dmap)
plot(dmap)
```

diffusionKmeans *Diffusion K-means*

Description

Clusters a data set based on its diffusion coordinates.

Usage

```
diffusionKmeans(dmap, K, params = c(), Niter = 10, epsilon = 0.001)
```

Arguments

dmap	a '"dmap"' object, computed by diffuse()
K	number of clusters
params	optional parameters for each data point. Entry can be a vector of length n, or a matrix with n rows. If this argument is given, cluster centroid parameters are returned.
Niter	number of K-means iterations performed.
epsilon	stopping criterion for relative change in distortion for each K-means iteration

Details

A '"dmap"' object computed by diffuse() is the input, so diffuse() must be performed first. Function is written this way so the K-means parameters may be varied without having to recompute the diffusion map coordinates in each run.

Diffusion K-means is a special form of spectral clustering. It is a unique algorithm because the eigenvectors of the symmetric Laplacian are weighted in such a way to guarantee that Euclidean distance in diffusion space will be approximately equal to the diffusion distance between objects. Clustering by Euclidean distance in diffusion space exploits this fact.

Value

The returned value is a list with components

`part`	final labelling of data from K-means. n-dimensional vector with integers between 1 and K
`cent`	K geometric centroids found by K-means
`D`	minimum of total distortion (loss function of K-means) found across K-means runs
`DK`	n by k matrix of squared (Euclidean) distances from each point to every centroid for the optimal K-means run
`centparams`	optional parameters for each centroid. Only returned if params is specified in the function call. Is a matrix with k rows.

References

Lafon, S., & Lee, A., (2006), IEEE Trans. Pattern Anal. and Mach. Intel., 28, 1393

Richards, J. W., Freeman, P. E., Lee, A. B., and Schafer, C. M., (2009), ApJ, 691, 32

Richards, J. W., Freeman, P. E., Lee, A. B., Schafer, C. M., (2009), MNRAS, Volume 399, Issue 2, pp. 1044-1057

See Also

`diffuse()`

Examples

```
library(scatterplot3d)

## example with annulus data set
data(annulus)
par(mfrow=c(2,1))
plot(annulus,main="Annulus Data",pch=20,cex=.7)
D = dist(annulus) # use Euclidean distance
dmap = diffuse(D,eps.val=0.05) # compute diffusion map
k=2  # number of clusters
dkmeans = diffusionKmeans(dmap, k)
plot(annulus,main="Colored by diffusion K-means clustering",pch=20,
   cex=.7,col=dkmeans$part)
table(dkmeans$part,c(rep(1,500),rep(2,500)))


## example with Chainlink data set
data(Chainlink)
lab.col = c(rep("red",500),rep("blue",500)); n=1000
scatterplot3d(Chainlink$C1,Chainlink$C2,Chainlink$C3,color=lab.col,
   main="Chainlink Data") # plot Chainlink data
D = dist(Chainlink) # use Euclidean distance
dmap = diffuse(D,neigen=3,eps.val=.01) # compute diffusion map & plot
plot(dmap)
```

```
dkmeans = diffusionKmeans(dmap, K=2)
col.dkmeans=ifelse(dkmeans$part==1,"red","blue")
scatterplot3d(Chainlink,color=col.dkmeans,
   main="Chainlink Data, colored by diff. K-means class")
table(dkmeans$part,lab.col)
```

`distortionMin` *Distortion Minimization via K-means*

Description

Runs one K-means loop based on the diffusion coordinates of a data set, beginning from an initial set of cluster centers.

Usage

```
distortionMin(X, phi0, K, c0, epsilon = 0.001)
```

Arguments

`X`	diffusion coordinates, each row corresponds to a data point
`phi0`	trivial left eigenvector of Markov matrix (stationary distribution of Markov random walk) in diffusion map construction
`K`	number of clusters
`c0`	initial cluster centers
`epsilon`	stopping criterion for relative change in distortion

Details

Used by diffusionKmeans().

Value

The returned value is a list with components

`S`	labelling from K-means loop. n-dimensional vector with integers between 1 and K
`c`	K geometric centroids found by K-means
`D`	minimum of total distortion (loss function of K-means) found in K-means run
`DK`	n by k matrix of squared (Euclidean) distances from each point to every centroid

References

Lafon, S., & Lee, A., (2006), IEEE Trans. Pattern Anal. and Mach. Intel., 28, 1393

See Also

`diffusionKmeans()`

Examples

```
data(annulus)
n = dim(annulus)[1]
D = dist(annulus) # use Euclidean distance
dmap = diffuse(D,0.03) # compute diffusion map
km = distortionMin(dmap$X,dmap$phi0,2,dmap$X[sample(n,2),])
plot(annulus,col=km$S,pch=20)
table(km$S,c(rep(1,500),rep(2,500)))
```

`epsilonCompute` *Compute default diffusion map epsilon.*

Description

Uses the pair-wise distances to estimate a diffusion map epsilon value by the median p*n-th nearest neighbor

Usage

```
epsilonCompute(D, p = 0.01)
```

Arguments

`D`	n-by-n pairwise distance matrix for a data set with n points, or alternatively output from the dist() function
`p`	distances to p*n-th nearest neighbor are used. Default value is .01

Details

Function is used as the default value in diffuse(). For inference problems, it is advised that the results be optimized over epsilon.

Value

`epsilon`	value of epsilon to be used in diffusion map

See Also

`diffuse()`

Examples

```
data(annulus)
D = dist(annulus) # use Euclidean distance
epsilonCompute(D,.005)
epsilonCompute(D,.01)
epsilonCompute(D,.05)
epsilonCompute(D,.1)
```

---

nystrom — *Perform Nystrom Extension to estimate diffusion coordinates of data.*

Description

Given the diffusion map coordinates of a training data set, estimates the diffusion map coordinates of a new set of data using the pairwise distance matrix from the new data to the original data.

Usage

```
nystrom(dmap, Dnew, sigma = dmap$epsilon)
```

Arguments

`dmap`	a '"dmap"' object from the original data set, computed by diffuse()
`Dnew`	distance matrix between each new data point and every point in the training data set. Matrix is m-by-n, where m is the number of data points in the new set and n is the number of training data points
`sigma`	scalar giving the size of the Nystrom extension kernel. Default uses the tuning parameter of the original diffusion map

Details

Often, it is computationally infeasible to compute the exact diffusion map coordinates for large data sets. In this case, one may use the exact diffusion coordinates of a training data set to extend to a new data set using the Nystrom approximation.

A Gaussian kernel is used: exp(-D(x,y)^2/sigma). The default value of sigma is the epsilon value used in the construction of the original diffusion map. Other methods to select sigma, such as Algorithm 2 in Lafon, Keller, and Coifman (2006) have been proposed.

The dimensionality of the diffusion map representation of the new data set will be the same as the dimensionality of the diffusion map constructed on the original data.

Value

The estimated diffusion coordinates for the new data, a matrix of dimensions m by p, where p is the dimensionality of the input diffusion map

References

Freeman, P. E., Newman, J. A., Lee, A. B., Richards, J. W., and Schafer, C. M. (2009), MNRAS, Volume 398, Issue 4, pp. 2012-2021

Lafon, S., Keller, Y., and Coifman, R. R. (2006), IEEE Trans. Pattern Anal. and Mach. Intel., 28, 1784

See Also

`diffuse()`

Examples

```
library(stats)
Norig = 1000
Next = 4000
t=runif(Norig+Next)^.7*10
al=.15;bet=.5;
x1=bet*exp(al*t)*cos(t)+rnorm(length(t),0,.1)
y1=bet*exp(al*t)*sin(t)+rnorm(length(t),0,.1)

D = as.matrix(dist(cbind(x1,y1)))
Dorig = D[1:Norig,1:Norig] # training distance matrix
DExt = D[(Norig+1):(Norig+Next),1:Norig] # new data distance matrix
# compute original diffusion map
dmap = diffuse(Dorig,neigen=2)
 # use Nystrom extension
dmapExt = nystrom(dmap,DExt)
plot(dmapExt[,1:2],pch=8,col=2,
  main="Diffusion map, black = original, red = new data",
  xlab="1st diffusion coefficient",ylab="2nd diffusion coefficient")
points(dmap$X[,1:2],pch=19,cex=.5)
```

Index