Package 'modelcf'

February 2, 2011

Type Package

Version 2.1

Date 2011-02-02

Title Modeling physical computer codes with functional outputs using clustering and dimensionality reduction

Author Benjamin Auder

Maintainer Benjamin Auder <Benjamin.Auder@gmail.com>

Depends R (>= 2.10.1)

Suggests class, rpart, e1071, kernlab, randomForest, klaR, wmtsa,mlegp, splines

Description Statistical learning with vectorial inputs and smooth 1D curves as outputs. The main function builds a model from n samples (x_i,y_i).

License GPL (>= 3)

LazyLoad yes

Archs modelcf.so

R topics documented:

`classification` *Building a classifier*

Description

`learnClassif` builds a classifier object (see code for details).

`optimParams_classif` optimize parameters for the chosen method.

These two methods should not be called directly. Using the specific technique inside its own package is a better idea.

Usage

```
learnClassif(x, y, method, params)

optimParams_classif(x, y, method, k, trcv)
```

Arguments

`x`	matrix of `n` input vectors in rows. `x[i,]` is the i-th p-dimensional input
`y`	matrix of `n` outputs in rows. `y[i,]` is the i-th m-dimensional output
`method`	classification method, to be chosen between "kNN" (k-nearest-neighbors), "ctree" (classification trees), "RDA" (Regularized Discriminant Analysis), "rforest" (random forests), "SVM" (Support Vector Machines)
`params`	vector of parameters for the chosen method
`k`	fixed number of neighbors at each point to build the training set in cross-validation procedure
`trcv`	fraction of total examples on which a model is trained during cross-validation procedure.

Value

`learnClassif` returns a classifier object (internal specifications).

`optimParams_classif` returns a vector of optimized parameters for the chosen method.

clustering *Main clustering functions*

Description

phclust performs R hierarchical cluster (using hclust()) with Ward linkage, and call cutree() after.

This function should not be called directly. Use the following one instead.

gtclusts main function to cluster data according to any method.

findK_gtclusts is a procedure to determine the number of classes (and associate partitioning).

gtclusts_inout calls the previous method one on outputs, and then on each inputs cluster (main procedure).

Usage

```
phclust(dissims, K)

gtclusts(method, data, K, d=min(5,ncol(data)), adn="none", knn=0,
symm=TRUE, weight=FALSE, sigmo=FALSE)

findK_gtclusts(x, y, method, d=min(10, ncol(x)), adn="none", knn=0,
symm=TRUE, weight=FALSE, sigmo=FALSE, minszcl=30,
maxcl=Inf, mclass="kNN", taus=0.8, Ns=10, tauc=0.8, Nc=10,
trcv=0.7, nstagn=10)

gtclusts_inout(x, y, method, d=min(10, ncol(x)), redy=FALSE, adn="none",
knn=0, symm=TRUE, weight=FALSE, sigmo=FALSE, minszcl=30,
maxcl=Inf, mclass="kNN", taus=0.8, Ns=10, tauc=0.8, Nc=10,
trcv=0.7, verb=TRUE, nstagn=10)
```

Arguments

method	the clustering method, to be chosen between “HDC” (k-means based on Hitting Times), “CTH” (Commute-Time Hierarchic), “CTKM” (Commute-Time k-means), “spec” (spectral clustering), “CH” (hierarchical clustering), “PCA” (PCA-k-means from Chiou and Li ; see references), “KM” (basic k-means)
data	matrix of n vectors in rows ; data[i,] is the i-th m-dimensional vector
x	matrix of n input vectors in rows. x[i,] is the i-th p-dimensional input
y	matrix of n discretized outputs in rows. y[i,] is the i-th D-dimensional output
dissims	matrix of dissimilarities (can be simple L2 distances, or more complicated like commute-times)
K	expected number of clusters
d	estimated (real) outputs dimensionality (should be far less than D) ; useful only if one of the following parameters is set: redy,adn,method=="ACP". It can be estimated using functions from dimension file
adn	string for adapted point-varying neighborhoods. "none" for no adaptivity, "adbas" for simple local PCA based neighborhoods (see code), "ad1" for the Zhan et al. method, and "ad2" for Wang et al. method. In short, the more linear data is around x, the more x has neighbors

knn	fixed number of neighbors at each point ; used only if adn=="none". If zero, a simple heuristic will determine it around sqrt(nrow(data))
symm	boolean at TRUE for symmetric similarity matrix (see code. It does not impact much the result
weight	boolean at TRUE for weighted hitting/commute times, like in the article of Liben-Nowell and Kleinberg
sigmo	boolean at TRUE for sigmoid commute-time kernel, like in the article of Yen et al.
redy	boolean telling if the outputs should be reduced (with PCA) as a preprocessing step
minszcl	minimum size for a cluster. This is interesting to not allow too small clusters for the regression stage ; recommanded values are above 30-50
maxcl	maximum number of clusters ; Inf stands for "no limit", i.e. determined by stability-prediction loops only
mclass	type of classifier to use in the prediction accuracy step ; choice between "kNN" (k-nearest-neighbors), "ctree" (classification tree), "RDA" (Regularized Discriminant Analysis), "rforest" (random forests), "SVM" (Support Vector Machines). Only the first two were intensively tested
taus	threshold for stability check ; value between 0 (every method accepted) and 1 (only ultra-stable method accepted). Recommanded between 0.6 and 0.9
Ns	number of stability runs before averaging results (the higher the better, although slower..)
tauc	threshold for prediction accuracy check (after subsampling) ; value between 0 (every clustering accepted) and 1 (only "well separated" clusters accepted). Recommanded between 0.6 and 0.9
Nc	number of partitions predictions runs before averaging results (same remark as for Ns above)
trcv	fraction of total examples on which a model is trained during cross-validation procedures.
verb	TRUE for printing what is going on. A further release will allow to choose levels of verbosity.
nstagn	number of allowed stages (increasing the number of clusters K) without added clusters (if minszcl is large enough small clusters may end being merged).

Details

adn should not be set when working with small datasets and/or in low dimension (<= 3).

When sigmo is set, the sigmoid commute-time kernel (Yen et al.) is computed with a=1. In the paper authors say it need manual tuning.

The algorithm for simultaneous estimate of K and clustering works in two main steps :

1. subsample original data in data1 and data2, then cluster both, and measure similarity between partitions at the intersection using the variation of information index of Meila article.
2. subsample a training set Tr in [1,n] where n is the number of data rows, then subsample a set S which must contain [1,n] \ Tr. Cluster both sets, and use Tr to predict labels of the testing set. Finally compare the partitions using simple "matching counter" after renumbering (with the hungarian algorithm).

Both are repeated Ns, Nc times to get accurate estimators. We stop when these estimators fall below the thresholds taus, tauc, and return corresponding partition.

Value

An integer vector describing classes (same as `kmeans()$cluster` field).

References

J-M. Chiou and P-L. Li, **Functional clustering and identifying substructures of longitudinal data**, in Journal of the Royal Statistical Society 69(4): 679-699, 2007

L. Yen, D. Vanvyve, F. Wouters, F. Fouss, M. Verleysen and M. Saerens, **Clustering using a random-walk based distance measure**, at Symposium on Artificial Neural Networks 13: 317-324, Bruges, Belgium, 2005

L. Yen, F. Fouss, C. Decaestecker, P. Francq and M. Saerens, **Graph nodes clustering with the sigmoid commute-time kernel: A comparative study**, in Data \& Knowledge Engineering 68(3): 338-361, 2009

Examples

```
#generate a mixture of three gaussian data sets
data = rbind( matrix(rnorm(200,mean=2,sd=0.5),ncol=2),
    matrix(rnorm(200,mean=4,sd=0.5),ncol=2),
    matrix(rnorm(200,mean=6,sd=0.5),ncol=2) )
#cluster it using k-means
km = gtclusts("KM", data, 3)
#and using Commute-Time Hierarchic clustering
ct = gtclusts("CTH", data, 3, k=20, symm=FALSE)
#plot results
plotPts(data, cl=km)
plotPts(data, cl=ct)

#generate a (smaller) mixture of three gaussian data sets
inData = rbind( matrix(rnorm(60,mean=2,sd=0.5),ncol=2),
    matrix(rnorm(60,mean=4,sd=0.5),ncol=2),
    matrix(rnorm(60,mean=6,sd=0.5),ncol=2) )
#build artificial corresponding outputs
sPoints = seq(from=0,to=2*pi,by=2*pi/200)
cosFunc = cos(sPoints)
sinFunc = sin(sPoints)
outData = as.matrix(inData[,1]) %*% cosFunc + as.matrix(inData[,2]^2) %*% sinFunc
#partition inputs-outputs using Commute-Time Hierarchic clustering
ct = gtclusts_inout(inData, outData, "CTH", k=20, minszcl=20, mclass="kNN",
    taus=0.7, Ns=10, tauc=0.7, Nc=10)
#plot results, inputs then outputs
plotPts(inData, cl=ct)
plotC(outData, cl=ct)
```

---

comparts *Comparing partitions (clustering)*

Description

`checkParts` is an assymetric measure of the matching of P relatively to P_ref.

The two next indices are symmetric.

`varInfo` computes the variation of information index from Meila article.

countPart is a simple counter of matched elements, e.g. the matching level of (1,1,1,2) and (1,1,2,3) is 2.

Usage

```
checkParts(P, P_ref)

varInfo(P1, P2)

countPart(P1, P2)
```

Arguments

P,P_ref,P1,P2
: a partition of some data, as outputs by gtclusts ; e.g., (1,1,1,1,2,2,2,2,1,1,3,3,3)

Details

All indices are normalized to lie in the range (0,1).

The checkParts method uses P clusters overlap over P_ref ones to compute an adequation index. It is quite severe, designed for testing of clustering methods.

The "variation of information" index of Meila is a (mathematical) measure between partitions. This is actually a nice property ; see article.

Value

A real number between 0 and 1, indicating the matching level between the two partitions.

References

M. Meila, **Comparing Clusterings**, Statistics Technical Report 418, University of Washington, 2002

Examples

```
#comparing the three indices
P = c(1,1,2,2,2,2,2,2,3,3,3,4,4,4,1,1)
P_ref = c(1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4)
print(checkParts(P, P_ref))
print(varInfo(P, P_ref))
print(countPart(P, P_ref))
```

---

connexity *Functions around graph conexity*

Description

Internal use only ; should not be called by the final user.

gt_cxcomps gets the connected components based on neighborhoods.

testConnexity returns neighborhoods that assure graph (weak) connexity. If NI does not lead to connexity, kNN-graph is built instead.

Usage

```
gt_cxcomps(NI, ctype=FALSE, k=0)

testConnexity(data, NI, k)
```

Arguments

`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`NI`	list of (graph) neighborhoods
`k`	fixed number of neighbors at each point ; used only if `adn == FALSE`
`ctype`	(weak) connexity type, TRUE for mutual-kNN graph (clustering case), FALSE for general graphs (dimensionality reduction)

Value

`gt_cxcomps` returns a vector like (1 1 1 2 2 2 1 1 3 3 3) describing connected components.

`testConnexity` returns the smallest list of neighborhoods that ensure connexity.

`dataSets` *Two artificial data sets*

Description

The first artificial dataset is generated by the function :

f1 = function(t)

{ return(0.8*atan(a*t) + exp(b*(-4-t)+1) + log(c*(4-t)+1)) }

and the second one by :

f2 = function(t)

{ return(0.8*atan(a*(4-t)) + exp(b*(-4+t)+1) + log(c*(4+t)+1)) }

for a, b, c varying uniformly in [0,4] or [0,7].

The matrix `dataIn` contains the input parameters a, b, c in rows, while the matrices `dataOut1` and `dataOut2` are filled with the corresponding curves in rows (200 sample points).

Usage

```
datacf
```

Format

Matrices with 300 rows, and 3 columns for `dataIn`, 200 for the others (sample points).

`dimension` *Dimension estimation*

Description

`dest_PCA` and `dest_clust` use the PCA locally to estimate dimension, and then average the results. The second one determine local regions using k-means clustering (Bruske and Sommer, 1998), while the first can be considered as a fast suboptimal version of the algorithm by Fan et al. (2010).

`dest_pett`, `dest_fara`, `dest_levi` and `dest_unbl` estimate the intrinsic dimension of `data`, following respectively the algorithm of Pettis et al. (1979), Farahmand et al. (2007), Levina and Bickel (2005) and a debiased version of this last one, by MacKay and Ghahramani. All these methods are based on a relation between the dimension and density of data.

`dest_RML` and `dest_rgrl` implement an idea from the papers of Lin et al. (2006), using "non flat" simplices to evaluate dimension. The second one is a regularization attempt, which has not proven effective yet. `dest_sliv` implements a variation on an idea by Cheng and Chiu (2009), simplified although heavier to run.

Usage

```
dest_PCA(data, k, thvar=0.01)

dest_clust(data, nclusts, thvar=0.01)

dest_pett(data, kmax)

dest_levi(data, k)

dest_unbl(data, k)

dest_fara(data, k)

dest_RML(data, kmax, N=10, tsoft=0.0)

dest_rgrl(data, kmax, N=10, alpha=3)

dest_sliv(data, k, N=10000, thtest=0.05)
```

Arguments

`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`k`	fixed number of neighbors at each point
`thvar`	expected threshold on explained variance (between 0 and 1 ; should be close to 0)
`nclusts`	number of cells to be obtained by the k-means algorithm
`kmax`	maximum number of neighbors at each point
`N`	number of Monte-Carlo loops
`tsoft`	tolerance factor for the visibility graph computation (between 0 and 1 ; should be close to 0)

`alpha`	regularization parameter ; a positive value which give (decreasing) weighted emphasize to increasing values of `tsoft`. When `alpha` becomes large, the weights associated with "high" values of `tsoft` fade rapidly
`thtest`	threshold on the p-value of the statistical test for densities adequation

Value

An integer equals to the estimated dimension.

References

J. Bruske and G. Sommer, **Intrinsic dimension estimation with optimally topology preserving maps**, in IEEE Transactions on Pattern Analysis and Machine Intelligence 20: 572-575, 1998

M. Fan, N. Gu, H. Qiao and B. Zhang, **Intrinsic dimension estimation of data by principal component analysis**, submitted for publication, 2010.

K. W. Pettis, T. A. Bailey, A. K. Jain and R. C. Dubes, **An Intrinsic Dimensionality Estimator from Near-Neighbor Information**, in IEEE Transactions on Pattern Analysis and Machine Intelligence 1 (1): 25-37, 1979

A. M. Farahmand, C. Szepesvari and J-Y. Audibert, **Manifold-adaptive dimension estimation**, at 24th International Conference on Machine Learning 227: 265-272, 2007

E. Levina and P. J. Bickel, **Maximum Likelihood Estimation of Intrinsic Dimension**, in Advances in Neural Information Processing Systems 17: 777–784, 2005

D. J. MacKay and Z. Ghahramani, Comments on "Maximum Likelihood Estimation of Intrinsic Dimension" by E. Levina and P. Bickel (2004), http://www.inference.phy.cam.ac.uk/mackay/dimension/, 2005

T. Lin, H. Zha and S. U. Lee, **Riemannian Manifold Learning for Nonlinear Dimensionality Reduction**, at European Conference on Computer Vision, Graz, Austria 9: 44-55, 2006

S-W. Cheng and M-K. Chiu, **Dimension detection via slivers**, at 20th Annual ACM-SIAM Symposium on Discrete Algorithms: 1001-1010, 2009

J. M. Lee and M. Verleysen, **Nonlinear Dimensionality Reduction** (chapter 3), Springer, 2007

Examples

```
#generate a swissroll dataset
n = 300 ; h = 3
phi = runif(n, min=0, max=2*pi)
z = runif(n, min=0, max=h)
sw = cbind( phi*cos(phi), phi*sin(phi), z )

#estimate dimension
print(dest_PCA(sw, 20))
print(dest_unbl(sw, 20))
print(dest_pett(sw, 20))
print(dest_fara(sw, 20))
print(dest_levi(sw, 20))
print(dest_unbl(sw, 20))
    print(dest_RML(sw, 20, 10))
    print(dest_rgrl(sw, 20, 10))
    print(dest_sliv(sw, 20, 10))
```

errors *Empirical error estimators*

Description

`fperrors` estimates the error of a model on a specific testing set. It computes MSE errors indicators, by comparing predictions to real curves.

Usage

```
fperrors(ypred, yreal, mntrain)
```

Arguments

`ypred`	matrix of the predicted functions in rows (D sample points / columns)
`yreal`	matrix of the expected functions (same format as ypred above)
`mntrain`	mean curve of training outputs

Value

A list with MSE values for the model, and the constant estimator (equals to the training mean). The corresponding attributes are named respectively "MSE" and "pvar".

Examples

```
#get the first artificial dataset and build a standard model of it
#using 250 training samples
data(datacf)
trainInds = sample(1:300, 250)
m = fmetam(dataIn[trainInds,],dataOut1[trainInds,],d=3,wcl=FALSE,mdim="linear")
#get the predicted curves and errors
pred = predict.modelcf(m, dataIn[-trainInds,])
errs = fperrors(pred,dataOut1[-trainInds,],colMeans(dataOut1[trainInds,]))
#plot the MSE and Q2 error curves
plot(errs$MSE, type="l", ylab="MSE")
plot(1-errs$MSE/errs$pvar, type="l", ylim=c(0,1), ylab="Q2")
```

kmeans *k-means like functions*

Description

`km_dists` = k-means based on a distance matrix.

`km_PCA` = generalization of classical k-means for functional case, by Chiou and Li.

Usage

```
km_dists(distm, K, nstart=10, maxiter=100)

km_PCA(data, K, d, simplif=TRUE, maxiter=50)
```

Arguments

`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`distm`	matrix of distances (can be simple L2 distances, or more complicated like commute-times)
`K`	expected number of clusters
`d`	estimated data dimension (e.g. through functions from `dimension` file)
`simplif`	boolean at `TRUE` for simplified algorithm, without leave-one-out SVD's (actually very costly)
`nstart`	number of algorithm runs with random initialization
`maxiter`	maximum number of iterations within one algorithm run

Details

The k-means using a distances matrix is exactly the same algorithm as classical k-means, except for the choice of centroids, which must belong to the dataset.

The PCA-k-means algorithm replaces the centroids by centroids **plus** local basis functions obtained by (functional) PCA. The closeness to a cluster is computed relatively to this full system, instead of a centroid only. Apart from this point, the algorithm is similar to k-means ; but more general. The `simplif` argument allows or not a simplification avoiding very costly leave-one-out procedure, (re)computing local basis after slight data change. It can be switched off without fears for big enough clusters (say, more then a few dozens).

Value

An integer vector describing classes (same as `kmeans()$cluster` field).

References

J-M. Chiou and P-L. Li, **Functional clustering and identifying substructures of longitudinal data**, in Journal of the Royal Statistical Society 69(4): 679-699, 2007

Examples

```
#generate a mixture of three gaussian data set, and compute distances
data = rbind( matrix(rnorm(200,mean=2,sd=0.5),ncol=2),
    matrix(rnorm(200,mean=4,sd=0.5),ncol=2),
    matrix(rnorm(200,mean=6,sd=0.5),ncol=2) )
dists = as.matrix(dist(data))
#cluster using k-means
km = km_dists(dists, 3)
#plot result
plotPts(data, cl=km)
#and using km_PCA clustering after artificial functional transformation
sPoints = seq(from=0,to=2*pi,by=2*pi/200)
cosFunc = cos(sPoints)
sinFunc = sin(sPoints)
fdata = as.matrix(data[,1]) %*% cosFunc + as.matrix(data[,2]^2) %*% sinFunc
kp = km_PCA(fdata, 3, 2)
#plot result
plotC(fdata, cl=kp)
```

LPcaML *Local PCA Manifold Learning*

Description

LPcaML embeds data in the d-dimensional space using the Local PCA Manifold Learning method from the Zhan et al. article.

LPcaML_rec inverses the above procedure, reconstructing a curve (or any high dimensional vector) from its low-dimensional representation.

Usage

```
LPcaML(data, d, adn="none", k=0, alpha=0.5, trcv=0.7)

LPcaML_rec(LPout, newEmb)
```

Arguments

data	matrix of n vectors in rows ; data[i,] is the i-th m-dimensional vector
d	estimated data dimension (e.g. through functions from dimension file)
adn	string for adapted point-varying neighborhoods. "none" for no adaptivity, "adbas" for simple local PCA based neighborhoods (see code), "ad1" for the Zhan et al. method, and "ad2" for Wang et al. method. In short, the more linear data is around x, the more x has neighbors
k	fixed number of neighbors at each point (used only if adn==FALSE). If zero, a simple heuristic will determine it around sqrt(nrow(data))
alpha	fraction of overlapping elements when building the traversal sequence of neighborhoods
trcv	fraction of total examples on which a model is trained during cross-validation procedures
LPout	an object as output by LPcaML function
newEmb	a new embedding from which the high dimensional object has to be estimated

Details

The algorithm works in two main steps :

1. A traversal sequence of (overlapping) local neighborhoods is constructed, and a simple PCA is computed in each neighborhood.
2. The reduced coordinates are then computed step by step, by optimizing an affine transformation matrix on the overlap between two neighborhoods.

The reconstruction function LPcaML_rec first find the right neighborhood, then apply inverse affine transformation. For better explanations, see the article.

Value

A list with the embedding in $embed, and some technical parameters for reconstruction.

References

Y. Zhan, J. Yin, G. Zhang and E. Zhu, **Incremental Manifold Learning Algorithm Using PCA on Overlapping Local Neighborhoods for Dimensionality Reduction**, at 3rd International Symposium on Advances in Computation and Intelligence 5370: 406-415, 2008

Examples

```
#generate a swissroll dataset
n = 300 ; h = 3
phi = runif(n, min=0, max=2*pi)
z = runif(n, min=0, max=h)
#::set colors
rSize = 64
r = rainbow(rSize)
cols = r[pmin(floor((rSize/(2.0*pi))*phi)+1,rSize)]
#end set colors::
sw = cbind( phi*cos(phi), phi*sin(phi), z )

#launch algorithm and visualize result
emb = LPcaML(sw, 2, alpha=0.7)$embed
plotPts(emb, cl=cols)
```

---

mixpred *Mixing functional models*

Description

Functions to define a mixture of already created models.

`getcoefc` returns a curve matching the maximums given by user (to facilitate models mixing).

`mixpredf` takes several models as arguments, and mix them after calling `predict.modelcf`. This allows to benefit from different kinds of models.

Usage

```
getcoefc(D, inds, maxs=c(), rgs=c())

mixpredf(mods, coefs, x, verb = FALSE)
```

Arguments

`D`	outputs dimensionality (usually a few hundreds)
`inds`	(strictly) positive integer vector of desired local maximums locations
`maxs`	positive real vector of desired local maximums amplitudes
`rgs`	minimum number of neighbors at each point
`mods`	a list of **modelcf** models, outputs of `fmetam`
`coefs`	a list of curves (same length as training outputs), which are taken as mixture coefficients (see details below)
`x`	matrix of n testing input vectors in rows ; `x[i,]` is the i-th m-dimensional testing input vector
`verb`	`TRUE` for printing what is going on. A further release will allow to choose levels ov verbosity.

Details

`getcoefc` outputs a piecewise constant function, which locally constant parts are centered around indices given in `inds`. The (integer) width of each locally constant part is given by the `rgs` vector argument ; if not provided, the width is taken constant, equals to the maximum value which avoid overlapping. `maxs` indicates the amplitude of each local maximum (piecewise constant), and will equals `(1,1,1,1,...)` if not provided.

Value

`getcoefc` returns a sampled curve (with `D` values).

`mixpredf` returns a model prediction (matrix with curves in rows) ; same output format as `predict.modelcf`.

Examples

```
#get the first artificial dataset and build three different models of it
#using 250 training samples
data(datacf)
trainInds = sample(1:300, 250)
m1 = fmetam(dataIn[trainInds,], dataOut1[trainInds,], d=3, wcl=FALSE,
    mdim="linear")
m2 = fmetam(dataIn[trainInds,], dataOut1[trainInds,], d=3, wcl=FALSE,
    mdim="RML", kmin=15, kmax=25)
m3 = fmetam(dataIn[trainInds,], dataOut1[trainInds,], d=3, wcl=FALSE,
    mreg="fkNN")
#mix the three, giving \dQuote{first third} weight to the first,
#\dQuote{second third} weight to the second
#and \dQuote{third third} weight to the third one
mix = mixpredf(list(m1,m2,m3), list(c(rep(1,66),rep(0,134)),
    c(rep(0,66),rep(1,67),rep(0,67)),c(rep(0,133),rep(1,67))),
    dataIn[-trainInds,], verb=TRUE)
#plot the (L1) error between real and predicted curves
plotC(dataOut1[-trainInds,] - mix)
```

`modelcf` *package modelcf*

Description

This package contains a generic way to build surrogate models of physical computer codes, when inputs are vectors (in R^p) and outputs (continuous) curves from [a,b] to R. The curves are discretized on a finite grid t1,...,tD.

Note

This work was supported by the CEA Cadarache, where I was employed three years for my PhD. I would also like to thank Bertrand Iooss and Michel Marques for all their suggestions of improvement, and all the people who helped me, directly or not, for creating this package.

See Also

`fmetam`, `predict.modelcf`, `mixpredf`, `nfoldcv`.

`modeling` *Build and validate the functional outputs model*

Description

`fmetam_1cl` is a subroutine to do the dimensionality reduction step. Internal use only.

`fmetam` is the main method to build a model, using clustering and dimensionality reduction.

`fm_resids` will call `fmetam` twice, first on basic data and then on residuals for a better fit.

`nfoldcv` builds and tests several models with fixed parameters and ramdomly generated training sets (cross-validation).

Usage

```
fmetam_1cl(x, y, d, mdim, adnRD, knnRD, linbt, filt, wvar, alpha,
kmin, kmax, tsoft, thlvl, hdth, advhr, mreg, ppts, stred, trcv, verb)

fmetam(x, y, d=0, mclust="CTH", mclass="kNN", redy=FALSE, adnCC="none",
knnCC=0, wcl=TRUE, iclusts=rep(0,nrow(y)), symm=TRUE, weight=FALSE,
sigmo=FALSE, minszcl=30, maxcl=Inf, taus=0.8, Ns=10, tauc=0.8, Nc=10,
mdim="linear", adnRD="none", knnRD=0, linbt="PCA", filt="haar",
wvar=TRUE, alpha=0.5, kmin=0, kmax=0, tsoft=0.1, thlvl=0.3, hdth=2,
advhr=FALSE, mreg="PPR", ppts=FALSE, stred=TRUE, trcv = 0.7, verb = TRUE)

fm_resids(x, y, d=0, mclust="CTH", mclass="kNN", redy=FALSE, adnCC="none",
knnCC=0, wcl=TRUE, iclusts=rep(0,nrow(y)), symm=TRUE, weight=FALSE,
sigmo=FALSE, minszcl=30, maxcl=Inf, taus=0.8, Ns=10, tauc=0.8, Nc=10,
mdim1="linear", mdim2="RML", adnRD="none", knnRD=0, linbt="PCA",
filt="haar", wvar=TRUE, alpha=0.5, kmin=0, kmax=0, tsoft=0.1, thlvl=0.3,
hdth=2, advhr=FALSE, mreg1="PPR", mreg2="PPR", ppts=FALSE, stred=TRUE,
trcv = 0.7, verb = TRUE)

nfoldcv(x, y, d=0, single=TRUE, mclust="CTH", mclass="kNN", redy=FALSE,
adnCC="none", knnCC=0, wcl=TRUE, symm=TRUE, weight=FALSE, sigmo=FALSE,
minszcl=30, maxcl=Inf, taus=0.8, Ns=10, tauc=0.8, Nc=10,
mdim1="linear", mdim2="RML", adnRD="none", knnRD=0, linbt="PCA", filt="haar",
wvar=TRUE, alpha=0.5, kmin=0, kmax=0, tsoft=0.1, thlvl=0.3, hdth=2,
advhr=FALSE, mreg1="PPR", mreg2="PPR", ppts=FALSE, stred=TRUE, trcv = 0.7,
loo = FALSE, nfold=100, nhold=10, verb = TRUE, plotc=TRUE)
```

Arguments

`x`	matrix of `n` input vectors in rows, given as a R matrix or filename. `x[i,]` is the i-th p-dimensional input
`y`	matrix of `n` discretized outputs in rows, given as a R matrix or filename. `y[i,]` is the i-th D-dimensional output
`d`	estimated (real) outputs dimensionality (should be far less than `D`)
`single`	boolean telling if the model will be composite (base + residuals)

`mdim, mdim1, mdim2`
: the dimensionality reduction method (to be) used for base model (1) or residuals (2) : choice between “linear” for orthonormal basis, “RML” for Riemannian Manifold Learning, “LPcaML” for Local PCA Manifold Learning and “GCEM” for Global Coordination of Exponential Maps

`adnRD`
: string for adapted point-varying neighborhoods in dimensionality reduction. "none" for no adaptivity, "adbas" for simple local PCA based neighborhoods (see code), "ad1" for the Zhan et al. method, and "ad2" for Wang et al. method. In short, the more linear data is around `x`, the more `x` has neighbors

`knnRD`
: fixed number of neighbors at each point for dimensionality reduction (used only if `adnRD=="none"`). If zero, a simple heuristic will determine it around `sqrt(nrow(data))`. Irrelevant if `mdim=="RML"`

`linbt`
: the type of (linear) orthonormal basis ; “PCA” for functional PCA, “wav” for wavelets basis, “four” for Fourier basis and “bsp” for B-spline basis

`filt`
: the desired filter in case of wavelets basis ; choice between EXTREMAL PHASE (daublet): “haar”, “dX” where X belongs to (4, 6, 8, 10, 12, 14, 16, 18, 20); LEAST ASYMMETRIC (symmlet): “sX” where X belongs to (4, 6, 8, 10, 12, 14, 16, 18, 20); BEST LOCALIZED: “lX” where X belongs to (2, 4, 6, 14, 18, 20); COIFLET: “cX” where X belongs to (6, 12, 18, 24, 30)

`wvar`
: boolean telling if we should select the sub-basis with most variable coefficients

`alpha`
: fraction of overlapping elements when building the traversal sequence of neighborhoods

`kmin`
: minimum number of neighbors at each point

`kmax`
: maximum number of neighbors at each point

`tsoft`
: tolerance factor for the visibility graph computation (between 0 and 1 ; should be close to 0)

`thlvl`
: fraction of total elements of data to be embedded using the initial local basis

`hdth`
: “hard” threshold, same as above parameter but integer. It defines the maximum level of elements in the Dijkstra graph which will be embedded using the initial local basis. If zero, only `thlvl` is considered

`advhr`
: if TRUE, the heuristic for RML’s last step is based on estimated graph distances from minimal connectivity graph ; if FALSE, the heuristic use euclidian distances

`mreg, mreg1, mreg2`
: regression method to use (1 for base model, 2 for residuals) ; choice between between “PPR” (Projection Pursuit Regression), “rforest” (random forests), “kNN, fkNN” (Nadaraya-Watson, after dimensionality reduction or not), “lPCA” (local PCA regression, without dimensionality reduction), “GP” (gaussian processes), “SVR” (Support Vector Regression)

`ppts`
: TRUE for pointwise regression

`stred`
: TRUE for standardized outputs

`trcv`
: fraction of total examples on which a model is trained during cross-validation procedures

`mclust`
: the clustering method, to be chosen between “HDC” (k-means based on Hitting Times), “CTH” (Commute-Time Hierarchic), “CTKM” (Commute-Time k-means), “spec” (spectral clustering), “CH” (hierarchical clustering), “PCA” (PCA-k-means from Chiou and Li ; see references), “KM” (basic k-means)

mclass	type of classifier to use in the prediction accuracy step ; choice between "kNN" (k-nearest-neighbors), "ctree" (classification tree), "RDA" (Regularized Discriminant Analysis), "rforest" (random forests), "SVM" (Support Vector Machines). Only the first two were intensively tested
redy	boolean telling if the outputs should be reduced (with PCA) as a preprocessing step
adnCC	same as adnRD above, for clustering
knnCC	fixed number of neighbors at each point in clustering ; used only if adnCL == FALSE
wcl	FALSE for disable clustering step ; can be useful for comparison purposes
iclusts	imposed clustering, like (1,1,2,2,2,1) (if known by user ; used in the fm_resids method)
symm	boolean at TRUE for symmetric similarity matrix (see code. It does not impact much the result
weight	boolean at TRUE for weighted hitting/commute times, like in the article of Liben-Nowell and Kleinberg
sigmo	boolean at TRUE for sigmoid commute-time kernel, like in the article of Yen et al.
minszcl	minimum size for a cluster. This is interesting to not allow too small clusters for the regression stage ; recommanded values are above 30-50
maxcl	maximum number of clusters ; Inf stands for "no limit", i.e. determined by stability-prediction loops only
taus	threshold for stability check ; value between 0 (every method accepted) and 1 (only ultra-stable method accepted). Recommanded between 0.6 and 0.9
Ns	number of stability runs before averaging results (the higher the better, although slower..)
tauc	threshold for prediction accuracy check (after subsampling) ; value between 0 (every clustering accepted) and 1 (only "well separated" clusters accepted). Recommanded between 0.6 and 0.9
Nc	number of partitions predictions runs before averaging results (same remark as for Ns above)
loo	TRUE for leave-one-out cross-validation
nfold	number of cross-validation loops to run
nhold	number of curves to hold in the training step for cross-validation
verb	TRUE for printing what is going on. A further release will allow to choose levels of verbosity
plotc	TRUE for plotting current Q2 curves at each step

Details

If coded argument is left unspecified (0), it will be estimated using Farahmand et al. algorithm.

The algorithm in fmetam works in three main steps :

1. Optional clustering of intputs-outputs.
2. Dimensionality reduction in each outputs cluster.
3. Statistical learning "inputs –> reduced coordinates".

The predict.modelcf function then computes the associated reconstruction "recuced coordinates –> curves".

Value

`fmetam_1cl` and `fmetam` return a list of relevant parameters for internal use.

`nfoldcv` returns a list with the following attributes:

- curves = predicted curves (only in leave-one-out mode);
- MSE = (average) mean squares error curve for the model chosen;
- stMSE = corresponding standard deviation;
- pvar = (average) mean squares error curve for the training mean model;
- stvar = corresponding standard deviation;
- Q2 = Q2 error curve (should be above 0 and close to 1);
- stQ2 = corresponding standard deviation;
- ssclust = measure of clusters' sizes homogeneity (>=0, should be as small as possible);
- snclust = histogram vector of the number of clusters found over the runs; e.g., `(0,0,32,78,0,...,0)` means 78 runs with 4 clusters and 32 runs with 3 clusters.

NOTE: standard deviations cannot be accurate if `nfold` parameter is too small. Value around 100 or above is recommended.

Examples

```
data(datacf)
#plot curves of the dataset
plotC(dataOut1)
plotC(dataOut2)

#build a standard model of the first dataset using 250 training samples
trainInds = sample(1:300, 250)
m = fmetam(dataIn[trainInds,],dataOut1[trainInds,],d=3,wcl=FALSE,mdim="linear")
# print the model
print(m)
#get the predicted curves
pred = predict.modelcf(m, dataIn[-trainInds,])
#get and plot error estimators
errs = fperrors(pred,dataOut1[-trainInds,],colMeans(dataOut1[trainInds,]))
plot(errs$MSE, type="l", ylab="MSE")
plot(1-errs$MSE/errs$pvar, type="l", ylim=c(0,1), ylab="Q2")

# run cross validation for the second dataset
## Not run: nf = nfoldcv(dataIn,dataOut2,d=3,wcl=FALSE,mdim="linear",plotc=FALSE)
nf = nfoldcv(dataIn[1:200,],dataOut2[1:200,],d=3,wcl=FALSE,mdim1="linear",
    nfold=10,plotc=FALSE) #for speed
# plot MSE +/- standard deviation
rg = range(nf$MSE-nf$stMSE,nf$MSE+nf$stMSE)
plot(nf$MSE-nf$stMSE,type="l",lwd=3,col=4,ylim=rg); par(new=TRUE)
plot(nf$MSE+nf$stMSE,type="l",lwd=3,col=4,ylim=rg); par(new=TRUE)
plot(nf$MSE,type="l",lwd=3,col=1,ylim=rg)
# plot Q2 +/- standard deviation
rg = c(-0.5, 1.5)
plot(nf$Q2-nf$stQ2,type="l",lwd=3,col=4,ylim=rg); par(new=TRUE)
plot(nf$Q2+nf$stQ2,type="l",lwd=3,col=4,ylim=rg); par(new=TRUE)
plot(nf$Q2,type="l",lwd=3,ylim=rg)
```

neighbors *Top-level neighborhoods functions*

Description

Internal use only ; should not be called by the final user.

`getNI` gets neighborhoods (designed for any algorithm), adaptively or not.

`neighbs_RML` gets neighborhoods designed for Riemannian Manifold Learning algorithm (see `RML`).

Usage

```
getNI(data, adn, d, k, mutual=FALSE, threshP = 0.95, eta = 0.05,
expand=FALSE)

neighbs_RML(data, rgdists, kmin, kmax, tsoft=0.1)
```

Arguments

`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`rgdists`	rough approximations of graph distances (only for `RML`)
`kmin`	minimum number of neighbors at each point
`kmax`	maximum number of neighbors at each point
`adn`	string for adapted point-varying neighborhoods. "none" for no adaptivity, "ad-bas" for simple local PCA based neighborhoods (see code), "ad1" for the Zhan et al. method, and "ad2" for Wang et al. method. In short, the more linear data is around `x`, the more `x` has neighbors
`d`	estimated data dimensionality ; useful only if `adn` is set
`k`	fixed number of neighbors at each point ; used only if `adn == FALSE`
`tsoft`	tolerance factor for the visibility graph computation (between 0 and 1 ; should be close to 0)
`mutual`	boolean for computing mutual graph neighborhoods
`threshP`	threshold percentage to estimate number of neighbors for a locally linear surface
`eta`	threshold percentage used for neighborhood contraction
`expand`	boolean for enable the final expansion step in the Wang et al. algorithm

Details

`neighbs_RML` computes the visibility graph as described in the article of Lin et al. See this paper for further explanations.

Value

A list of neighborhoods, describing a graph.

References

T. Lin, H. Zha and S. U. Lee, **Riemannian Manifold Learning for Nonlinear Dimensionality Reduction**, at European Conference on Computer Vision, Graz, Austria 9: 44-55, 2006

J. Wang, Z. Zhang and H. Zha, **Adaptive Manifold Learning**, in Advances in Neural Information Processing Systems 17: 1473-1480, 2005

Y. Zhan, J. Yin, X. Liu and G. Zhang, **Adaptive Neighborhood Select Based on Local Linearity for Nonlinear Dimensionality Reduction**, at International Symposium on Advances in Computation and Intelligence, Huangshi, China 5821: 337-348, 2009

`orthBasis` *Around orthonormal bases*

Description

`linEmb` performs decomposition onto an orthonormal basis amoung functional PCA, wavelets (any filter), Fourier and B-spline basis.

`linear_rec` performs linear reconstruction based on coefficients.

Usage

```
linEmb(data, d, linbt="PCA", filt="haar", wvar=TRUE)

linear_rec(basis, coefs)
```

Arguments

`data`	matrix of n functions (written as vectors) in rows ; `data[i,]` is the i-th D-dimensional function
`d`	desired number of coefficients ; corresponds to basis resolution, reduced `d`-dimensionality
`linbt`	the type of (linear) orthonormal basis ; "PCA" for functional PCA, "wav" for wavelets basis, "four" for Fourier basis and "bsp" for B-spline basis
`filt`	the desired filter in case of wavelets basis ; choice between EXTREMAL PHASE (daublet): "haar", "dX" where X belongs to (4, 6, 8, 10, 12, 14, 16, 18, 20); LEAST ASYMMETRIC (symmlet): "sX" where X belongs to (4, 6, 8, 10, 12, 14, 16, 18, 20); BEST LOCALIZED: "lX" where X belongs to (2, 4, 6, 14, 18, 20); COIFLET: "cX" where X belongs to (6, 12, 18, 24, 30)
`wvar`	boolean telling if we should select the sub-basis with most variable coefficients
`basis`	orthonormal functions (written as vectors) in rows
`coefs`	matrix of projected coefficients in rows

Value

`linEmb` returns a list L, with L$embed = matrix of d-dimensional embeddings in rows, L$basis = matrix of orthonormal functions (in rows).

`linear_rec` performs linear reconstruction based on coefficients.

References

Functional PCA: J. Ramsay and B. W. Silverman, **Functional Data Analysis**, Springer 2005

Wavelets basis R package used is *wmtsa* available here http://cran.r-project.org/web/packages/wmtsa/index.html

Examples

```
#generate a \dQuote{triginometric} functional dataset
cosFunc = cos( seq( from=0,to=2*pi,by=2*pi/200 ) )
sinFunc = sin( seq( from=0,to=2*pi,by=2*pi/200 ) )
coefs = matrix( runif(200),ncol=2 )
fdata = coefs %*% rbind(cosFunc, sinFunc)
#plot the two first Fourier functions
four = linEmb(fdata, 2, "four")
plotC(four$basis)
#output the three first PCA functions
fpca = linEmb(fdata, 3, "PCA")
plotC(fpca$basis)
```

---

predict *Predictions for some models*

Description

`predictClassif` estimates the label of an object x.

`predictRegress` estimates the output y for an input x.

These two last functions should not be used directly. Prefer calling specific methods from some R package.

`predict.modelcf` estimates the output curve y for an input vector x, using a model built by the `fmetam` function.

Usage

```
predictRegress(model,newIn_s)

predictClassif(model,newIns)

## S3 method for class 'modelcf'
predict(object, x, verb = FALSE, ...)
```

Arguments

`model`	a classification or regression model, as output by `learnClassif` or `learnRegress`
`newIn_s, newIns`	a matrix of (testing) input vectors in rows
`object`	a **modelcf** model, output of `fmetam`
`x`	a matrix of n input vectors in rows, which can be given as a R matrix or a text file. x[i,] is the i-th p-dimensional input.
`verb`	TRUE for printing what is going on. A further release will allow to choose levels of verbosity
`...`	unused (for compatibility with generic method `predict`)

Value

`predictClassif` (resp. `predictRegress`) returns a vector of integer (resp. real) values.

`predict.modelcf` return a matrix of curves in rows, one for each testing example.

Examples

```
#get the first artificial dataset and build a standard model of it
#using 250 training samples
data(datacf)
trainInds = sample(1:300, 250)
m = fmetam(dataIn[trainInds,],dataOut1[trainInds,],d=3,wcl=FALSE,mdim="linear")
#get the predicted curves
pred = predict.modelcf(m, dataIn[-trainInds,])
#plot the (L1) error between real and predicted curves
plotC(dataOut1[-trainInds,] - pred)
```

---

printPlot *Printing and plotting utilitiy functions*

Description

`plotC` plots a matrix of curves (in rows).

`plotPts` plots a set of 2D points given by the column numbers in a matrix.

`print.modelcf` prints some relevant parameters of a constructed model (as output by `fmetam`).

Usage

```
plotC(data, cl=rep(1,nrow(data)), rg=c(min(data),max(data)), ...)

plotPts(data, cols=c(1,2), cl=rep(1,nrow(data)), ...)

## S3 method for class 'modelcf'
print(x, ...)
```

Arguments

`data`	matrix of n vectors ("or functions") in rows ; `data[i,]` is the i-th m-dimensional vector
`cl`	an integer vector with R colors to be applied to each row
`rg`	the range on y axis in case of functions drawing
`cols`	the two selected columns in case of points plotting
`x`	a model as output by `fmetam`
`...`	any other relevant graphical parameter(s)

Examples

```
#plot first artificial dataset
data(datacf)
plotC(dataOut1)
#generate a mixture of three gaussian data set
data = rbind( matrix(rnorm(200,mean=2,sd=0.5),ncol=2),
    matrix(rnorm(200,mean=4,sd=0.5),ncol=2),
    matrix(rnorm(200,mean=6,sd=0.5),ncol=2) )
#cluster it using k-means
km = gtclusts("KM", data, 3)
#plot result
plotPts(data, cl=km)
```

---

redDim *Dimensionality reduction and associate reconstruction*

Description

`nlin_redDim` nlin_redDim is a generic method for dimensionality reduction.

`nlin_adaptRec` is a generic method for reconstruction.

For internal use only ; use specific methods directly if you need.

Usage

```
nlin_redDim(method, data, d, adn, k, alpha, trcv,
kmin, kmax, tsoft, thlvl, hdth, advhr)

nlin_adaptRec(method, embobj, newEmb)
```

Arguments

`method`	the dimensionality reduction method (to be) used, to be chosen between `RML` and `LPcaML`
`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`d`	estimated data dimension (e.g. through functions from `dimension` file)
`adn`	string for adapted point-varying neighborhoods. "none" for no adaptivity, "adbas" for simple local PCA based neighborhoods (see code), "ad1" for the Zhan et al. method, and "ad2" for Wang et al. method. In short, the more linear data is around `x`, the more `x` has neighbors
`k`	fixed number of neighbors at each point (used only if `adn=="none"`). If zero, a simple heuristic will determine it around `sqrt(nrow(data))`
`alpha`	fraction of overlapping elements when building the traversal sequence of neighborhoods
`trcv`	fraction of total examples on which a model is trained during cross-validation procedures
`kmin`	minimum number of neighbors at each point
`kmax`	maximum number of neighbors at each point

tsoft	tolerance factor for the visibility graph computation (between 0 and 1 ; should be close to 0)
thlvl	fraction of total elements of data to be embedded using the initial local basis
hdth	"hard" threshold, same as above parameter but integer. It defines the maximum level of elements in the Dijkstra graph which will be embedded using the initial local basis. If zero, only thlvl is considered
advhr	if TRUE, the heuristic for RML's last step is based on estimated graph distances from minimal connectivity graph ; if FALSE, the heuristic use euclidian distances
embobj	an object as output by RML or LPcaML functions
newEmb	a new embedding from which the high dimensional object has to be estimated

Value

A list with the embedding in $embed, and some technical parameters for reconstruction.

---

refining *Rearrangement of clusters*

Description

reordering changes the clusters numerotation to use all the integers from 1 to K.

fusion_smcl merges clusters until no one has size inferior than minszcl argument.

mergeToK merges clusters given through its arguments until there are exactly K classes.

Usage

```
reordering(clusts)

fusion_smcl(data, clusts, minszcl)

mergeToK(data, clusts, K)
```

Arguments

data	matrix of n vectors in rows ; data[i,] is the i-th m-dimensional vector
clusts	a partition of the data, as outputs by gtclusts ; e.g., (1,1,1,1,2,2,2,2,1,1,3,3,3)
K	expected number of clusters
minszcl	minimum size of a cluster

Value

An integer vector describing classes (same as kmeans()$cluster field).

Examples

```
#on an artificial dataset
data = matrix(runif(300),ncol=3)
clusts = gtclusts("KM",data,10)
print(clusts)
#fusion clusters of size >=20
print(reordering(fusion_smcl(data,clusts,20)))
#merge until 3 clusters
print(mergeToK(data,clusts,3))
```

---

regression *Statistical learning (regression)*

Description

`learnRegress` builds a regression object (see code for details).

`optimParams_regress` optimize parameters for the chosen method.

These two methods should not be called directly. Using the specific technique inside its own package is a better idea.

Usage

```
learnRegress(x, y, method, params, stred)

optimParams_regress(x, y, method, k, d, trcv, verb)
```

Arguments

`x`	matrix of `n` input vectors in rows. `x[i,]` is the i-th p-dimensional input
`y`	matrix of `n` outputs in rows. `y[i,]` is the i-th m-dimensional output
`method`	regression method, to be chosen between "PPR" (Projection Pursuit Regression), "rforest" (random forests), "kNN, fkNN" (Nadaraya-Watson, after dimensionality reduction or not), "lPCA" (local PCA regression, without dimensionality reduction), "GP" (gaussian processes), "SVR" (Support Vector Regression)
`params`	vector of parameters for the chosen method
`stred`	boolean at `TRUE` for standardize outputs `y`
`k`	fixed number of neighbors at each point to build the training set in cross-validation procedure
`d`	estimated outputs dimensionality ; relevant only if data has not been reduced
`trcv`	fraction of total examples on which a model is trained during cross-validation procedure.
`verb`	`TRUE` for printing what is going on.

Value

`learnRegress` returns a regression object (internal specifications).

`optimParams_regress` returns a vector of optimized parameters for the chosen method.

RML *Riemannian Manifold Learning*

Description

RML embeds data in the d-dimensional space using the Riemannian Manifold Learning method from the Lin et al. article.

RML_rec inverses the above procedure, reconstructing a curve (or any high dimensional vector) from its low-dimensional representation.

Usage

```
RML(data, d, kmin=0, kmax=0, tsoft=0.1,
thlvl=0.3, hdth=2, advhr=FALSE)

RML_rec(RLout, newEmb)
```

Arguments

data	matrix of n vectors in rows ; data[i,] is the i-th m-dimensional vector
d	estimated data dimension (e.g. through functions from dimension file)
kmin	minimum number of neighbors at each point
kmax	maximum number of neighbors at each point
tsoft	tolerance factor for the visibility graph computation (between 0 and 1 ; should be close to 0)
thlvl	fraction of total elements of data to be embedded using the initial local basis
hdth	"hard" threshold, same as above parameter but integer. It defines the maximum level of elements in the Dijkstra graph which will be embedded using the initial local basis. If zero, only thlvl is considered
advhr	if TRUE, the heuristic for RML's last step is based on estimated graph distances from minimal connectivity graph ; if FALSE, the heuristic use euclidian distances
RLout	an object as output by RML function
newEmb	a new embedding from which the high dimensional object has to be estimated

Details

The algorithm works in two main steps :

1. An origin vector y0 is determined, and its neighbors are embedded by projection onto a local tangent basis.
2. For further away elements y, we first find the predecessor yp of y on a shortest path from y0, and the yp neighbors written yi1,...,yik. The core idea then is to preserve (as much as possible) angles y-yp-yij to get the embedding z.

The reconstruction function RML_rec does exactly the same thing but from low-dimensional space to high-dimensional one. For better explanations, see the article.

Value

A list with the embedding in `$embed`, and some technical parameters for reconstruction.

References

T. Lin, H. Zha and S. U. Lee, **Riemannian Manifold Learning for Nonlinear Dimensionality Reduction**, at European Conference on Computer Vision, Graz, Austria 9: 44-55, 2006

T. Lin and H. Zha, **Riemannian Manifold Learning**, in IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (5): 796-809, 2008

Examples

```
#generate a swissroll dataset
n = 300 ; h = 3
phi = runif(n, min=0, max=2*pi)
z = runif(n, min=0, max=h)
#::set colors
rSize = 64
r = rainbow(rSize)
cols = r[pmin(floor((rSize/(2.0*pi))*phi)+1,rSize)]
#end set colors::
sw = cbind( phi*cos(phi), phi*sin(phi), z )

#launch algorithm and visualize result
emb = RML(sw, 2, kmin=15, kmax=30)$embed
plotPts(emb, cl=cols)
```

Index