Package 'modelcf'

September 29, 2010

Type Package

Version 2.0

Date 2010-09-29

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), class

Suggests rpart, e1071, randomForest, gbm, klaR, wmtsa, mlegp

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

License GPL (>= 3)

LazyLoad yes

R topics documented:

`A_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).

`CL_chameleon` *CHAMELEON clustering*

Description

`updateDissims` auxiliar function to update dissimilarities between one cluster and every other.

`chameleon` main function to cluster data according to CHAMELEON method, described in the article of Karypis et al. given in reference.

These two functions should not be called directly. Use `gtclusts` instead.

Usage

```
updateDissims(data, dissims, clusts, flags, index, alpha)

chameleon(data, dissims, K, alpha)
```

Arguments

`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`dissims`	matrix of dissimilarities (can be simple L2 distances, or more complicated like commute-times)
`K`	expected number of clusters
`alpha`	parameter controlling the relative importance of clusters' connectivity ; usual values range from 0.5 to 2
`clusts`	the vector of current clusters (something like `(1 1 1 2 2 3 1 3 3 4 4)` ... describing classes)
`flags`	boolean vector ; `flags[i] == TRUE` if `i` is the representative element for its cluster
`index`	index from which dissimilarities must be computed

Value

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

References

George Karypis, Eui-Hong Han and Vipin Kumar, CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling, in IEEE Computer 32(8): 68-75, 1999

`CL_clustering` *Main clustering function*

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.

Usage

```
phclust(dissims, K)

gtclusts(method, data, K, d=min(10, ncol(data)), adn=FALSE, knn=0,
symm=TRUE, weight=FALSE, sigmo=FALSE, alpha=1.0)
```

Arguments

`method`	the clustering method, to be chosen between "HDC" (k-means based on Hitting Times), "CTH" (Commute-Time Hierarchic), "CTHC" (Commute-Time CHAMELEON), "CTKM" (Commute-Time k-means), "specH" ("spectral-hierarchical" clustering), "specKM" (spectral clustering with k-means), "CH" (hierarchical clustering), "CHC" (CHAMELEON 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

`dissims`	matrix of dissimilarities (can be simple L2 distances, or more complicated like commute-times)
`K`	expected number of clusters
`d`	estimated data dimension (needed only for "ACP" method and when `adn` is `TRUE`). It can be estimated using functions `dimest1` or `dimest2`
`adn`	boolean for adapted point-varying neighborhoods, from Wang et al. article ; 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 == FALSE`. 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.
`alpha`	parameter controlling the relative importance of clusters' connectivity in CHAMELEON clustering ; usual values range from 0.5 to 2

Details

"Safe" methods are HDC, CTH, CTKM, CH, PCA and KM. Others could output weird results.

The spectral clustering is taken from the article of Ng et al., and adapted to work on a possibly disconnected graph

`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.

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

G. Karypis, E.-H. Han and V. Kumar, **CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling**, in IEEE Computer 32(8): 68-75, 1999

A. Y. Ng, M. Jordan and Y. Weiss, **On Spectral Clustering: Analysis and an algorithm**, at Advances in Neural Information Processing Systems, Vancouver, BC, Canada 14: 849-856, 2002

D. Liben-Nowell and J. Kleinberg ; **The link-prediction problem for social networks**, in Journal of the American Society for Information Science and Technology 58(7): 1019-1031, 2007

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

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, knn=20, symm=FALSE)
#plot results
plotPts(data, cl=km)
plotPts(data, cl=ct)
```

CL_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))
```

CL_findK_Clust *Clustering input-output data*

Description

`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

```
findK_gtclusts(x, y, method, d=min(10, ncol(data)), adn=FALSE, knn=0,
symm=TRUE, weight=FALSE, sigmo=FALSE, alpha=1.0, minszcl=30,
maxcl=Inf, mclass="kNN", taus=0.95, Ns=10, tauc=0.95, Nc=10,
trcv=0.7, nstagn=10)

gtclusts_inout(x, y, method, d=min(10, ncol(data)), redy=TRUE, adn=FALSE,
knn=0, symm=TRUE, weight=FALSE, sigmo=FALSE, alpha=1.0, minszcl=30,
maxcl=Inf, mclass="kNN", taus=0.95, Ns=10, tauc=0.95, Nc=10,
trcv=0.7, verb=TRUE, nstagn=10)
```

Arguments

`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
`method`	clustering method, to be chosen between "HDC" (k-means based on Hitting Times), "CTH" (Commute-Time Hierarchic), "CTHC" (Commute-Time CHAMELEON), "CTKM" (Commute-Time k-means), "specH" ("spectral-hierarchical" clustering), "specKM" (spectral clustering with k-means), "CH" (hierarchical clustering), "CHC" (CHAMELEON clustering), "PCA" (ACP-k-means from Chiou and Li ; see references), "KM" (basic k-means)
`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"`
`redy`	boolean telling if the outputs should be reduced (with PCA) as a preprocessing step
`adn`	boolean for adapted point-varying neighborhoods, from Wang et al. article ; 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 == FALSE
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.
alpha	parameter controlling the relative importance of clusters' connectivity in CHAMELEON clustering ; usual values range from 0.5 to 2
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

The algorithm 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). The number of clusters is equal to its maximum value.

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

G. Karypis, E.-H. Han and V. Kumar, **CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling**, in IEEE Computer 32(8): 68-75, 1999

A. Y. Ng, M. Jordan and Y. Weiss, **On Spectral Clustering: Analysis and an algorithm**, at Advances in Neural Information Processing Systems, Vancouver, BC, Canada 14: 849-856, 2002

D. Liben-Nowell and J. Kleinberg ; **The link-prediction problem for social networks**, in Journal of the American Society for Information Science and Technology 58(7): 1019-1031, 2007

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

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

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
inData = 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) )
#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 only outputs using hierarchical clustering
ch = findK_gtclusts(inData, outData, "CH", knn=20, minszcl=50, mclass="kNN",
    taus=0.8, Ns=10, tauc=0.8, Nc=10)
#plot result
plotC(outData, cl=ch)
#partition inputs-outputs using Commute-Time Hierarchic clustering
ct = gtclusts_inout(inData, outData, "CTH", knn=20, minszcl=50, mclass="kNN",
    taus=0.8, Ns=10, tauc=0.8, Nc=10)
#plot results, inputs then outputs
plotPts(inData, cl=ct)
plotC(outData, cl=ct)
```

CL_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. It can be estimated using functions dimest1 or dimest2
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)
```

CL_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))
```

CL_spectral *Spectral clustering*

Description

`spec_emb` embeds data into `R^d`, using symmetric laplacian (see Ng et al. article).

`spec_clust` clusters data into `K` classes using spectral embedding.

These two functions should not be called directly. Use `gtclusts` instead.

Usage

```
spec_emb(data, K, d, adn, knn)

spec_clust(method, data, K, d, adn, knn)
```

Arguments

`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`method`	the clustering (sub-)method, to be chosen between "specH" ("spectral-hierarchical" clustering) and "specKM" (spectral clustering with k-means)
`K`	expected number of clusters
`d`	estimated data dimension, e.g. as output by functions `dimest1` or `dimest2`
`adn`	boolean for adapted point-varying neighborhoods, from Wang et al. article ; 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 == FALSE`

Details

The `spec_clust` procedures works also for disconnected graphs by merging connected components (if more than `K`), or subdivising largest clusters when `K` is bigger than the number of connected components.

Value

`spec_emb` returns a matrix embedding the data in rows.

`spec_clust` returns an integer vector describing classes (same as `kmeans()$cluster` field).

References

A. Y. Ng, M. Jordan and Y. Weiss, **On Spectral Clustering: Analysis and an algorithm**, at Advances in Neural Information Processing Systems, Vancouver, BC, Canada 14: 849-856, 2002

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

CR_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, knn, 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
`knn`	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.

CR_knnlpca *Prediction for no-dimensionality-reduction-methods*

Description

`knnPredict and lpcaPredict` predict the response(s) for the given input(s), using `fkNN` and `lPCA` models.

These functions should not be called directly. Use `predictRegress` instead.

Usage

```
knnPredict(x_train, y_train, x, knn)

lpcaPredict(x_train, y_train, x, knn, stzouts=TRUE)
```

Arguments

`x_train`	matrix of `n` training input vectors in rows. `x_train[i,]` is the i-th p-dimensional training input
`y_train`	matrix of `n` training outputs in rows. `y_train[i,]` is the i-th m-dimensional output
`x`	matrix (or vector) of `q` testing input vectors in rows. `x[i,]` is the i-th p-dimensional testing input
`knn`	fixed number of neighbors at each point
`stzouts`	boolean at `TRUE` for standardize outputs `y_train` (after internal dimensionality reduction)

Value

A matrix of predictions, in same format as `y_train`.

CR_planExp *Extract training set*

Description

`xtr_plan1` and `xtr_plan2` extract a training set from n samples.

Usage

```
xtr_plan1(data, knn, trcv)

xtr_plan2(data, knn, trcv)
```

Arguments

data	matrix of n vectors in rows. data[i,] is the i-th m-dimensional vector
knn	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.

Details

xtr_plan1 searches for the maximum local variance points. xtr_plan2 searches for the maximum local density points.

Value

An integer vector giving the indices of the selected plan.

Examples

```
#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) )
#extract a plan of type 1
x1 = xtr_plan1(data, 20, 0.7)
cols = rep(1,300) ; cols[x1] = 2
plotPts(data,cl=cols)
#extract a plan of type 2
x2 = xtr_plan2(data, 20, 0.7)
cols = rep(1,300) ; cols[x2] = 2
plotPts(data,cl=cols)
```

CR_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, knn, 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), "BRT" (boosting of regression trees), "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
knn	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.
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.

M_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$MSE, type="l", ylim=c(0,1), ylab="MSE")
plot(1-errs$MSE/errs$pvar, type="l", ylim=c(0,1), ylab="Q2")
```

M_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)
```

`RD_dimension` *Dimension estimation*

Description

`locdim1` estimates the local dimension at some point (a row of the distance matrix).

`dimest1` estimates the intrinsic dimension of `data`, following the algorithm of Farahmand et al.

`dimest2` uses the `RML` graph to estimate dimension through its simplices as indicated by Lin et al. (this is really slow).

Usage

```
locdim1(x, knn)

dimest1(data, knn)

dimest2(data, knnmin, knnmax, tsoft=0.1)
```

Arguments

`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`x`	a row of the distance matrix computed from `data`
`knn`	fixed number of neighbors at each point
`knnmin`	minimum number of neighbors at each point
`knnmax`	maximum number of neighbors at each point
`tsoft`	tolerance factor for the visibility graph computation (between 0 and 1 ; should be close to 0)

Value

An integer equals to the estimated dimension.

References

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

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

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(dimest1(sw, 20))
## Not run: print(dimest2(sw, 15, 25)) #WARNING: very very slow!
```

RD_GCEM *Global Coordination of Exponential Maps*

Description

GCEM embeds data in the d-dimensional space using a mixing of the Local PCA Manifold Learning method from the Zhan et al. article, and the Riemannian Manifold Learning method from the Lin et al. article.

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

Usage

```
GCEM(data, d, adn=FALSE, knn=0, alpha=0.5, tsoft=0.1,
thlvl=0.3, hdth=0)

GCEM_rec(GCout, newEmb)
```

Arguments

data	matrix of n vectors in rows ; data[i,] is the i-th m-dimensional vector
d	estimated data dimension. It can be estimated using functions dimest1 or dimest2
adn	boolean for adapted point-varying neighborhoods, from Wang et al. article ; 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==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
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
GCout	an object as output by GCEM function
newEmb	a new embedding from which the high dimensional object has to be estimated

Details

The algorithm works exactly as `LPcaML` algorithm, but the local PCA coordinates are replaced by "local" RML coordinates. It is very experimental, and currently does not work as well as expected.

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

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, G. Zhang and En 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 = GCEM(sw, 2, alpha=0.7)$embed
plotPts(emb, cl=cols)
```

RD_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=FALSE, knn=0, alpha=0.5)

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. It can be estimated using functions dimest1 or dimest2
adn	boolean for adapted point-varying neighborhoods, from Wang et al. article ; 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==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
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

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, G. Zhang and En 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)
```

RD_orthBasis *Around orthonormal bases*

Description

basisMostVar selects the sub-basis of most variable coefficients.

genFourier generates the Fourier basis on an interval.

getMedElem returns the functional median based on L2 norm.

simpleWavBasis returns the wavelet basis expansion corresponding "best" to some dataset.

All these methods should not be used directly. Use the following one instead.

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

```
basisMostVar(basis, data, nbCoefs)

genFourier(data, nbCoefs, withVar=TRUE)

getMedElem(data)

simpleWavBasis(data, lvl, filt)

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

Arguments

basis	orthonormal functions (written as vectors) in rows
data	matrix of n functions (written as vectors) in rows ; data[i,] is the i-th D-dimensional function
nbCoefs,dim	desired number of coefficients ; corresponds to basis resolution, reduced d-dimensionality
withVar,wvar	boolean telling if we should select the sub-basis with most variable coefficients
lvl	the desired level (depth) in case of wavelets 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)
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

Value

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

`basisMostVar`, `genFourier` and `simpleWavBasis` return a matrix of orthonormal functions in rows.

`getMedElem` returns the functional median as a vector (like a row of any output matrix just above).

`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)
```

RD_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, knn, alpha, knnmin, knnmax,
tsoft, thlvl, hdth)

nlin_adaptRec(method, embobj, newEmb)
```

Arguments

`method`	the dimensionality reduction method (to be) used
`data`	matrix of n vectors in rows ; `data[i,]` is the i-th m-dimensional vector
`d`	estimated data dimension. It can be estimated using functions `dimest1` or `dimest2`
`adn`	boolean for adapted point-varying neighborhoods, from Wang et al. article ; 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==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
`knnmin`	minimum number of neighbors at each point
`knnmax`	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
`embobj`	an object as outputs by `RML`, `LPcaML` or `GCEM` 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.

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, G. Zhang and En 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

`RD_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, adn=FALSE, knnmin=0, knnmax=0, tsoft=0.1,
thlvl=0.3, hdth=0)

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. It can be estimated using functions dimest1 or dimest2
adn	boolean for adapted point-varying neighborhoods, from Wang et al. article ; in short, the more linear data is around x, the more x has neighbors
knnmin	minimum number of neighbors at each point
knnmax	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
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

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

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, knnmin=15, knnmax=30)$embed
plotPts(emb, cl=cols)
```

Z_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", knnmin=15, knnmax=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)
```

`Z_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.

See Also

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

Z_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.

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,
knnmin, knnmax, tsoft, thlvl, hdth, mreg, ppts, stred, trcv, verb)

fmetam(x, y, d=0, mclust="CTH", mclass="kNN", redy=TRUE, adnCC=TRUE,
knnCC=0, wcl=TRUE, symm=FALSE, weight=TRUE, sigmo=FALSE, alphaCL=1.0,
minszcl=30, maxcl=Inf, taus=0.95, Ns=10, tauc=0.95, Nc=10,
mdim="linear", adnRD=FALSE, knnRD=0, linbt="PCA", filt="haar",
wvar=TRUE, alphaRD=0.5, knnmin=0, knnmax=0, tsoft=0.1, thlvl=0.3,
hdth=0, mreg="PPR", ppts=FALSE, stred=TRUE, trcv = 0.7, verb = TRUE)

nfoldcv(x, y, d=0, mclust="CTH", mclass="kNN", redy=TRUE, adnCC=TRUE,
knnCC=0, wcl=TRUE, symm=FALSE, weight=TRUE, sigmo=FALSE, alphaCL=1.0,
minszcl=30, maxcl=Inf, taus=0.95, Ns=10, tauc=0.95, Nc=10,
mdim="linear", adnRD=FALSE, knnRD=0, linbt="PCA", filt="haar",
wvar=TRUE, alphaRD=0.5, knnmin=0, knnmax=0, tsoft=0.1, thlvl=0.3,
hdth=0, mreg="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) ; useful only if one of the following parameters is set: redy,adn,method=="ACP"
mdim	the dimensionality reduction method (to be) used : 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	boolean for adapted point-varying neighborhoods in dimensionality reduction, from Wang et al. article ; 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==FALSE). If zero, a simple heuristic will determine it around sqrt(nrow(data))

`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,alphaRD` fraction of overlapping elements when building the traversal sequence of neighborhoods

`knnmin` minimum number of neighbors at each point

`knnmax` 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

`mreg` regression method to use ; choice between between "PPR" (Projection Pursuit Regression), "rforest" (random forests), "BRT" (boosting of regression trees), "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` clustering method, to be chosen between "HDC" (k-means based on Hitting Times), "CTH" (Commute-Time Hierarchic), "CTHC" (Commute-Time CHAMELEON), "CTKM" (Commute-Time k-means), "specH" ("spectral-hierarchical" clustering), "specKM" (spectral clustering with k-means), "CH" (hierarchical clustering), "CHC" (CHAMELEON clustering), "PCA" (ACP-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` boolean for adapted point-varying neighborhoods in clustering, from Wang et al. article ; in short, the more linear data is around `x`, the more `x` has neighbors

`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

`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.
`alphaCL`	parameter controlling the relative importance of clusters' connectivity in CHAMELEON clustering ; usual values range from 0.5 to 2
`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.

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

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

G. Karypis, E.-H. Han and V. Kumar, **CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling**, in IEEE Computer 32(8): 68-75, 1999

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

D. Liben-Nowell and J. Kleinberg ; **The link-prediction problem for social networks**, in Journal of the American Society for Information Science and Technology 58(7): 1019-1031, 2007

A. Y. Ng, M. Jordan and Y. Weiss, **On Spectral Clustering: Analysis and an algorithm**, at Advances in Neural Information Processing Systems, Vancouver, BC, Canada 14: 849-856, 2002

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

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

Y. Zhan, J. Yin, G. Zhang and En 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

```
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$MSE, type="l", ylim=c(0,1), ylab="MSE")
plot(1-errs$MSE/errs$pvar, type="l", ylim=c(0,1), ylab="Q2")

# run cross validation for the second dataset
nf = nfoldcv(dataIn,dataOut2,d=3,wcl=FALSE,mdim="linear",plotc=FALSE)
# 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)
```

`Z_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)
```

Index