

VBmix package user manual

Copyright (C) 2011 Pierrick Bruneau, <mailto:pbruneau@gmail.com>
see README for full notice

February 2011

Contents

1	installation	4
2	usage: function index	4
2.1	newGmm	4
2.2	appendToGmm	5
2.3	newMppca	5
2.4	appendToMppca	6
2.5	subMppca	7
2.6	mppcaToGmm	7
2.7	subGmm	8
2.8	randomGmm	8
2.9	normMppca	8
2.10	eigenMppca	9
2.11	gmmToMppca	9
2.12	gramschmidt	10
2.13	norm	10
2.14	gaussianKL	11
2.15	getQforComp	11
2.16	isNonVoid	12
2.17	appendToList	12
2.18	normalizeVariable	13
2.19	binnedEntropy	13
2.20	multinomial	13
2.21	mvngen	14
2.22	gmmgen	14
2.23	mvndensity	15
2.24	gmmdensity	15
2.25	klmc	16
2.26	jsmc	16
2.27	klut	17
2.28	jsut	17
2.29	covgen	18

2.30	rDirichlet	18
2.31	dDirichlet	19
2.32	setDomain	19
2.33	generateSparsePoints	20
2.34	buildFrame	20
2.35	knn	21
2.36	mergeClassif	22
2.37	constrClassif	22
2.38	sampleClassif	23
2.39	classicEM	24
2.40	varbayes	24
2.41	vbcomp	25
2.42	vbconstr	26
2.43	mppca	26
2.44	mmppca	27
2.45	gmmkmsock	28
2.46	pca	29
2.47	getCouple	29
2.48	getLabels	30
2.49	getResp	30
2.50	extractSimpleModel	31
2.51	subVarbayes	31
2.52	incredMerge	32
2.53	ZtoLabels	32
2.54	getDataLikelihood	33
2.55	getBic	33
2.56	getVarbayesResp	33
2.57	getTimestamp	34
2.58	getElapsed	34
2.59	sort_index	35
2.60	getPointer	35
2.61	getSEXP	35
2.62	datagen	36
2.63	circlegen	37
2.64	spiralgen	37
2.65	generate2Dtransform	38
2.66	dat1sample	38
2.67	dat2sample	39
2.68	dat3sample	39
2.69	plotGmm	40
2.70	displayScatter	40
2.71	displayGraph	41
2.72	displaySVM	42
2.73	displayNnet	43
2.74	Rdct	44
2.75	Rdct2D	44

2.76 RinvDct2D	44
2.77 RGBtoLab	45
2.78 readPixmapFile	45
2.79 readLabelFile	46
2.80 reBuild	46
2.81 pixmapToVector	47
3 data sets index	47
4 PL scripts	48
5 socket client and server	49

1 installation

Full instructions for building the package may be found in the `INSTALL` file.

The package is loaded in R using:

```
source("[path_to_VBmix]/VBmix/VBmix.R", chdir=TRUE)
```

`chdir=TRUE` is necessary for independence with regard to the chosen installation directory. Note that the path to VBmix is stored in `VBmixwd` for further use in R.

VBmix depends on the following R packages (available at <http://cran.r-project.org/web/packages/>):

- `e1071` (SVM + misc)
- `lattice` (standard graphics package)
- `grid` (standard graphics package)
- `nnet` (neural networks)
- `pixmap` (image manipulation)
- `mnormt` (multivariate normal)
- `MASS` (statistical tools and data sets)

2 usage: function index

In this section we propose an entry per function available in the package. Note that C/C++ routines of this package should be accessed only through interface functions (`interfaces.R`).

Type-checking is not always performed in C routines: incorrect parametrization may cause R to crash.

2.1 `newGmm`

```
newGmm()
```

creates an empty GMM data structure.

parameters: none

returns: list object with the following members:

- `$w`: numeric vector containing the component weights of the mixture model.
- `$mean`: list with respective means (numeric vectors) as elements.

- `$cov`: list with respective covariance matrices as elements.
- `$a`: constraints between components, encoded in a `numeric` vector. One value per component. 2 components associated to the same value are said to be from the same origin. Used in `vbconstr`.

localisation: misc/Rutils.R

example:

```
temp <- newGmm()
```

see:

2.2 appendToGmm

```
appendToGmm(mod1, mod2)
```

concatenates `mod2` to `mod1`.

parameters:

- `mod1`: GMM to which `mod2` is appended.
- `mod2`: GMM appended to `mod1`.

returns: GMM with concatenated models, and `$a` set accordingly.

localisation: misc/Rutils.R

example:

```
temp <- appendToGmm(gmmpen[[1]], gmmpen[[2]])
```

see:

`newGmm`

2.3 newMppca

```
newMppca()
```

creates an empty posterior MPPCA data structure.

parameters: none.

returns: list object with the following members:

- `$alpha`: numeric vector for α parameter.
- `$numoment`: list of numeric vectors, containing $\mathbb{E}[\nu_{k,j}]$ parameters.

- `$nub`: list of numeric vectors, containing b_{kj} parameters for ν .
- `$taumoment`: numeric vector for τ parameter. NB: all set identically and statically to 1, as in [Bruneau et al., 2010b,c] a single static τ parameter is used.
- `$taua`: numeric vector for a_k parameters for τ .
- `$taub`: numeric vector for b_k parameters for τ .
- `$wmean`: list of matrices containing $\mathbb{E}[\Lambda_k]$ parameters.
- `$wsigma`: list of matrices containing $Cov(\Lambda_k^i)$.
- `$xsigma`: list of matrices containing $Cov(\mathbf{x}_k)$.
- `$mumean`: list of numeric vectors, containing means of the MPPCA model.
- `$musigma`: list of matrices with covariances for the mean estimates.
- `$mustar`: list of numeric vectors, containing prior means of the MPPCA model, used for initialisation.

localisation: misc/Rutils.R

example:

```
temp <- newMppca()
```

see:

[Bruneau et al., 2010b,c]

2.4 appendToMppca

```
appendToMppca(mod1, mod2)
```

appends mod2 to mod1.

parameters:

- `mod1`: MPPCA model to be appended to.
- `mod2`: MPPCA to append to mod1.

returns: appended models.

localisation: misc/Rutils.R

example:

```
temp <- appendToMppca(pcapien[[1]], pcapien[[2]])
```

see:

`newMppca`

2.5 subMppca

```
subMppca(model, prune=FALSE, thres=2.001, quick=FALSE, noxmean=TRUE)
```

removes unused components and factor columns from `model`.

parameters:

- `model`: MPPCA model to be shrunked.
- `prune`: if `TRUE`, void factor columns are removed.
- `thres`: threshold for component selection. A components is selected iif $\alpha > \text{thres}$.
- `quick`: influences method for void factor columns detection. If `FALSE`, a KL-based criterion is employed (more accurate). If `TRUE`, column norms are used (useful for very high dimensional data sets).
- `noxmean`: should always be set to `TRUE`.

returns: shrunked MPPCA model.

localisation: misc/Rutils.R

example:

```
temp <- mppca(pendat, 15, qmax=8, maxit=50)
temp2 <- subMppca(temp, prune=T, quick=TRUE)
```

see:

2.6 mppcaToGmm

```
mppcaToGmm(model, notau=FALSE)
```

converts a MPPCA model to a GMM.

parameters:

- `model`: MPPCA model to be converted.
- `notau`: if `TRUE`, covariances are built with $\Lambda\Lambda^T$ without adding τ .

returns: GMM after conversion.

localisation: misc/Rutils.R

example:

```
temp <- mppcaToGmm(pcapen[[1]])
```

see: [Tipping and Bishop, 1999]

2.7 subGmm

```
subGmm(model, dims=c(1,2), inds=NULL)
```

select a subset of components and dimensions from an input GMM.

parameters:

- **model:** GMM from which to extract subsets.
- **dims:** numeric vector of the extracted dimensions.
- **inds:** numeric vector of selected components indices. If `NULL`, all components are selected.

returns: subset of input GMM.

localisation: misc/Rutils.R

example:

```
temp <- subGmm(gmmpen[[1]], inds=1:3)
```

see:

2.8 randomGmm

```
randomGmm(domain=10)
```

sample randomly a GMM. Number of components is sampled from a Poisson law, means uniformly from `[-domain, domain]`, and covariance matrices using `covgen` function.

parameters:

- **domain:** determines the domain from which means are sampled.

returns: randomly sampled GMM.

localisation: misc/Rutils.R

example:

```
temp <- randomGmm()
```

see:

2.9 normMppca

```
normMppca(mppca1)
```

adjusts a MPPCA model to ensure that all factor matrices have same rank (`q`).

parameters:

- `mppca1`: MPPCA model to be adjusted.

returns: adjusted MPPCA.

localisation: `misc/Rutils.R`

example:

```
temp <- newMppca()
for(i in 1:5) temp <- appendToMppca(temp, pcapen[[i]])
temp <- normMppca(temp)
```

see:

2.10 `eigenMppca`

```
eigenMppca(mod)
```

uses eigen decompositions to align factor matrices to principal bases (see [Tipping and Bishop, 1999]). NB: `mppca` and `mmppca` already perform this operation during their post-processing.

parameters:

- `mod`: MPPCA model which components have to be aligned.

returns: adjusted MPPCA.

localisation: `misc/Rutils.R`

example:

```
temp <- eigenMppca(pcapen[[2]])
```

see:

2.11 `gmmToMppca`

```
gmmToMppca(model, alpha=500)
```

uses eigen decompositions to convert a GMM to a MPPCA model.

parameters:

- `model`: GMM to be converted.
- `alpha`: GMM are associated to weights, and MPPCA models to population sizes. `alpha` is the chosen population size for the output MPPCA.

returns: converted MPPCA model.

localisation: misc/Rutils.R

example:

```
temp <- gmmToMppca(gmmpen[[3]])
```

see:

2.12 gramschmidt

```
gramschmidt(mat)
```

performs Gram-Schmidt orthogonalization on `mat`.

parameters:

- `mat`: matrix object to orthogonalize.

returns: orthogonalized matrix.

localisation: misc/Rutils.R

example:

```
temp <- gramschmidt(pcapen[[3]]$wmean[[1]])
```

see:

2.13 norm

```
norm(vec)
```

computes Euclidian norm of `vec`.

parameters:

- `vec`: numeric vector.

returns: norm value.

localisation: misc/Rutils.R

example:

```
temp <- norm(gmmpen[[2]]$mean[[1]])
```

see:

2.14 gaussianKL

```
gaussianKL(N0, N1)
```

computes $\text{KL}(\mathcal{N}(\mathbf{0}, \Sigma_0) || \mathcal{N}(\mathbf{0}, \Sigma_1))$.

parameters:

- N0: Σ_0 .
- N1: Σ_1 .

returns: KL value.

localisation: misc/Rutils.R

example:

```
temp <- gaussianKL(gmpen[[1]]$cov[[1]], gmpen[[1]]$cov[[2]])
```

see:

2.15 getQforComp

```
getQforComp(loadings, tau=1.0, verbose=FALSE, quick=FALSE)
```

gets the rank associated with a properly aligned factor matrix.

parameters:

- **loadings:** aligned factor matrix.
- **tau:** diagonal noise used for KL computations.
- **verbose:** if TRUE maximal info is displayed.
- **quick:** if TRUE, column norm values are used instead of KL computations (less accurate but faster).

returns: rank associated with loadings.

localisation: misc/Rutils.R

example:

```
temp <- getQforComp(handmods[[1]]$wmean[[2]], quick=TRUE)
```

see:

2.16 isNonVoid

```
isNonVoid(loadings)
```

checks if `loadings` contains only void columns.

parameters:

- `loadings`: matrix from which we check the columns.

returns: TRUE if at least 1 column is not void.

localisation: misc/Rutils.R

example:

```
isNonVoid(handmods[[1]]$wmean[[2]])  
[1] TRUE
```

see:

2.17 appendToList

```
appendToList(lst, obj, appendList=FALSE)
```

appends 1 list object to another.

parameters:

- `lst`: list object to which we append an object.
- `obj`: object to append.
- `appendList`: if TRUE, `obj` should be a list object, which elements are appended. if FALSE, `obj` is simply added to `lst`.

returns: list object with `obj` appended to `lst`.

localisation: misc/Rutils.R

example:

```
temp <- list()  
temp <- appendToList(temp, handmods[[1]]$wmean, appendList=TRUE)  
temp <- appendToList(temp, handmods[[2]]$wmean, appendList=TRUE)
```

see:

2.18 normalizeVariable

```
normalizeVariable(v)
```

normalizes a variable (numeric vector) in [0,1].

parameters:

- v: a numeric vector.

returns: normalized numeric vector.

localisation: misc/Rutils.R

example:

```
temp <- normalizeVariable(irisdata[,1])
```

see:

2.19 binnedEntropy

```
binnedEntropy(v, nbins=100)
```

uses bins to approximate the empirical entropy of a variable.

parameters:

- v: a numeric vector.
- nbins: number of bins used to estimate the entropy.

returns: entropy value.

localisation: misc/Rutils.R

example:

```
temp <- binnedEntropy(irisdata[,1])
```

see:

2.20 multinomial

```
multinomial(weights, k)
```

samples from a k-multinomial.

parameters:

- weights: numeric vector with the weights of the multinomial. Sum to 1.

- `k`: size of the weight vector.

returns: an integer value in $[1, k]$, coded as a 1-of- k variable [Bishop, 2006, chap. 9]

localisation: misc/utils.c, interfaces.R

example:

```
weights <- c(0.3, 0.5, 0.2)
multinomial(weights, 3)
[1] 0 1 0
```

see:

[Bishop, 2006, chap. 9]

2.21 mvngen

```
mvngen(mean, cov, nitem)
```

sample `nitem` elements from $\mathcal{N}(\text{mean}, \text{cov})$.

parameters:

- `mean`: numeric vector.
- `cov`: covariance matrix.
- `nitem`: number of items to generate.

returns: `nitem` x `d` matrix with elements as rows (further denoted as a matrix of row-elements)

localisation: misc/utils.c, interfaces.R

example:

```
mvngen(c(0, 0), diag(2), 5)
      [,1]      [,2]
[1,] -0.09898211  1.4516438
[2,]  0.20814926 -0.1233861
[3,]  0.18410071  0.5995621
[4,]  0.65994562  0.8328315
[5,]  2.33098055 -0.5943117
```

see:

2.22 gmmgen

```
gmmgen(mod, nitem)
```

sample elements from a GMM.

parameters:

- **mod:** GMM sampled from.
- **nitem:** number of elements to be sampled.

returns: `nitem` x `d` matrix with elements as rows.

localisation: `misc/utills.c`, `interfaces.R`

example:

```
temp <- gmmgen(gmmpen[[1]], 50)
```

see:

2.23 mvndensity

```
mvndensity(mean, cov, data)
```

get densities of a set of elements w.r.t a multivariate normal.

parameters:

- **mean:** numeric vector, mean of the multivariate normal.
- **cov:** covariance matrix of the multivariate normal.
- **data:** matrix of row-elements.

returns: numeric vector containing densities.

localisation: `misc/utills.c`, `interfaces.R`

example:

```
temp <- mvngen(c(0, 0), diag(2), 5)
mvndensity(c(0,0), diag(2), temp)
[1] 0.137188286 0.032318242 0.005181099 0.047312602 0.033178600
```

see:

2.24 gmmdensity

```
gmmdensity(mod, data)
```

get densities of a set of elements w.r.t a GMM.

parameters:

- `mod`: reference GMM.
- `data`: matrix of row-elements.

returns: numeric vector containing densities.

localisation: `misc/Utils.c`, `interfaces.R`

example:

```
temp <- gmmgen(gmmpen[[1]], 50)
temp2 <- gmmdensity(gmmpen[[1]], temp)
```

see:

2.25 `klmc`

```
klmc(mod1, mod2, nsamp=5000)
```

computes Monte Carlo estimate of KL divergence between GMM.

parameters:

- `mod1`: GMM parameter to $KL(mod1||mod2)$.
- `mod2`: GMM parameter to $KL(mod1||mod2)$.
- `nsamp`: number of samples used to build estimate.

returns: KL value.

localisation: `misc/Utils.c`, `interfaces.R`

example:

```
temp <- klmc(gmmpen[[1]], gmmpen[[2]])
```

see:

2.26 `jsmc`

```
jsmc(mod1, mod2, nsamp=5000)
```

computes Monte Carlo estimate of Jensen-Shannon (JS) divergence between GMM.

parameters:

- `mod1`: GMM parameter to $JS(mod1||mod2)$.
- `mod2`: GMM parameter to $JS(mod1||mod2)$.

- `nsamp`: number of samples used to build estimate.

returns: JS divergence value.

localisation: `misc/Utils.c`, `interfaces.R`

example:

```
temp <- jsmc(gmmpen[[1]], gmmpen[[2]])
```

see:

2.27 klut

```
klut(mod1, mod2)
```

compute Unscented Transform approximation to KL divergence between GMM.

parameters:

- `mod1`: GMM parameter to $KL(mod1||mod2)$.
- `mod2`: GMM parameter to $KL(mod1||mod2)$.

returns: KL value.

localisation: `misc/Utils.c`, `interfaces.R`

example:

```
temp <- klut(gmmpen[[1]], gmmpen[[2]])
```

see:

[Goldberger et al., 2003]

2.28 jsut

```
jsut(mod1, mod2)
```

compute Unscented Transform approximation to Jensen-Shannon (JS) divergence between GMM.

parameters:

- `mod1`: GMM parameter to $JS(mod1||mod2)$.
- `mod2`: GMM parameter to $JS(mod1||mod2)$.

returns: JS divergence value.

localisation: `misc/Utils.c`, `interfaces.R`

example:

```
temp <- jsut(gmpen[[1]], gmpen[[2]])
```

see: [Goldberger et al., 2003]

2.29 covgen

```
covgen(d=2, bounds=c(1, 5))
```

generates random definite positive matrices (i.e. valid covariance matrices).

parameters:

- **d:** rank of the square matrix to be returned.
- **bounds:** minima and maximal values for diagonal values.

returns: matrix cells are sampled with an heuristic not guaranteed to lead to definite positiveness: this characteristic is only controlled before function return. If positive definite after control, the matrix is returned. If not, an error message is issued.

localisation: misc/Rutils.R

example:

```
temp <- covgen()
```

see:

2.30 rDirichlet

```
rDirichlet(K, alpha=0.1)
```

samples from the Dirichlet distribution.

parameters:

- **K:** order of the sample.
- **alpha:** α parameter of the distribution (i.e. α repeated K times).

returns: numeric vector, which values are $\in [0, 1]$ and sum to 1.

localisation: misc/utlis.c, interfaces.R

example:

```
temp <- rDirichlet(4)
```

see:

2.31 dDirichlet

```
dDirichlet(alpha=0.1, x1, x2)
```

get density of a sample w.r.t Dirichlet distribution (3D only).

parameters:

- **alpha:** α parameter of the distribution (i.e. α repeated 3 times).
- **x1:** 1st dimension of the sample.
- **x2:** 2nd dimension of the sample.

returns: density value.

localisation: misc.utils.c, interfaces.R

example:

```
temp <- dDirichlet(x1=0.4, x2=0.2)
# 3rd dimension is 1-x1-x2 = 0.2
```

see:

2.32 setDomain

```
setDomain(dat, span=10, oldspan=NULL)
```

performs linear rescaling of given data.

parameters:

- **dat:** data to rescale. **matrix** object, with elements as rows, and variables as columns (i.e. variables are rescaled).
- **span:** new domain to which **dat** is rescaled. If type is **numeric** and length = 1: `[-span, span]` is used for all variables. If type is **numeric** and length = 2: `[span[1], span[2]]` is used for all variables. If a **list** object: `[span[[1]]i, span[[2]]i]` is used for each variable *i*.
- **oldspan:** if **NULL**, old domains are computed from **dat** inspection. Otherwise, is structured as **span** and replaces inspected values for rescaling.

returns: scaled data matrix.

localisation: misc/Rutils.R

example:

```
temp <- setDomain(irisdata, span=15)
```

see:

2.33 generateSparsePoints

```
generateSparsePoints(npoints, dim=2, span=10, mindist=2, maxit=20)
```

generates a set of points pairwise-separated by a minimal distance. Is not guaranteed to converge: when `maxit` is reached, current points are returned.

parameters:

- `npoints`: number of points to generate (i.e. in a matrix with elements as rows).
- `dim`: number of variables to generate.
- `span`: `[-span, span]` is used as bounds to uniform sampling for all variables.
- `mindist`: minimal distance that each element should have with all others. the `control C` routine is used to perform this verification. All points that do not respect this constraint are resampled.
- `maxit`: maximal number of iterations before current elements are returned.

returns: matrix with well separated elements as its rows.

localisation: `misc/Rutils.R`, `control C` routine in `misc/utils.c`

example:

```
temp <- generateSparsePoints(10)
```

see:

2.34 buildFrame

```
buildFrame(datamatrix, labels, dims=1:2)
```

builds a data frame from a `matrix` of elements and a vector of `numeric` labels.

parameters:

- `datamatrix`: matrix of row-elements.
- `labels`: vector of numeric labels.
- `dims`: subset of variables extracted from `datamatrix`.

returns: built data frame.

localisation: `graphics/displayScatter.R`

example:

```
irisdata[c(1,7,35,56,131),]
```

```

Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,] 5.1 3.5 1.4 0.2
[2,] 4.6 3.4 1.4 0.3
[3,] 4.9 3.1 1.5 0.2
[4,] 5.7 2.8 4.5 1.3
[5,] 7.4 2.8 6.1 1.9
irislabels[c(1,7,35,56,131)]
[1] 1 1 1 2 3
temp <- buildFrame(irisdata, irislabels, dims=1:4)

```

see:

2.35 knn

```
knn(data, labels, n=2, KLparam=500)
```

performs k-nearest neighbors over a collection of GMM. It uses `jsmc` to compute distances. Each elements in `data` is classified against all the others, and inferred class is compared to the true one (leave-one-out).

parameters:

- `data`: list of GMM.
- `labels`: vector of numeric labels associated to `data`.
- `n`: k of the algorithm.
- `KLparam`: number of samples for `jsmc`.

returns: classification error ratio $\in [0, 1]$.

localisation: `algorithms/knn.c`, `interfaces.R`

example:

```

temp1 <- sample(1:1243, 150)
temp2 <- list()
for(i in temp1) {
temp2 <- appendToList(temp2, imgmods[[i]])
}
temp3 <- imglabels[temp1]
temp4 <- knn(temp2, temp3)

```

see:

2.36 mergeClassif

```
mergeClassif(data, labels, KLparam=500, rho=new.env())
```

performs task analogous to `knn` (i.e. leave-one-out classification), but uses synthetic representatives to infer labels, instead of k-NN. Each representative is obtained by concatenating all GMM (i.e. elements) of a specific label value, and applying `vbcomp` on this redundant mixture.

parameters:

- `data`: list of GMM.
- `labels`: vector of numeric labels associated to `data`.
- `KLparam`: number of samples for `jsmc`.
- `rho`: R environment object. Used to issue R commands within the C routine.

returns: classification error ratio $\in [0, 1]$.

localisation: `algorithms/knn.c`, `interfaces.R`

example:

```
temp1 <- sample(1:1243, 150)
temp2 <- list()
for(i in temp1) {
  temp2 <- appendToList(temp2, imgmods[[i]])
}
temp3 <- imglabels[temp1]
temp4 <- mergeClassif(temp2, temp3)
```

see:

2.37 constrClassif

```
constrClassif(data, labels, KLparam=500, rho=new.env())
```

performs task analogous to `knn` (i.e. leave-one-out classification), but uses synthetic representatives to infer labels, instead of k-NN. Each representative is obtained by concatenating all GMM (i.e. elements) of a specific label value, and applying `vbconstr` on this redundant mixture.

parameters:

- `data`: list of GMM.
- `labels`: vector of numeric labels associated to `data`.
- `KLparam`: number of samples for `jsmc`.
- `rho`: R environment object. Used to issue R commands within the C routine.

returns: classification error ratio $\in [0, 1]$.

localisation: algorithms/knn.c, interfaces.R

example:

```
temp1 <- sample(1:1243, 150)
temp2 <- list()
for(i in temp1) {
  temp2 <- appendToList(temp2, imgmods[[i]])
}
temp3 <- imglabels[temp1]
temp4 <- constrClassif(temp2, temp3)
```

see:

2.38 sampleClassif

```
sampleClassif(data, labels, KLparam=500, rho=new.env())
```

performs task analogous to `knn` (i.e. leave-one-out classification), but uses synthetic representatives to infer labels, instead of k-NN. Each representative is obtained by concatenating all GMM (i.e. elements) of a specific label value, resampling from this redundant mixture, and applying `varbayes` on this sample.

parameters:

- `data`: list of GMM.
- `labels`: vector of numeric labels associated to `data`.
- `KLparam`: number of samples for `jsmc`.
- `rho`: R environment object. Used to issue R commands within the C routine.

returns: classification error ratio $\in [0, 1]$.

localisation: algorithms/knn.c, interfaces.R

example:

```
temp1 <- sample(1:1243, 150)
temp2 <- list()
for(i in temp1) {
  temp2 <- appendToList(temp2, imgmods[[i]])
}
temp3 <- imglabels[temp1]
temp4 <- sampleClassif(temp2, temp3)
```

see:

2.39 classicEM

```
classicEM(data, k, thres=0.1, maxit=NULL)
```

estimates a GMM on `data` using EM algorithm. A lower bound is calculated and monitored at each iteration.

parameters:

- `data`: matrix of row-elements.
- `k`: maximal number of components in the GMM. In case of degeneracies, the final model size may be less than 0.
- `thres`: threshold for lower bound variations between 2 iterations. Convergence is decided if this variation is below `thres`.
- `maxit`: if `NULL`, the stopping criterion is related to `thres`. If not `NULL`, `maxit` iterations are performed.

returns: estimated GMM with at most k components, with `$labels` containing associated labels for `data` in addition.

localisation: algorithms/classicEM.R

example:

```
temp <- classicEM(irisdata, 4)
```

see:

[Bishop, 2006, chap. 9]

`newGmm`

2.40 varbayes

```
varbayes(data, ncomp, thres=0.1, maxit=NULL)
```

estimates the variational posterior distribution of a GMM on `data` using the variational EM algorithm [Bishop, 2006, chap. 10]. A lower bound is calculated and monitored at each iteration. This posterior can be used for various purposes (e.g. MC proposal distribution). It can be transformed using `extractSimpleModel`, outputting a GMM.

parameters:

- `data`: matrix of row-elements.
- `ncomp`: number of components in the posterior.
- `thres`: threshold for lower bound variations between 2 iterations. Convergence is decided if this variation is below `thres`.

- **maxit**: if `NULL`, the stopping criterion is related to **thres**. If not `NULL`, **maxit** iterations are performed.

returns: estimated posterior with **ncomp** components. Structured in a `list` object as follows:

- **\$alpha**: hyperparameters influencing the active components in the posterior.
- **\$beta**: hyperparameters regarding shaping of the Normal-Wishart posteriors.
- **\$nu**: hyperparameters regarding shaping of the Normal-Wishart posteriors.
- **\$mean**: hyperparameters regarding shaping of the Normal-Wishart posteriors.
- **\$wish**: hyperparameters regarding shaping of the Normal-Wishart posteriors.

localisation: `algorithms/varbayes.c`, `interfaces.R`

example:

```
temp <- varbayes(irisdata, 20)
```

see:

[Bishop, 2006, chap. 10]

2.41 vbcomp

```
vbcomp(models, ncomp, thres=0.1, maxit=NULL)
```

estimates the variational posterior distribution of a GMM that aggregates a collection of GMM. A lower bound is calculated and monitored at each iteration. This posterior can be used for various purposes (e.g. MC proposal distribution). It can be transformed using `extractSimpleModel`, outputting a GMM.

parameters:

- **models**: GMM made with the weighted sum of the collection of GMM to aggregate.
- **ncomp**: number of components in the posterior.
- **thres**: threshold for lower bound variations between 2 iterations. Convergence is decided if this variation is below **thres**.
- **maxit**: if `NULL`, the stopping criterion is related to **thres**. If not `NULL`, **maxit** iterations are performed.

returns: estimated posterior with **ncomp** components.

localisation: `algorithms/vbcomp.c`, `interfaces.R`

example:

```
temp1 <- newGmm()
```

```
for(i in 1:10) temp1 <- appendToGmm(temp1, gmmpen[[i]])
temp2 <- vbcomp(temp1, 50)
```

see:

[Bruneau et al., 2010a]

2.42 vbconstr

```
vbconstr(models, ncomp, thres=0.1, maxit=NULL)
```

estimates the variational posterior distribution of a GMM that aggregates a constrained collection of GMM. A lower bound is calculated and monitored at each iteration. This posterior can be used for various purposes (e.g. MC proposal distribution). It can be transformed using `extractSimpleModel`, outputting a GMM.

parameters:

- **models**: GMM made with the weighted sum of the collection of GMM to aggregate. **\$a** is used to model constraints between components in this GMM.
- **ncomp**: number of components in the posterior.
- **thres**: threshold for lower bound variations between 2 iterations. Convergence is decided if this variation is below **thres**.
- **maxit**: if `NULL`, the stopping criterion is related to **thres**. If not `NULL`, **maxit** iterations are performed.

returns: estimated posterior with **ncomp** components.

localisation: `algorithms/vbconstr.c`, `interfaces.R`

example:

```
temp1 <- newGmm()
for(i in 1:10) temp1 <- appendToGmm(temp1, gmmpen[[i]])
temp2 <- vbconstr(temp1, 50)
```

see:

[Bruneau et al., 2010a]

2.43 mppca

```
mppca(data, ncomp, thres=0.1, maxit=NULL, qmax=NULL)
```

estimates the variational posterior distribution of a MPPCA on a data set. A lower bound is calculated and monitored at each iteration. This posterior can be used for various purposes (e.g. MC proposal distribution). It can be transformed using `mppcaToGmm` and `subMppca`, outputting a GMM.

parameters:

- **data**: matrix of row-elements.
- **ncomp**: number of components in the posterior.
- **thres**: threshold for lower bound variations between 2 iterations. Convergence is decided if this variation is below **thres**.
- **maxit**: if `NULL`, the stopping criterion is related to **thres**. If not `NULL`, **maxit** iterations are performed.
- **qmax**: maximal rank of the posterior factor matrices. If `NULL`, is set to $d - 1$.

returns: estimated posterior MPPCA with **ncomp** components.

localisation: `algorithms/mppca.c`, `interfaces.R`

example:

```
temp <- mppca(pendat, 100, maxit=30, qmax=8)
```

see:

2.44 mmppca

```
mmppca(mods, ncomp, thres=0.1, maxit=NULL)
```

estimates the variational posterior distribution of a MPPCA that aggregates a collection of input MPPCA models. A lower bound is calculated and monitored at each iteration. This posterior can be used for various purposes (e.g. MC proposal distribution). It can be transformed using `mppcaToGmm` and `subMppca`, outputting a GMM. The maximal rank of output factor matrices is determined by the inputs.

parameters:

- **mods**: input MPPCA that concatenates the set of components to aggregate.
- **ncomp**: number of components in the posterior.
- **thres**: threshold for lower bound variations between 2 iterations. Convergence is decided if this variation is below **thres**.
- **maxit**: if `NULL`, the stopping criterion is related to **thres**. If not `NULL`, **maxit** iterations are performed.

returns: estimated posterior MPPCA with **ncomp** components.

localisation: `algorithms/mmppca.c`, `interfaces.R`

example:

```
temp <- newMppca()
```

```
for(i in 1:10) temp <- appendToMppca(temp, pcapen[[i]])
temp2 <- mppca(temp, 100, maxit=30)
```

see:

2.45 gmmksock

```
gmmksock(models, names, ngroups, rho=new.env(), host="127.0.0.1")
```

perform k-means specifically designed for a set of GMM [Bruneau et al., 2010d]. At each iteration, sends information about current prototypes to a server via a socket connection [Bruneau et al., 2010d, see] for info about protocol.

parameters:

- **models:** list of GMM objects.
- **names:** character vector with respective names of the GMM objects.
- **ngroups:** (maximal) number of clusters.
- **rho:** R environment object, used for calls to R functions within C code.
- **host:** IP address of the server for the socket (port 1979).

returns: a set of GMM prototypes, and inferred labels (i.e. associated to the input objects).

localisation: algorithms/gmmksock.cpp, interfaces.R

example:

```
temp1 <- sample(1:1243, 150)
temp2 <- list()
for(i in 1:length(temp1)) temp2 <- appendToList(temp2, imgmods[[temp1[i]])
temp3 <- imgnames[temp1]
temp4 <- gmmksock(temp2, temp3, 5)
```

```
# messages are sent to the server using the following protocol:
TEMPOSPRING 1.0\n
0/[path_to_1st_closest_element_for_centroid_0]
#[path_to_2nd_closest_element_for_centroid_0]
#[... up to 4]/[path_to_2nd_farthest_element_for_centroid_0]#[... up to 4]
/1/[... for all centroids]/\n
[pairwise_distance_centroids_0_and_1_formatted_x.xxxe+-xx]
[pairwise_distance_centroids_0_and_2][... for all n(n-1)/2 possibilities]/#
```

see:

[Bruneau et al., 2010d]

2.46 `pca`

```
pca(dat, ncomp=NULL)
```

transforms a data set, and returns coordinates in the principal basis.

parameters:

- `dat`: matrix of row-elements.
- `ncomp`: number of retained variables in the output result. If `NULL`, all transformed variables are returned.

returns: matrix of transformed row-elements.

localisation: misc/Rutils.R

example:

```
temp <- pca(irisdata, 3)
```

see:

for a review of PCA, and probabilistic variants, see [Tipping and Bishop, 1999, Bruneau et al., 2010b,c].

2.47 `getCouple`

```
couple(vec1, vec2)
```

computes classification error function described in [Fowlkes and Mallows, 1983, Picarougne et al., 2007], a.k.a couple error. In brief, evaluates how elements are gathered similarly, irrespectively of exact label values (adapted to clustering).

parameters:

- `vec1`: vector of numeric labels.
- `vec2`: vector of numeric labels.

returns: classification error $\in [0, 1]$.

localisation: data/couple.c, interfaces.R

example:

```
temp <- classicEM(irisdata, 4)
getCouple(temp$labels, irislabels)
[1] 0.1524832
```

see:

2.48 getLabels

```
getLabels(model, data)
```

gets numeric labels that associates a data set and a GMM.

parameters:

- **model:** GMM.
- **data:** matrix of row-elements.

:

returns: vector of numeric labels, that take values of the respective component indexes in the GMM.

localisation: data/getLabels.c, interfaces.R

example:

```
temp <- classicEM(irisdata, 4)
temp2 <- getLabels(temp, irisdata)
```

see:

2.49 getResp

```
getResp(data, model)
```

get posterior responsibilities of elements in a data set, according to a posterior MPPCA distribution.

parameters:

- **data:** matrix of row-elements.
- **model:** posterior MPPCA.

returns: $n \times k$ matrix (with n the number of row-elements, and k the number of components in the MPPCA) of membership probabilities. (i.e. \mathbf{Z} in [Bruneau et al., 2010b,c])

localisation: algorithms/mppca.cpp, interfaces.R

example:

```
temp <- getResp(pendat, pcapen[[1]])
```

see:

mppca
mmppca

2.50 extractSimpleModel

```
extractSimpleModel(model, labels)
```

extracts a GMM from a posterior variational distribution. Only relevant components (i.e. associated to a significant population) are extracted.

parameters:

- `model`: variational posterior.
- `labels`: boolean indicating whether to extract a label vector. If `TRUE`, `model`, a `list` object, should also contain a `$data` attribute, used to build label vector.

returns: GMM object.

localisation: misc/Utils.c, interfaces.R

example:

```
temp <- varbayes(irisdata, 20)
temp2 <- extractSimpleModel(temp)
```

see:

2.51 subVarbayes

```
subVarbayes(model, thres=2.001)
```

filters a variational posterior GMM, keeping only components with sufficient support.

parameters:

- `model`: variational posterior GMM.
- `thres`: minimal support for component selection.

returns: filtered variational posterior GMM.

localisation: misc/Rutils.R

example:

```
temp <- varbayes(irisdata, 20)
temp2 <- subVarbayes(temp)
```

see:

2.52 `incredMerge`

```
incredMerge(modref, newmod, k=200, nit=100, quick=FALSE)
```

updates a reference MPPCA model with an input distribution.

parameters:

- `modref`: reference MPPCA to update.
- `newmod`: new MPPCA to incorporate.
- `k`: number of components of the output variational posterior.
- `nit`: number of iterations used in the `mmppca` call that performs the update.
- `quick`: boolean parameter transmitted to the `subMppca` routine that shrinks the output variational posterior.

returns: updated variational posterior.

localisation: `misc.Rutils.R`

example:

```
temp <- incredMerge(pcapen[[1]], pcapen[[2]], quick=T)
```

see:

2.53 `ZtoLabels`

```
ZtoLabels(resp)
```

converts a responsibility matrix (\mathbf{Z} in [Bishop, 2006, chap. 9]) to a vector of `numeric` labels.

parameters:

- `resp`: responsibility matrix to convert.

returns: labels vector.

localisation: `misc/Rutils.R`

example:

```
temp <- getResp(pendat, pcapen[[2]])  
temp2 <- ZtoLabels(temp)
```

see:

2.54 getDataLikelihood

```
getDataLikelihood(gmm, dat)
```

gets log-likelihoods associated to a matrix of row-elements.

parameters:

- `gmm`: GMM object.
- `dat`: matrix of row-elements.

returns: numeric vector of log-likelihoods.

localisation: misc/Rutils.R

example:

```
temp <- getDataLikelihood(gmmpen[[3]], pendat)
```

see:

2.55 getBic

```
getBic(gmm, dat)
```

computes BIC criterion [Schwarz, 1978] for a specific GMM and data set.

parameters:

- `gmm`: GMM object.
- `dat`: matrix of row-elements.

returns: BIC estimate.

localisation: misc/Rutils.R

example:

```
temp <- getBic(gmmpen[[1]], pendat)
```

see:

[Schwarz, 1978]

2.56 getVarbayesResp

```
getVarbayesResp(data, model)
```

gets posterior responsibilities for a data set, according to the variational posterior of a GMM.

parameters:

- **data**: matrix of row-elements.
- **model**: variational posterior of a GMM

returns: responsibility matrix (\mathbf{Z} in [Bishop, 2006, chap. 10]) resulting from the parameters.

localisation: misc/Rutils.R

example:

```
temp <- getVarbayesResp(pendat, vbpen[[2]])
```

see:

2.57 getTimestamp

```
getTimestamp()
```

returns current timestamp.

parameters: none

returns: numeric vector with seconds and milliseconds since epoch.

localisation: misc/utils.c, interfaces.R

example:

```
temp <- getTimestamp()
```

see:

2.58 getElapsed

```
getElapsed(stamp)
```

get elapsed time since the moment defined by the parameter.

parameters:

- **stamp**: timestamp of a specific moment, obtained with `getTimestamp`.

returns: numeric vector with seconds and milliseconds of elapsed time.

localisation: misc/utils.c, interfaces.R

example:

```
temp <- getTimestamp()
temp2 <- getElapsed(temp)
```

see:

2.59 `sort_index`

```
sort_index(vec, order=0)
```

returns indexes associated to the sorted values of the parameter vector.

parameters:

- `vec`: vector to be sorted.
- `order`: if 0, ascending order, if 1, descending order.

returns: indexes associated to the sorted input vector.

localisation: misc/utils.c, interfaces.R

example:

```
temp <- rnorm(10)
temp2 <- sort_index(temp)
```

see:

2.60 `getPointer`

```
getPointer(object)
```

gets the internal R pointer (Integer value) to any object. **Warning:** direct usage of internal R pointers should be made with great care.

parameters:

- `object`: any R object.

returns: the internal R pointer to the memory storage of the object (stored in a Integer value).

localisation: misc/utils.c, interfaces.R

example:

```
temp <- rnorm(10)
temp2 <- getPointer(temp)
```

see:

2.61 `getSEXP`

```
getSEXP(pointer)
```

returns the object associated to an internal R pointer. **Warning:** direct usage of internal R pointers should be made with great care.

parameters:

- **pointer:** Integer pointer to an R object.

returns: a reference to the actual R object.

localisation: misc/utills.c, interfaces.R

example:

```
temp <- rnorm(10)
temp2 <- getPointer(temp)
temp3 <- getSEXP(temp2)
```

see:

2.62 datagen

```
datagen(dreal=2, deff=6, npts=200, noise=0.1, genmean=rep(0, dreal), genspan=6,
iso=F)
```

generates data from a random multivariate Gaussian, and adds redundant dimensions by random linear combinations with noise.

parameters:

- **dreal:** dimensionality of the multivariate Gaussian.
- **deff:** dimensionality of the returned sample.
- **npts:** number of elements to be sampled.
- **noise:** noise magnitude for the linear combination.
- **genmean:** mean of the multivariate Gaussian.
- **genspan:** maximal magnitude of the diagonal elements in the covariance matrix. Non-diagonal elements are sampled under constraints of positive-definiteness.
- **iso:** sample from an isotropic multivariate Gaussian (i.e. diagonal covariance matrix).

returns: matrix of sampled row-elements.

localisation: misc/Rutils.R

example:

```
temp <- datagen()
```

see:

2.63 circlegen

```
circlegen(npts=200, radius=10, noise=1)
```

generate data elements along a 2D circle with additional noise.

parameters:

- **npts:** number of elements to generate.
- **radius:** radius of the circle.
- **noise:** determines the width of the circle stroke.

returns: matrix of sampled row-elements.

localisation: misc/Rutils.R

example:

```
temp <- circlegen()
```

see:

2.64 spiralgen

```
spiralgen(radius=10, n=1000, laps=2, noise=1)
```

generates data elements along a spiral with additional noise.

parameters:

- **radius:** determines the radius of a spiral revolution.
- **n:** number of elements to generate.
- **laps:** number of revolutions of the spiral.
- **noise:** determines the width of the spiral stroke.

returns: matrix of sampled row-elements.

localisation: misc/Rutils.R

example:

```
temp <- spiralgen()
```

see:

2.65 generate2Dtransform

```
generate2Dtransform(dims=4)
```

generate a random matrix to transform a 2D signal to higher dimensional spaces.

parameters:

- **dims**: dimensionality of the target space.

returns: a `dims` x 2 matrix defining the transform.

localisation: `misc/Rutils.R`

example:

```
temp <- generate2Dtransform()
```

see:

2.66 dat1sample

```
dat1sample(nelts, gmm, noise, transform, oldbounds=NULL, newbounds=NULL)
```

generates data elements according to SYN1 process (sample from a 2D GMM, linearly transformed with additive noise).

parameters:

- **nelts**: number of elements to generate.
- **gmm**: 2D GMM to be sampled from.
- **noise**: additive noise magnitude.
- **transform**: matrix defining linear transform.
- **oldbounds**: optional argument for sample rescaling. If not NULL, transmitted to `setDomain` as `oldspan`.
- **newbounds**: optional argument for sample rescaling. If not NULL, transmitted to `setDomain` as `newspan`.

returns: matrix of sampled row-elements.

localisation: `misc/Rutils.R`

example:

```
temp <- dat1sample(500, randomGmm(), 1, generate2Dtransform())
```

see:

2.67 dat2sample

```
dat2sample(nelts, radius, noise, oldbounds=NULL, newbounds=NULL)
```

generates data elements according to SYN2 process (sample along a semi-sphere with additive noise).

parameters:

- **nelts**: number of elements to generate.
- **radius**: radius of the sphere to sample from.
- **noise**: additive noise magnitude.
- **oldbounds**: optional argument for sample rescaling. If not NULL, transmitted to `setDomain` as `oldspan`.
- **newbounds**: optional argument for sample rescaling. If not NULL, transmitted to `setDomain` as `newspan`.

returns: matrix of sampled row-elements.

localisation: misc/Rutils.R

example:

```
temp <- dat2sample(500, 10, 1)
```

see:

2.68 dat3sample

```
dat3sample(nelts, radius, noise, transform, oldbounds=NULL, newbounds=NULL)
```

generates data elements according to SYN3 process (sample along a 2D circle with additive noise, and linearly transform to higher dimensional space with further noise addition).

parameters:

- **nelts**: number of elements to generate.
- **radius**: radius of the sphere to sample from.
- **noise**: additive noise magnitude.
- **transform**: matrix defining linear transform.
- **oldbounds**: optional argument for sample rescaling. If not NULL, transmitted to `setDomain` as `oldspan`.
- **newbounds**: optional argument for sample rescaling. If not NULL, transmitted to `setDomain` as `newspan`.

returns: matrix of sampled row-elements.

localisation: misc/Rutils.R

example:

```
temp <- dat3sample(500, 10, 1, generate2Dtransform())
```

see:

2.69 plotGmm

```
plotGmm(mod)
```

3D density plot of a 2D GMM.

parameters:

- **mod:** GMM object to plot

returns: a new plotting window with the 3D density plot.

localisation: misc/Rutils.R

example:

```
plotGmm(randomGmm())
```

see:

2.70 displayScatter

```
displayScatter(data=NULL, model=NULL, labels=NULL, datasizes=NULL, compcolors=NULL,  
complabels=NULL, compstrokes="solid", space=1:2, xlim=NULL, ylim=NULL, main="", xlab="",  
ylab="", smooth=FALSE, alphacol=0.8, alphanocol=0.5, cex.lab=1, lwd=1)
```

general plotting function for data sets (matrix of row-elements), optionally associated to labels and a GMM. Labels influence the color and symbols of plotted data points. Gaussian envelopes of the components in the GMM are drawn. NB: data set and GMM arguments cannot be both NULL.

parameters:

- **data:** matrix of row-elements. If NULL, the GMM is plotted alone.
- **model:** GMM object.
- **labels:** vector of numeric labels. May alternatively be present as a member of `model`, `$labels`.
- **datasizes:** vector of integer magnification factors for data symbols. If `length=1`, same coefficient applies to all points.

- **compcolors**: vector of integer color indexes. These indexes are internally associated to one color among a set of appropriately chosen ones. If length=1, all GMM components are colored the same way. If length=k, each component is associated to its own color index. This k-length vector may contain NA values: associated components will be white-colored.
- **complabels**: **character** vector containing text strings to be printed over Gaussian envelopes.
- **compstrokes**: this character vector may be used to specify non default strokes for envelopes.
- **space**: this function prints a 2D scatterplot. If data and model have higher dimensionality, this argument specifies the axes to be printed.
- **xlim**: bounds for the first variable. If NULL, will be inferred from available data.
- **ylim**: bounds for the second variable. If NULL, will be inferred from available data.
- **main**: main label for the plotting window.
- **xlab**: label for the x-axis.
- **ylab**: label for the y-axis.
- **smooth**: if TRUE, display the response to a kernel density function, instead of symbols for data elements.
- **alphacol**: alpha blending parameter when a component is non-white colored.
- **alphanocol**: alpha blending parameter when a component is white colored.
- **cex.lab**: magnification factor for all text in the plotting window.
- **lwd**: width of the stroke used for data symbols.

returns: a new plotting window displaying the data set and associated model.

localisation: graphics/displayScatter.R

example:

```
displayScatter(irisdata, NULL, irislabels)
```

see:

2.71 displayGraph

```
displayGraph(measure, dev, vect, xlab="K", ylab="measure", main=" ")
```

displays a curve (vect, measure), and associated deviations. Typically used to present experimental results.

parameters:

- **measure**: y-axis for the curve.

- **dev**: deviations for the y-axis measures.
- **vect**: x-axis for the curve.
- **xlab**: label for x-axis.
- **ylab**: label for y-axis.
- **main**: main label for the plotting window.

returns: a new plotting window displaying the curve.

localisation: graphics/displayGraph.R

example:

```
displayGraph(rnorm(10, mean=4, sd=3), rnorm(10, mean=0, sd=0.5), 1:10)
```

see:

2.72 displaySVM

```
displaySVM(svm.model, dataframe, displayPoints=TRUE, subset=NULL, steps=100, alpha=0.4,
lwd=1)
```

displays the colored decision regions of a SVM model. Data symbols are also optionally displayed. Data and model should be 2D.

parameters:

- **svm.model**: a SVM model, as returned by `svm` (e1071 library)
- **dataframe**: `data.frame` object, containing row-elements, and associated labels in the last variable.
- **displayPoints**: if `FALSE`, only decision regions are displayed.
- **subset**: vector of indexes of a data subset to be displayed. If `NULL`, all points are displayed.
- **steps**: influences the resolution of the decision regions. Low values will provoke aliasing, high values are slower to be displayed.
- **alpha**: alpha blending parameter between decision regions and data symbols.
- **lwd**: magnification factor for the stroke width used to plot symbols.

returns: a new plotting window displaying SVM decision regions.

localisation: graphics/displayScatter.R

example:

```
# extract 2 first variables and build data.frame
```

```
temp <- buildFrame(irisdata, irislabels)
iris.model <- svm(labels ~ ., data=temp, cost=100, gamma=1)
displaySVM(iris.model, temp)
```

see:

2.73 displayNnet

```
displayNnet(nnet.model, datamatrix, datalabels, subset=NULL, displayPoints=TRUE,
steps=100, alpha=0.4, lwd=1)
```

displays the colored decision regions of a neural network model. Data symbols are also optionally displayed. Data and model should be 2D.

parameters:

- **nnet.model**: a neural network model, as returned by `nnet` (`nnet` library)
- **datamatrix**: a matrix of row-elements.
- **datalabels**: matrix of binary indicator variables for labels (as used by `nnet`).
- **subset**: vector of indexes of a data subset to be displayed. If `NULL`, all points are displayed.
- **displayPoints**: if `FALSE`, only decision regions are displayed.
- **steps**: influences the resolution of the decision regions. Low values will provoke aliasing, high values are slower to be displayed.
- **alpha**: alpha blending parameter between decision regions and data symbols.
- **lwd**: magnification factor for the stroke width used to plot symbols.

returns: a new plotting window displaying decision regions associated to the parametrized neural network.

localisation: `graphics/displayScatter.R`

example:

```
temp <- class.ind(irislabels)
temp2 <- setDomain(irisdata[,1:2], 10)
temp3 <- nnet(temp2, temp, size=10)
displayNnet(temp3, temp2, temp)
```

see:

2.74 Rdct

```
Rdct(vect)
```

performs DCT on a real vector.

parameters:

- `vect`: vector of real values.

returns: vector of DCT transformed values.

localisation: misc/utils.c, interfaces.R

example:

```
temp <- Rdct(irisdata[,1])
```

see:

2.75 Rdct2D

```
Rdct2D(mat)
```

performs 2D DCT on a real matrix.

parameters:

- `mat`: matrix of real values.

returns: matrix of DCT transformed values.

localisation: misc/utils.c, interfaces.R

example:

```
temp <- Rdct2D(irisdata)
```

see:

2.76 RinvDct2D

```
RinvDct2D(mat)
```

performs inverse 2D DCT on a real matrix.

parameters:

- `mat`: matrix of real values.

returns: matrix of inverse DCT transformed values.

localisation: misc/utils.c, interfaces.R

example:

```
temp <- RinvDct2D(Rdct2D(irisdata))
```

see:

2.77 RGBtoLab

```
RGBtoLab(filename, filterWhite=FALSE, addCoords=TRUE)
```

transform a .ppm file into a matrix of (L,a,b) pixel intensities (1 row-element per pixel).

parameters:

- **filename:** path to .ppm file (as build by `names.pl`. Alternatively, if needed, R file path manipulating routines are documented in document `r-lang.pdf`, section 7.1)
- **filterWhite:** if TRUE, filter white points from result to return.
- **addCoords:** if TRUE, append 2 normalized (x,y) coordinates for each pixel.

returns: matrix of pixel row-elements.

localisation: graphics/RGBtoLab.R

example:

```
temp <- RGBtoLab(imgnames[[2]]), filterWhite=TRUE)
```

see:

2.78 readPixmapFile

```
readPixmapFile(name)
```

extracts a list of pixmap objects from the handwritten digits file format provided in [LeCun et al., 1998].

parameters:

- **name:** path to the file.

returns: a list of pixmapGrey objects.

localisation: data/readDataFile.cpp, interfaces.R

example:

```
temp <- readPixmapFile(file.path(VBmixwd, "data/train-images-idx3-ubyte"))
```

see:

[LeCun et al., 1998]

2.79 readLabelFile

```
readLabelFile(name)
```

reads the vector of numeric labels contained in a binary file. Labels are associated to handwritten digits, thus $\in [0 - 9]$.

parameters:

- **name:** path to the file.

returns: vector of digit labels.

localisation: data/readLabelFile.cpp, interfaces.R

example:

```
temp <- readLabelFile(file.path(VBmixwd, "data/train-labels-idx1-ubyte"))
```

see:

2.80 reBuild

```
reBuild(v, voids, nonvoids, domains, placeholder=1)
```

re-build a pixmapGrey object from a vector of pixel intensities. As some pixels may be irrelevant over a collection of images (e.g. pixel always white in handwritten digits), some variables may have been filtered or transformed before performing some machine learning process. These transforms are indicated as parameters, and give clues to recover objects in the original image space. NB: assumes that v is scaled in $[-10, 10]$. Additional transformations may thus be performed as appropriate before using this function.

parameters:

- **v:** vector to be converted to a pixmapGrey object.
- **voids:** vector of position indices in the original signal (i.e. 2D matrix with its columns casted in a vector) that did not carry any information. Replaced by a placeholder in recovered image.
- **nonvoids:** vector of positions to which v should be associated in the recovered image.
- **domains:** original data domains of pixel intensities prior to being transformed to v 's domain. Permit appropriate reconstruction in the domain of pixel intensities used by `pixmap` (i.e. subset of $[0, 1]$). Formatted similarly to what is required in `setDomain`.

- **placeholder**: placeholder value for pixel positions present in **voids**.

returns: `pixmapGrey` reconstructed object.

localisation: `misc/Rutils.R`

example:

```
temp <- rebuild(handdat[123,], handvoid, handnonvoid, handdomains)
```

see:

2.81 `pixmapToVector`

```
pixmapToVector(p)
```

converts a `pixmapGrey` object to a numeric vector. The pixel matrix is casted to a vector by appending successive columns.

parameters:

- **p**: `pixmapGrey` object.

returns: numeric vector containing pixel intensities.

localisation: `misc/Rutils.R`

example:

```
temp <- readPixmapFile(file.path(VBmixwd, "data/train-images-idx3-ubyte"))
temp2 <- pixmapToVector(temp[[3]])
```

see:

3 data sets index

Samples of data sets used for illustrations.

- **pendat**: matrix 2000 x 16 of real row-elements.
Subsample from a data set obtained at <http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits>. See [Alimoglu, 1996].
- **penlab**: vector of numeric labels associated to **pendat**.
- **handdat**: matrix 2000 x 717 of real row-elements.
Subsample from the handwritten digits collections obtainable at <http://yann.lecun.com/exdb/mnist/>. See [LeCun et al., 1998]. Original data is provided in `data/train-images-idx3-ubyte`. May be loaded into R with `readDataFile`.
handdat was built using `pixmapToVector` and filtering variables with zero entropy.

- **handlab**: vector of numeric labels associated to **handdat**.
- **handdomains**: original domains of non-void pixels in the handwritten digits collection, to be used along with **reBuild**.
- **handvoid**: vector of void pixel indices.
- **handnonvoid**: vector of non-void pixel indices.
- **irisdata**: matrix 150 x 4 of row-elements, extracted from **iris** standard **data.frame** (4 first variables). See [Fisher, 1936].
- **irislabels**: vector of numeric labels associated to **irisdata**.
- **gmmpen**: list of 20 GMM objects, estimated on subsets of the original 10992-elements **pendat** data set.
- **pcapen**: list of 20 MPPCA posterior objects, estimated on subsets of the original 10992-elements **pendat** data set.
- **vbpen**: list of 20 variational posterior GMM objects, estimated on subsets of the original 10992-elements **pendat** data set.
- **handmods**: list of 20 MPPCA posterior objects, estimated on subsets of the original 60000-elements **handdat** data set.
- **imgmods**: list of 1243 3D GMM, one for each image in **data/256mod**. Built using **RGBtoLab** and **varbayes**. See [Griffin et al., 2007] for information about **data/256mod**.
- **imglabels**: vector of numeric labels, indicating the directory in **data/256mod** associated to respective elements in **imgmods**.
- **imgnames**: absolute file paths of respective elements in **imgmods**.

4 PL scripts

A subset of Caltech-256 image collection [Griffin et al., 2007] may be found in **data/256mod** directory. More specifically, the 10 first categories (each in its separate sub-directory) were selected.

Some PL scripts were designed in order to process this collection of images:

- **clean.pl**: removes **.ppm** files from all sub-directories (i.e. relatively from the directory where the script is located and executed).
- **convert.pl**: creates **.ppm** copies of all image files in sub-directories.
- **crop.pl**: scales down all images to approximately 10000 pixels.
- **names.pl**: writes all absolute **.ppm** files names in **names.txt**. Numeric labels associated to every image's respective directory are also written in **labels.txt**.
- **resize.pl**: scales down all image files in sub-directories.

- `sizes.pl`: prints all sizes (w x h) of all images in all sub-directories. Results are written in `sizes.txt`.

NB: `imgnames` in section 3 is obtained by the following hack:

```
imgnames <- as.character(read.table("[path_to_VBmix]/VBmix/data/256mod/labels.txt")[,1])
```

And `imglabels` respectively by using `as.numeric`.

5 socket client and server

`gmmkmssock` includes a socket client, which sends data at each iteration to an adapted server. `misc/sockserver` was designed to provide a simple testing environment for this protocol. The socket client was also proposed as a standalone for testing purposes in `misc/sockclient`.

These should be used as follows:

- **sockserver**: as a command line, launches the server on port 1979. Prints data on std out as soon as received on the socket.
- **sockclient [server_IP]**: as a command line, connects a client to the server located a `server_IP`, port 1979. If no IP specified, tries to connect to 127.0.0.1. Sends a static text string every 2 seconds.

References

- F. Alimoglu. Combining multiple classifiers for pen-based handwritten digit recognition. Technical report, Institute of Graduate Studies in Science and Engineering, 1996.
- C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.
- P. Bruneau, M. Gelgon, and F. Picarougne. **Parsimonious reduction of Gaussian mixture models with a variational-Bayes approach**. *Pattern Recognition*, 43:850–858, Mars 2010a.
- P. Bruneau, M. Gelgon, and F. Picarougne. **A variational-Bayes technique for aggregating probabilistic PCA mixtures from their parameters**. Tech report, extended version of ICPR'2010, 30 pages, submitted to a journal, <http://hal.archives-ouvertes.fr/docs/00/47/60/76/pdf/pami.pdf>, HAL, Juin 2010b.
- P. Bruneau, M. Gelgon, and F. Picarougne. **Aggregation of probabilistic PCA mixtures with a variational-Bayes technique over parameters**. *Proceedings of IEEE/IAPR International Conference on Pattern Recognition (ICPR'10)*, Istanbul (Turkey), August 2010c.
- P. Bruneau, F. Picarougne, and M. Gelgon. **Interactive unsupervised classification and visualization for browsing an image collection**. *Pattern Recognition*, 43(2):485–493, 2010d.
- R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7 (Part II):179–188, 1936.

- E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *J. Am. Stat. Assoc.*, 78:553–569, 1983.
- J. Goldberger, S. Gordon, and H. Greenspan. An efficient image similarity measure based on approximations of KL-divergence between two Gaussian mixtures. *Proceedings of the 9th IEEE International Conference on Computer Vision*, 1:487–493, 2003.
- G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset, <http://authors.library.caltech.edu/7694>. Technical Report 7694, California Institute of Technology, 2007.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- F. Picarougne, H. Azzag, G. Venturini, and C. Guinot. A new approach of data clustering using a flock of agents. *Evolutionary Computation*, 15(3):345–367, 2007.
- G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.
- M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society - B Series*, 61(3):611–622, 1999.