MixAll: Learning mixture models

Serge Iovleff
University Lille 1

Abstract

The MixALL package can also be used in order to learn mixture models when the labels class are known. This short vignette assume that you have already read the vignette "Clustering With MixAll" (Iovleff (2016)).

Keywords: R, C++, STK++, Learning, missing values.

1. Introduction

It is possible to perform supervised learning with MixAll when the labels of the individuals are known. Let us recall the notations defined in the Iovleff (2016) vignette. $\mathcal{X}$ denote an arbitrary measurable space, $\mathcal{Z} = \{1, \ldots, K\}$ is the label set and $(\mathbf{x}, \mathbf{z}) = \{(\mathbf{x}_1, \mathbf{z}_1), ..., (\mathbf{x}_n, \mathbf{z}_n)\}$ represents n independent vectors in $\mathcal{X} \times \mathcal{Z}$ such that each $\Pr(\mathbf{z}_i = k) = p_k$ and such that conditionnaly to $\mathbf{z}_i = k$, $\mathbf{x}_i$ arises from a probability distribution with density

$$h(\mathbf{x}_i|\boldsymbol{\lambda}_k, \boldsymbol{\alpha}) \tag{1}$$

parameterized by $\boldsymbol{\lambda}_k$ and $\boldsymbol{\alpha}$.

Given the matrix of obervation$\mathbf{x}$, and the vector of labels $\mathbf{z}$, the learning methods will estimate the unknown parameters $\boldsymbol{\lambda}_k$ and $\boldsymbol{\alpha}$.

2. Learning with MixAll

Learning analysis can be performed with the functions

1. `learnDiagGaussian` for diagonal Gaussian mixture models,
2. `learnCategorical` for Categorical mixture models,
3. `learnPoisson` for Poisson mixture models,
4. `learnGamma` for gamma mixture models,
5. `learnKernel` for kernel mixture models,
6. `learnMixedData` for MixedData mixture models.

These functions have a common set of parameters with default values given in the table 1.

Input Parameter	Description
`data`	A matrix (or a list of matrix for mixed data) with the data to learn.
`labels`	A vector with the classes of each individuals. Values must be between 1 and K.
`models`	A `vector` with the models to adjust to each data set in case of mixed data, or a set of models to try to adjust. Default is `cluster*Names()` where '*' stands for DiagGaussian, Poisson, Gamma or Categorical.
`prop`	A vector of size K with the proportions of each class. If `prop` is `NULL` then the proportions are computed using the empirical distribution of the `labels`.
`algo`	A string defining the algorithm to use for the missing values. Possible values `"impute"`, `"simul"`.
`nbIter`	maximal number of iteration to perform. Default value is 100. Note that if there is no missing values, it should be 1.
`epsilon`	threshold to use in order to stop the iterations (not used by the `"simul"` algorithm). Default value 1e-08.
`criterion`	A string defining the model selection criterion to use. The best model is the one with the lowest criterion value. Possible values: `"AIC"`, `"BIC"`, `"ICL"`. Default is `"ICL"`.
`nbCore`	An integer defining the number of processor to use. Default is 1, 0 for all cores.

Table 1: List of common parameters of the learning functions.

The `learnKernel` function has two more arguments described in table 2.

Input Parameter	Description
`kernelName`	A `string` defining the kernel to use. Use a `"gaussian"` kernel by default. Possible values are `"gaussian"`, "polynomial" or "exponential".
`kernelParameters`	A vector with the kernel parameter value(s). Default value is 1.

Table 2: List of all the specific parameters of the `learnKernel` function.

2.1. Learning Multivariate (diagonal) Gaussian Mixture Models

Multivariate Gaussian mixture models (without correlations) can be learned using the `learnDiagGaussian` function. We illustrate this function with the well known geyser data set (Azzalini and Bowman (1990),Härdle (1991)).

```
> data(iris);
> x <- as.matrix(iris[,1:4]); z <- as.vector(iris[,5]); n <- nrow(x); p <- ncol(x);
```

```
> indexes <- matrix(c(round(runif(5,1,n)), round(runif(5,1,p))), ncol=2);
> cbind(indexes, x[indexes]) # display true values
```

```
     [,1] [,2] [,3]
[1,]   77    3  4.8
[2,]   78    1  6.7
[3,]   55    2  2.8
[4,]  123    2  2.8
[5,]  142    2  3.1
```

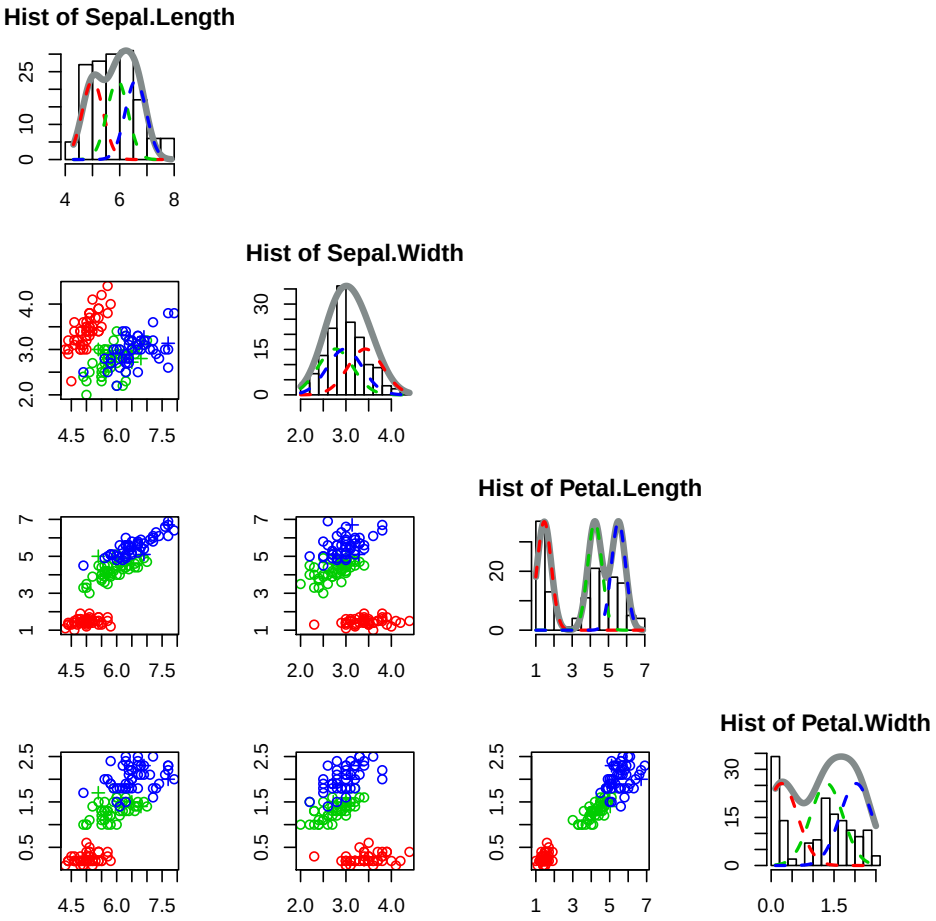
```
> x[indexes] <- NA;          # set them as missing
> model <- learnDiagGaussian(data=x, labels = z, models = clusterDiagGaussianNames(prop =
> summary(model)
```

```
************************************************************
* nbSample       =  150
* nbCluster      =  3
* lnLikelihood   =  -1037.382
* nbFreeParameter=  70
* criterion      =  2425.509
* model name     = gaussian_p_s
************************************************************
```

```
> missingValues(model)
```

```
  row col    value
1  78   1 5.391428
2  55   2 2.720444
3 123   2 3.135887
4 142   2 3.276110
5  77   3 5.027486
```

```
> plot(model)
```



2.2. Learning Multivariate categorical Mixture Models

Categorical (nominal) data can be learned using the `learnCategorical` function.
We illustrate this function with the birds data set.

```
> data(birds)
> ## add 10 missing values
> x <- as.matrix(birds[,2:5]); z <- as.vector(birds[,1]); n <- nrow(x); p <- ncol(x);
> indexes <- matrix(c(round(runif(5,1,n)), round(runif(5,1,p))), ncol=2);
> cbind(indexes, x[indexes]) # display true values

     [,1] [,2] [,3]
[1,] "59" "2"  "none"
[2,] "68" "3"  "white"
[3,] "64" "2"  "none"
[4,] "32" "2"  "none"
[5,] "51" "2"  "none"

> x[indexes] <- NA;          # set them as missing
> model <- learnCategorical( data=x, labels=z
```

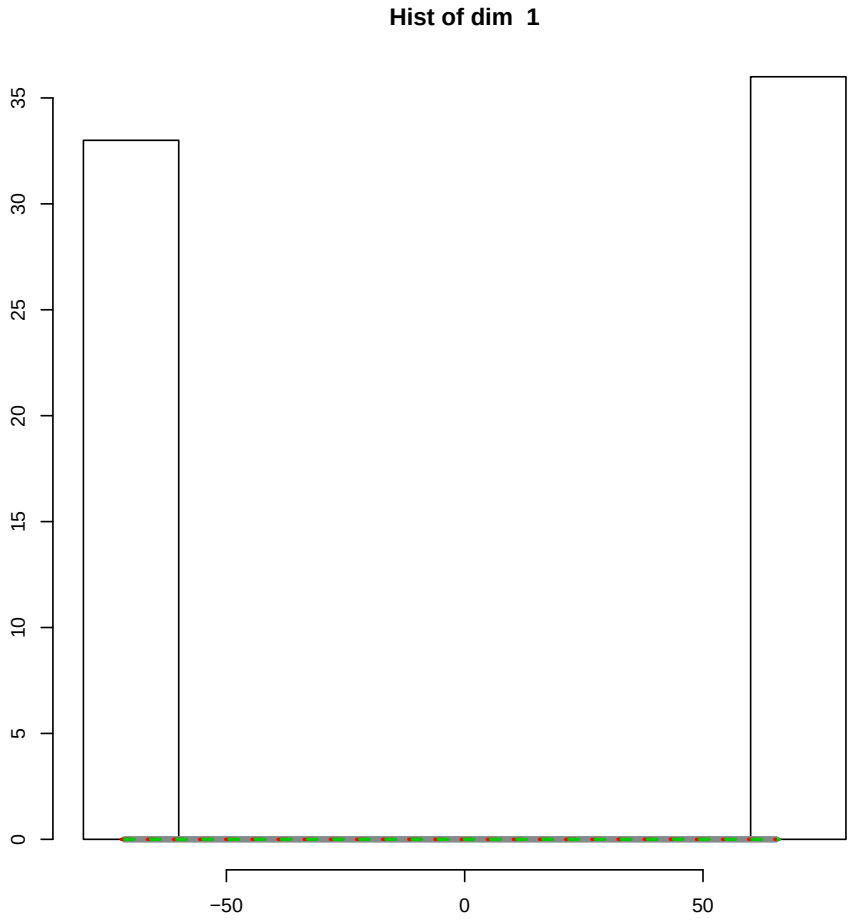
```
+                          , models = clusterCategoricalNames(prop = "equal")
+                          , algo="simul", nbIter = 2)
> summary(model)
```

```
************************************************************
* nbSample       =  69
* nbCluster      =  2
* lnLikelihood   =  -554.2009
* nbFreeParameter=  25
* criterion      =  1214.255
************************************************************
* levels of the variables =
[1] "none           , poor pronounced, pronounced     , very pronounced"
[2] "dotted, none  "
[3] "black        , black & WHITE, black & white, white        "
[4] "few , many, none"
* nbModalities   =  4
************************************************************
```

```
> missingValues(model)
```

```
  row col value
1  32   2     1
2  51   2     4
3  59   2     3
4  64   2     2
5  68   3     1
```

```
> plot(model)
```



2.3. Learning Multivariate Gamma Mixture Models

Gamma data can be learned using the `learnGamma` function.

We illustrate this function with the iris data set.

```
> data(iris)
> x <- as.matrix(iris[,1:4]); z <- as.vector(iris[,5]); n <- nrow(x); p <- ncol(x);
> indexes <- matrix(c(round(runif(5,1,n)), round(runif(5,1,p))), ncol=2);
> cbind(indexes, x[indexes]) # display true values

     [,1] [,2] [,3]
[1,]   84    3  5.1
[2,]   41    3  1.3
[3,]  108    2  2.9
[4,]  101    3  6.0
[5,]   68    3  4.1

> x[indexes] <- NA;          # set them as missing
> model <- learnGamma( data=x, labels= z,
+                     , models = clusterGammaNames(prop = "equal")
```

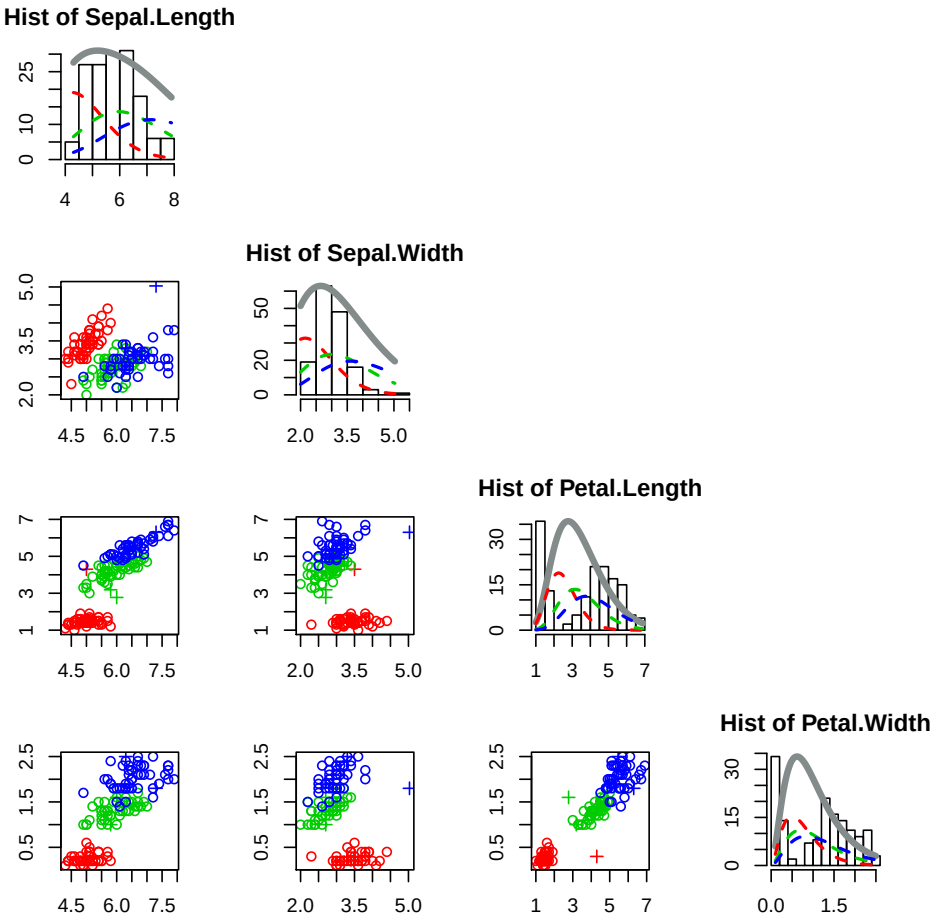
```
+                     , algo = "simul", nbIter = 2, epsilon = 1e-08
+                  )
> summary(model)
```

```
************************************************************
* nbSample       =  150
* nbCluster      =  3
* lnLikelihood   =  -7574.38
* nbFreeParameter=  142
* criterion      =  15860.27
* model name     = gamma_p_aj_bk
************************************************************
```

```
> missingValues(model)
```

```
  row col    value
1 108   2 5.029335
2  41   3 4.299906
3  68   3 3.186405
4  84   3 2.771776
5 101   3 5.455133
```

```
> plot(model)
```



2.4. Learning Multivariate Poisson Models

Poisson data (count data) can be learned using the `learnPoisson` function.

We illustrate this function with the debTrivedi data set.

```
> data(DebTrivedi)
> x <- DebTrivedi[, c(1, 6, 8, 15)]; z <- DebTrivedi$medicaid; n <- nrow(x); p <- ncol(x);
> indexes <- matrix(c(round(runif(5,1,n)), round(runif(5,1,p))), ncol=2);
> cbind(indexes, x[indexes]) # display true values

     [,1] [,2] [,3]
[1,] 1556    1    1
[2,] 1138    2    4
[3,] 2695    1    1
[4,] 2102    4   12
[5,] 3249    4   12

> x[indexes] <- NA;           # set them as missing
> model <- learnPoisson( data=x, labels=z
+                       , models = clusterPoissonNames(prop = "equal")
```

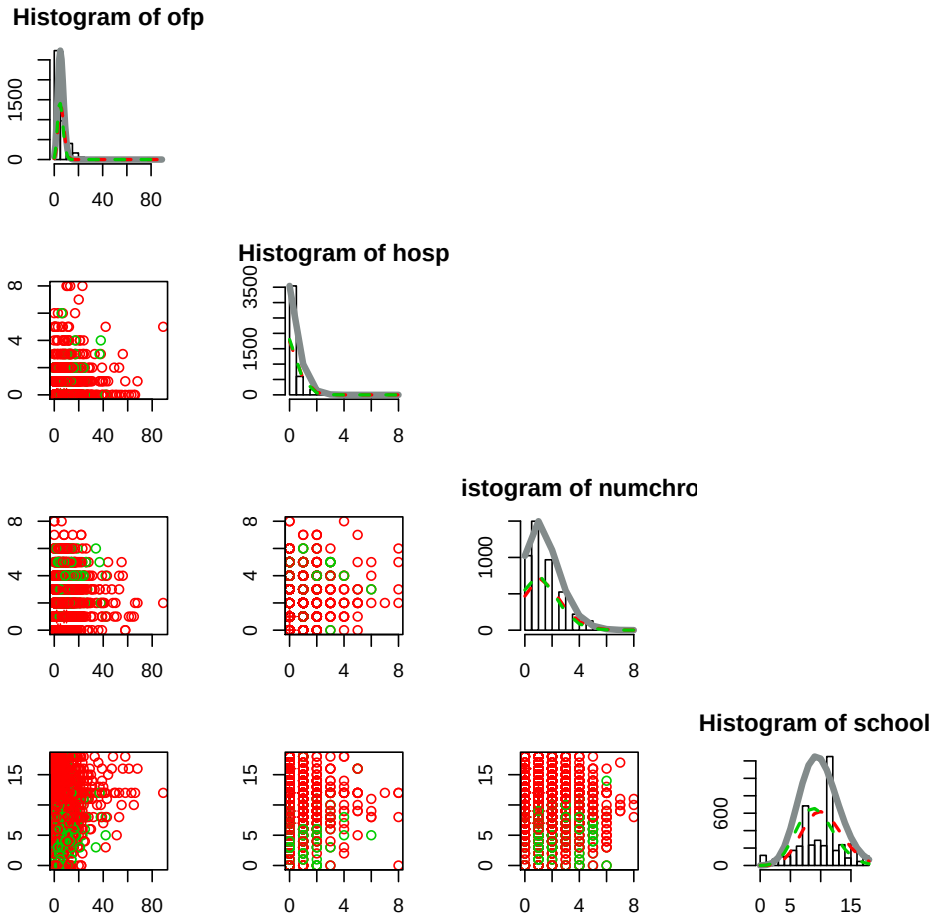
```
+                       , algo="simul", nbIter = 2, epsilon =  1e-08
+ )
> summary(model)
```

```
**************************************************************
* nbSample       =  4406
* nbCluster      =  2
* lnLikelihood   =  -161264.3
* nbFreeParameter=  17
* criterion      =  322671.2
* model name     = poisson_p_ljlk
**************************************************************
```

```
> missingValues(model)
```

```
   row col value
1 1556   1     5
2 2695   1    10
3 1138   2     0
4 2102   4     7
5 3249   4    16
```

```
> plot(model)
```



2.5. Learning Kernel Mixture Models

Data can be clustered using the `learnKernel` function.

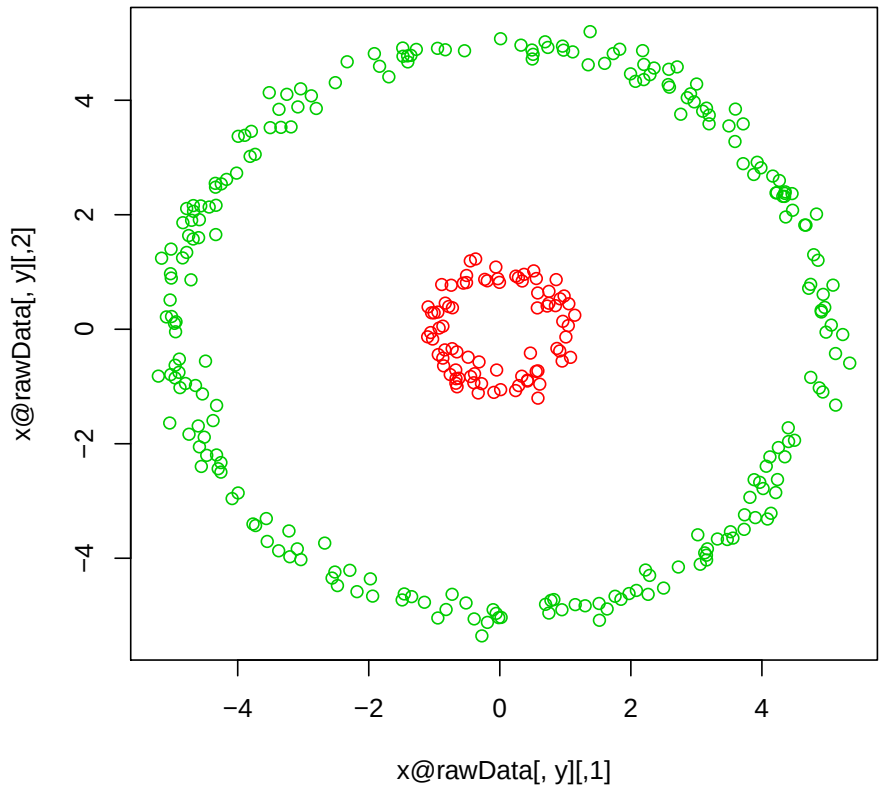
We illustrate this function with the bullsEye data set.

```
> data(bullsEye)
> x <- bullsEye[,1:2]; x = as.matrix(x); n <- nrow(x); p <- ncol(x);
> z <- bullsEye[,3];
> model <- learnKernel( data=x, labels=z
+                     , models = clusterKernelNames(prop = "equal")
+                     , dim = 50, kernelName = "gaussian", kernelParameters = 1.
+                     , algo="impute", nbIter = 1, epsilon =  1e-08
+                     )
> summary(model)

************************************************************
* nbSample        =  320
* nbCluster       =  2
* lnLikelihood    =  17883.62
* nbFreeParameter=  12
```

```
* criterion      =  -35698.02
* model name     = kernelGaussian_p_sk
*************************************************************

> plot(model)
```



2.6. Learning Mixed data sets

Mixed data sets can be learned using the `learnMixedData` function. The original mixed data set has to be splited in multiple homogeneous data sets and each one associated to a mixture model name.

We illustrate this function with the HeartDisease data set.

```
> data(HeartDisease.cat)
> data(HeartDisease.cont)
> data(HeartDisease.target)
> ldata = list(HeartDisease.cat, HeartDisease.cont);
> models = c("categorical_pk_pjk","gaussian_pk_sjk")
> z<-HeartDisease.target[[1]];
```

```
> model <- learnMixedData(ldata, models, z, algo="simul", nbIter=2)
> summary(model)
```

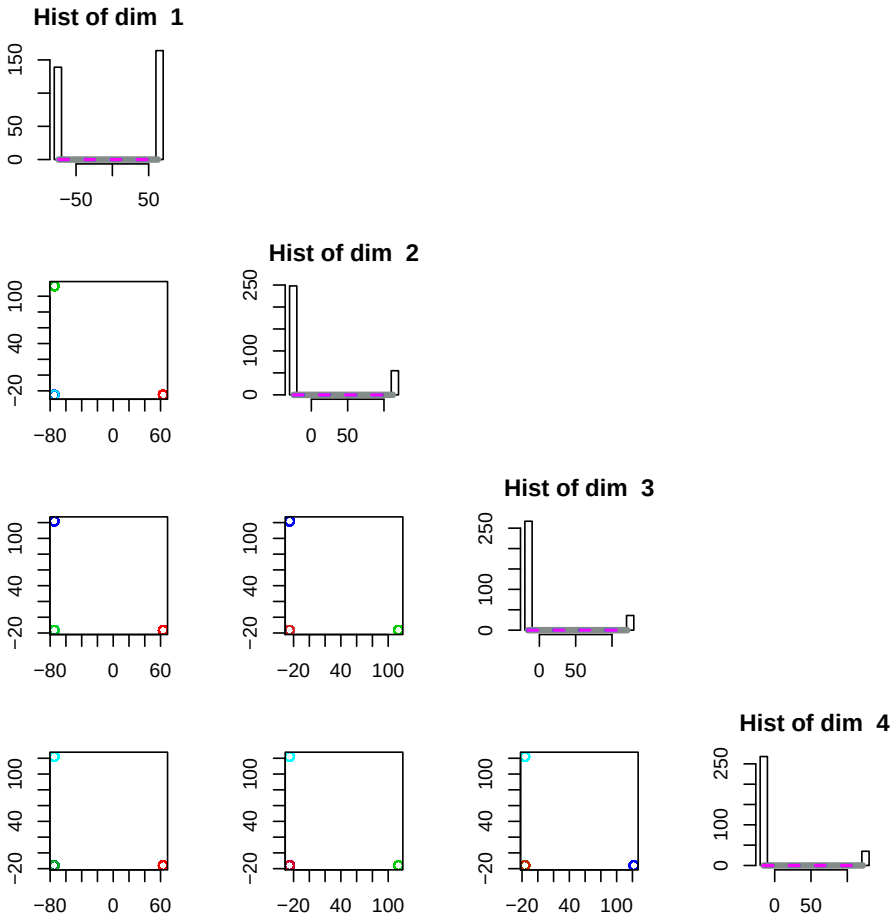
```
************************************************************
* nbSample       =  303
* nbCluster      =  5
* lnLikelihood   =  -7566.794
* nbFreeParameter=  129
* criterion      =  15870.66
************************************************************
```

```
> missingValues(model)
```

```
[[1]]
     row col value
[1,] 167   7     1
[2,] 193   7     2
[3,] 288   7     1
[4,] 303   7     1
[5,]  88   8     1
[6,] 267   8     3

[[2]]
     row col value
```

```
> plot(model)
```



References

Azzalini A, Bowman AW (1990). "A look at some data on the Old Faithful geyser." *Applied Statistics*, pp. 357–365.

Härdle W (1991). *Smoothing techniques: with implementation in S*. Springer Science & Business Media.

Iovleff S (2016). *Clustering With MixAll*. R package version 1.1.1, URL http://CRAN.R-Project.org/package=MixAll.

Affiliation:

Serge Iovleff
Univ. Lille 1, CNRS U.M.R. 8524, Inria Lille Nord Europe
59655 Villeneuve d'Ascq Cedex, France
E-mail: Serge.Iovleff@stkpp.org

URL: http://www.stkpp.org