Package ‘RMixtComp’

June 19, 2023

Type Package
Title Mixture Models with Heterogeneous and (Partially) Missing Data
Version 4.1.4
Date 2023-06-17
Copyright Inria - Université de Lille - CNRS
License AGPL-3
Description Mixture Composer (Biernacki (2015) <https://inria.hal.science/hal-01253393v1>) is a project to perform clustering using mixture models with heterogeneous data and partially missing data. Mixture models are fitted using a SEM algorithm. It includes 8 models for real, categorical, counting, functional and ranking data.
URL https://github.com/modal-inria/MixtComp
BugReports https://github.com/modal-inria/MixtComp/issues
Imports RMixtCompIO(>= 4.0.4), ggplot2, plotly, scales
Depends RMixtCompUtilities (>= 4.1.4), R (>= 3.5.0)
Suggests testthat, xml2, Rmixmod, knitr, ClusVis, rmarkdown
RoxygenNote 7.2.3
Encoding UTF-8
VignetteBuilder knitr
NeedsCompilation no
Author Vincent Kubicki [aut],
Christophe Biernacki [aut],
Quentin Grimonprez [aut, cre],
Matthieu Marbac-Lourdelle [ctb],
Étienne Goffinet [ctb],
Serge Iovleff [ctb],
Julien Vandaele [ctb]
Maintainer Quentin Grimonprez <quentingrim@yahoo.fr>
Repository CRAN
Date/Publication 2023-06-18 22:50:13 UTC
MixtComp (Mixture Composer, https://github.com/modal-inria/MixtComp) is a model-based clustering package for mixed data. It used mixture models (McLachlan and Peel, 2010) fitted using a SEM algorithm (Celeux et al., 1995) to cluster the data.

It has been engineered around the idea of easy and quick integration of all new univariate models, under the conditional independence assumption.

Five basic models (Gaussian, Multinomial, Poisson, Weibull, NegativeBinomial) are implemented, as well as two advanced models: Func_CS for functional data (Same et al., 2011) and Rank_ISR for ranking data (Jacques and Biernacki, 2014).

MixtComp has the ability to natively manage missing data (completely or by interval).

Main functions are mixtCompLearn for clustering, mixtCompPredict for predicting the cluster of new samples with a model learnt with mixtCompLearn. createAlgo gives you default values for required parameters.

Read the help page of mixtCompLearn for available models and data format. A summary of these information can be accessed with the function availableModels.

All utility functions (getters, graphical) are in the RMixtCompUtilities-package package.

In order to have an overview of the output, you can use print.MixtCompLearn, summary.MixtCompLearn and plot.MixtCompLearn functions,

Getters are available to easily access some results (see. mixtCompLearn for output format): getBIC, getICL, getCompletedData, getParam, getProportion, getTik, getEmpiricTik, getPartition, getType, getModel, getVarNames.
RMixtComp-package

You can compute discriminative powers and similarities with functions: `computeDiscrimPowerClass`, `computeDiscrimPowerVar`, `computeSimilarityClass`, `computeSimilarityVar`.

Graphics functions are `plot.MixtComp`, `plot.MixtCompLearn`, `heatmapClass`, `heatmapTikSorted`, `heatmapVar`, `histMisclassif`, `plotConvergence`, `plotDataBoxplot`, `plotDataCI`, `plotDiscrimClass`, `plotDiscrimVar`, `plotProportion`, `plotCrit`.

Datasets with running examples are provided: `titanic`, `CanadianWeather`, `prostate`, `simData`.

Documentation about input and output format is available: `vignette("dataFormat")` and `vignette("mixtCompObject")`.

MixtComp examples: `vignette("MixtComp")` or online `https://github.com/vandaele/mixtcomp-notebook`.

Using ClusVis with RMixtComp: `vignette("dataFormat")`.

References

C. Biernacki. MixtComp software: Model-based clustering/imputation with mixed data, missing data and uncertain data. MISSDATA 2015, Jun 2015, Rennes, France. hal-01253393


See Also

`mixtCompLearn availableModels RMixtCompUtilities-package, RMixtCompIO-package`. Other clustering packages: `Rmixmod`

Examples

data(simData)

# define the algorithm's parameters: you can use createAlgo function
algo <- list(
  nbBurnInIter = 50,
  nbIter = 50,
  nbGibbsBurnInIter = 50,
  nbGibbsIter = 50,
  nInitPerClass = 20,
  nSemTry = 20,
  confidenceLevel = 0.95
)

# run RMixtComp for learning using only 3 variables
resLearn <- mixtCompLearn(simData$dataLearn$matrix, simData$model$unsupervised[1:3], algo,
  nClass = 1:2, nRun = 2, nCore = 1
)
summary(resLearn)
plot(resLearn)

# run RMixtComp for predicting
resPred <- mixtCompPredict(
    simData$dataPredict$matrix, simData$model$unsupervised[1:3], algo,
    resLearn, nCore = 1
)
partitionPred <- getPartition(resPred)
print(resPred)

---

**CanadianWeather**

**Canadian average annual weather cycle**

**Description**

Daily temperature and precipitation at 35 different locations in Canada averaged over 1960 to 1994. Data from fda package.

**Usage**

data(CanadianWeather)

**Format**

A list containing 5 elements:

- tempav: a matrix of dimensions (365, 35) giving the average temperature in degrees celsius for each day of the year.
- precav: a matrix of dimensions (365, 35) giving the average rainfall in millimeters for each day of the year.
- time: sequence from 1 to 365.
- coordinates: a matrix giving 'N.latitude' and 'W.longitude' for each place.
- region: Which of 4 climate zones contain each place: Atlantic, Pacific, Continental, Arctic.

**Source**


**See Also**

Other data: prostate, simData, titanic
Examples

data(CanadianWeather)

# convert functional to MixtComp format
dat <- list(
  tempav = apply(CanadianWeather$tempav, 2, function(x) createFunctional(CanadianWeather$time, x)),
  precav = apply(CanadianWeather$precav, 2, function(x) createFunctional(CanadianWeather$time, x))
)

# create model with 4 subregressions ans 2 coefficients per regression
model <- list(
  tempav = list(type = "Func_CS", paramStr = "nSub: 4, nCoeff: 2"),
  precav = list(type = "Func_CS", paramStr = "nSub: 4, nCoeff: 2")
)

# create algo
algo <- createAlgo()

# run clustering
resLearn <- mixtCompLearn(dat, model, algo, nClass = 2:4, criterion = "ICL", nRun = 3, nCore = 1)
summary(resLearn)

plot(resLearn)

getPartition(resLearn)
getTik(resLearn, log = FALSE)

extractMixtCompObject

**Extract a MixtComp object**

**Description**

Extract a MixtComp object from a MixtCompLearn object

**Usage**

extractMixtCompObject(object, K)
Arguments

object

mixtCompLearn output

K

text

number of classes of the model to extract

Value

a MixtComp object containing the clustering model with K classes

Author(s)

Quentin Grimonprez

Examples

# run clustering
resLearn <- mixtCompLearn(data.frame(x = rnorm(500)),
nClass = 1:3, criterion = "ICL",
nRun = 1, nCore = 1
)

# extract the model with 2 classes
clustModel <- extractMixtCompObject(resLearn, K = 2)

Description

Estimate the parameter of a mixture model or predict the cluster of new samples. It manages heterogeneous data as well as missing and incomplete data.

Usage

mixtCompLearn(

data,  
model = NULL,  
algo = createAlgo(),  
nClass,  
criterion = c("BIC", "ICL"),  
hierarchicalMode = c("auto", "yes", "no"),  
nRun = 1,  
nCore = min(max(1, ceiling(detectCores()/2)), nRun),  
verbose = TRUE
)

mixtCompPredict(
mixtCompLearn

```r
mixtCompLearn(data, model = NULL, algo = resLearn$algo, resLearn, nClass = NULL, nRun = 1, nCore = min(max(1, ceiling(detectCores()/2)), nRun), verbose = FALSE)
```

Arguments

data a data.frame, a matrix or a named list containing the data (see Details and Data format sections).

model a named list containing models and hyperparameters (see Details section).

algo a list containing the parameters of the SEM-Gibbs algorithm (see Details or createAlgo).

nClass the number of classes of the mixture model. Can be a vector for mixtCompLearn only.

criterion "BIC" or "ICL". Criterion used for choosing the best model.

hierarchicalMode "auto", "yes" or "no". If "auto", it performs a hierarchical version of MixtComp (clustering in two classes then each classes is split in two ...) when a functional variable is present (see section Hierarchical Mode).

nRun number of runs for every given number of class. If >1, SEM is run nRun times for every number of class, and the best according to observed likelihood is kept.

nCore number of cores used for the parallelization of the nRun runs.

verbose if TRUE, print some information.

resLearn output of mixtCompLearn (only for mixtCompPredict function).

Details

The data object can be a matrix, a data.frame or a list. In the case of a matrix or data.frame, each column must be names and corresponds to a variable. In the case of a list, each element corresponds to a variable, each element must be named. Missing and incomplete data are managed, see section Data format for how to format them.

The model object is a named list containing the variables to use in the model. All variables listed in the model object must be in the data object. model can contain less variables than data. An element of the list is the model’s name to use (see below for the list of available models). For example, `model <- list(real1 = "Gaussian", counting1 = "Poisson")` indicates a mixture model with 2 variables named real1 and counting1 with Gaussian and Poisson as model. Some models require hyperparameters in this case, the model is described by a list of 2 elements: type containing the model name and paramStr containing the hyperparameters. For example: `model <- list(func1 = list(type = "func_CS", paramStr = "nSub: 4, nCoeff: 2"), counting1 = "Poisson")`. If the model is NULL, data are supposed to be provided in data.frame or list with R format (numeric, factor, character, NA as missing value). Models will be imputed as follows: "Gaussian" for numeric
variable, "Multinomial" for character or factor variable and "Poisson" for integer variable. A summary of available models (and associated hyperparameters and missing format) can be accessed by calling the availableModels function.

Eight models are available in RMixtComp: Gaussian, Multinomial, Poisson, NegativeBinomial, Weibull, Func_CS, Func_SharedAlpha_CS, Rank_ISR. Func_CS and Func_SharedAlpha_CS models require hyperparameters: the number of sub-regressions of functional and the number of coefficients of each sub-regression. These hyperparameters are specified by: \( nSub: i, nCoeff: k \) in the paramStr field of the model object. The Func_SharedAlpha_CS is a variant of the Func_CS model with the alpha parameter shared between clusters. It means that the start and end of each sub-regression will be the same across the clusters.

To perform a (semi-)supervised clustering, user can add a variable named \( z_{\text{class}} \) in the data and model objects with LatentClass as model in the model object.

The algo object is a list containing the different number of iterations for the algorithm. This list can be generated using the createAlgo function. The algorithm is decomposed in a burn-in phase and a normal phase. Estimates from the burn-in phase are not shown in output.

- \( \text{nbBurnInIter} \): Number of iterations of the burn-in part of the SEM algorithm.
- \( \text{nbIter} \): Number of iterations of the SEM algorithm.
- \( \text{nbGibbsBurnInIter} \): Number of iterations of the burn-in part of the Gibbs algorithm.
- \( \text{nbGibbsIter} \): Number of iterations of the Gibbs algorithm.
- \( \text{nInitPerClass} \): Number of individuals used to initialize each cluster (default = 10).
- \( \text{nSemTry} \): Number of try of the algorithm for avoiding an error.
- \( \text{confidenceLevel} \): confidence level for confidence bounds for parameter estimation
- \( \text{ratioStableCriterion} \): stability partition required to stop earlier the SEM
- \( \text{nStableCriterion} \): number of iterations of partition stability to stop earlier the SEM

Value

An object of classes MixtCompLearn and MixtComp for mixtCompLearn function. An object of class MixtComp for mixtCompPredict (see details section).

Data format

See the associated vignette for more details (RShowDoc("dataFormat", package = "RMixtComp").

- Gaussian data: Gaussian data are real values with the dot as decimal separator. Missing data are indicated by a ?. Partial data can be provided through intervals denoted by \([a:b]\) where \(a\) (resp. \(b\)) is a real or -inf (resp. +inf).

- Categorical Data: Categorical data must be consecutive integer with 1 as minimal value. Missing data are indicated by a ?. For partial data, a list of possible values can be provided by \(a_1, \ldots, a_j\), where \(a_i\) denotes a categorical value.

- Poisson and NegativeBinomial Data: Poisson and NegativeBinomial data must be positive integer. Missing data are indicated by a ?. Partial data can be provided through intervals denoted by \([a:b]\) where \(a\) and \(b\) are positive integers. \(b\) can be +inf.
- Weibull Data: Weibull data are real positive values with the dot as decimal separator. Missing data are indicated by a ?. Partial data can be provided through intervals denoted by \([a:b]\) where \(a\) and \(b\) are positive reals, \(b\) can be \(+\inf\).

- Rank data: The format of a rank is: \(o_1, \ldots, o_j\) where \(o_1\) is an integer corresponding to the number of the object ranked in 1st position. For example: 4, 2, 1, 3 means that the fourth object is ranked first then the second object is in second position and so on. Missing data can be specified by replacing and object by a ? or a list of potential object, for example: 4, \([2\ 3]\), \([2\ 1]\), ? means that the object ranked in second position is either the object number 2 or the object number 3, then the object ranked in third position is either the object 2 or 1 and the last one can be anything. A totally missing rank is specified by \(?\,\?,\ldots,\,\?\).

- Functional data: The format of a functional data is: \(time_1:value_1,\ldots, time_j:value_j\). Between individuals, functional data can have different length and different time. \(i\) is the number of sub-regressions in a functional data and \(k\) the number of coefficients of each regression (2 = linear, 3 = quadratic, ...). Missing data are not supported.

- \(z\_class\): To perform a (semi-)supervised clustering, user can add a variable named ‘\(z\_class\)’ (with eventually some missing values) with "LatentClass" as model. Missing data are indicated by a ?. For partial data, a list of possible values can be provided by \(a_1,\ldots, a_j\), where \(a_i\) denotes a class number.

**MixtComp object**

A MixtComp object is a result of a single run of MixtComp algorithm. It is a list containing three elements `mixture`, `variable` and `algo`. If MixtComp fails to run, the list contains a single element: `warnLog` containing error messages.

The `mixture` element contains

- BIC: value of BIC
- ICL: value of ICL
- nbFreeParameters: number of free parameters of the mixture
- lnObservedLikelihood: observed loglikelihood
- lnCompletedLikelihood: completed loglikelihood
- IDClass: entropy used to compute the discriminative power of variable: 
  \[-\sum_{i=1}^n t_{ikj} \log(t_{ikj})/(n^* \log(K))\]
- IDClassBar: entropy used to compute the discriminative power of variable: 
  \[-\sum_{i=1}^n (1 - t_{ikj}) \log((1 - t_{ikj}))/ (n^* \log(K))\]
- delta: similarities between variables (see `heatmapVar`)
- completedProbabilityLogBurnIn: evolution of the completed log-probability during the burn-in period (can be used to check the convergence and determine the ideal number of iteration)
- completedProbabilityLogRun: evolution of the completed log-probability after the burn-in period (can be used to check the convergence and determine the ideal number of iteration)
- runTime: list containing the total execution time in seconds and the execution time of some subpart.
- lnProbaGivenClass: log-proportion + log-probability of \(x\_i\) for each class
The algo list contains a copy of algo parameter with extra elements: nInd, nClass, mode ("learn" or "predict").

The variable list contains 3 lists: data, type and param. Each of these lists contains a list for each variable (the name of each list is the name of the variable) and for the class of samples (z_class). The type list contains the model used for each variable.

Each list of the data list contains the completed data in the completed element and some statistics about them (stat).

The estimated parameter can be found in the stat element in the param list (see Section View of an output object). For more details about the parameters of each model, you can refer to rnorm, rpois, rweibull, rbinom, rmultinom, or references in the References section.

**View of a MixtComp object**

Example of output object with variables named "categorical", "gaussian", "rank", "functional", "poisson", "nBinom" and "weibull" with respectively Multinomial, Gaussian, Rank_ISR, Func_CS (or Func_SharedAlpha_CS), Poisson, NegativeBinomial and Weibull as model.

```
<table>
<thead>
<tr>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>algo</td>
</tr>
<tr>
<td>nbBurnInIter</td>
</tr>
<tr>
<td>nbIter</td>
</tr>
<tr>
<td>nbGibbsBurnInIter</td>
</tr>
<tr>
<td>nbGibbsIter</td>
</tr>
<tr>
<td>nInitPerClass</td>
</tr>
<tr>
<td>nSemTry</td>
</tr>
<tr>
<td>ratioStableCriterion</td>
</tr>
<tr>
<td>nStableCriterion</td>
</tr>
<tr>
<td>confidenceLevel</td>
</tr>
<tr>
<td>mode</td>
</tr>
<tr>
<td>nInd</td>
</tr>
<tr>
<td>nClass</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>mixture</td>
</tr>
<tr>
<td>BIC</td>
</tr>
<tr>
<td>ICL</td>
</tr>
<tr>
<td>lnCompletedLikelihood</td>
</tr>
<tr>
<td>lnObservedLikelihood</td>
</tr>
<tr>
<td>IDClass</td>
</tr>
<tr>
<td>IDClassBar</td>
</tr>
<tr>
<td>delta</td>
</tr>
<tr>
<td>runTime</td>
</tr>
<tr>
<td>nbFreeParameters</td>
</tr>
<tr>
<td>completedProbabilityLogBurnIn</td>
</tr>
<tr>
<td>completedProbabilityLogRun</td>
</tr>
<tr>
<td>lnProbaGivenClass</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>variable</td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>z_class</td>
</tr>
</tbody>
</table>
```
MixtCompLearn object

The MixtCompLearn object is the result of a run of the `mixtCompLearn` function. It is a list containing `nClass`: the vector of number of classes given by user, `res` a list of MixtComp object (one per element of `nbClass`), `criterion` the criterion used to choose the best model, `crit` a matrix containing BIC and ICL for each run, `totalTime`, the total running time, and finally the elements of the MixtComp object with the best criterion value (`algo`, `mixture`, `variable` or `warnLog`).

See the associated vignette for more details: `RShowDoc("mixtCompObject", package = "RMixtComp")`
Hierarchical Mode

When the model’s parameter includes a functional model (Func_CS or Func_SharedAlpha_CS), the algorithm is automatically run in "Hierarchical Mode". In hierarchical mode, it first clusters the data in 2 classes. Then, it searches the best model in 3 classes by performing a clustering in 2 classes of each class of the previous step). The same process is used until the asked number (K) of classes is attained. All models from 2 to K classes are returned (even if the case nClass = K).

This strategy is used to solve some problem (initialization, empty classes...) when the number of classes is high with the functional model.

The user can control the activation of the hierarchical mode using the hierarchicalMode’s parameter. Three values are possible: "no", the algorithm is never run in hierarchical mode, "yes", the algorithm is always run in hierarchical mode, and "auto", the algorithm is run in hierarchical mode only when there is at least one functional variable (default).

Author(s)

Quentin Grimonprez

References


See Also

Graphical and utility functions in RMixtCompUtilities. Other clustering packages: Rmixmod

Examples

data(simData)

# define the algorithm's parameters
algo <- list(
  nbBurnInIter = 50,
  nbIter = 50,
  nbGibbsBurnInIter = 50,
  nbGibbsIter = 50,
  nInitPerClass = 20,
  nSemTry = 20,
  confidenceLevel = 0.95
)

# run RMixtComp in unsupervised clustering mode + data as matrix
resLearn1 <- mixtCompLearn(simData$dataLearn$matrix, simData$model$unsupervised[1:3], algo,
  nClass = 1:2, nRun = 2, nCore = 1)
# run RMixtComp in supervised clustering mode + data as matrix
resLearn2 <- mixtCompLearn(simData$dataLearn$data.frame, simData$model$supervised[1:3], algo,
  nClass = 1:2, nRun = 2, nCore = 1)

# run RMixtComp in predict mode + data as list
resPredict <- mixtCompPredict(simData$dataPredict$list, simData$model$unsupervised[1:3], algo,
  resLearn1,
  nClass = 2, nCore = 1)

plot.MixtCompLearn  Plot of a MixtCompLearn object

Description

Plot of a MixtCompLearn object

Usage

## S3 method for class 'MixtCompLearn'
plot(
  x,
  nVarMaxToPlot = 3,
  nClass = NULL,
  pkg = c("ggplot2", "plotly"),
  plotData = c("CI", "Boxplot"),
  ...
)

Arguments

x  MixtCompLearn object
nVarMaxToPlot  number of variables to display
nClass  number of classes of the model to plot
pkg  "ggplot2" or "plotly". Package used to plot
plotData  "CI" or "Boxplot". If "CI", uses plotDataCI function. If "Boxplot", uses plotDataBoxplot
...  extra parameter for plotDataCI or plotDataBoxplot

Value

ggplot2 or plotly object

Author(s)

Quentin Grimonprez
See Also

mixtCompLearn mixtCompPredict
Other plot: plotCrit()

Examples

data(iris)

# run RMixtComp in unsupervised clustering mode and in basic mode
resLearn <- mixtCompLearn(iris[, -5], nClass = 2:4, nCore = 1)

plot(resLearn)
plot(resLearn, nClass = 3, plotData = "Boxplot")

plotCrit

Plot BIC and ICL

Description

Plot BIC and ICL with regards to the number of classes

Usage

plotCrit(output, crit = c("BIC", "ICL"), pkg = c("ggplot2", "plotly"), ...)

Arguments

output MixtCompLearn object
crit criterion to plot (can be "BIC", "ICL" or c("BIC", "ICL") (default))
pkg "ggplot2" or "plotly". Package used to plot
... arguments to be passed to plot_ly

Value

ggplot2 or plotly object

Author(s)

Quentin Grimonprez

See Also

Other plot: plot.MixtCompLearn()
Examples

data(iris)

# define the algorithm's parameters
algo <- createAlgo()

# keep only 3 variables
model <- list(
  Petal.Width = "Gaussian", Petal.Length = "Gaussian",
  Sepal.Width = "Gaussian", Sepal.Length = "Gaussian"
)

# run RMixtComp in unsupervised clustering mode + data as matrix
res <- mixtCompLearn(iris, model, algo, nClass = 1:4, nCore = 1)

# plot
plotCrit(res)

predict.MixtComp       Predict using RMixtComp

Description

Predict the cluster of new samples.

Usage

## S3 method for class 'MixtComp'
predict(
  object,
  newdata = NULL,
  type = c("partition", "probabilities"),
  nClass = NULL,
  ... )

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>output of mixtCompLearn function.</td>
</tr>
<tr>
<td>newdata</td>
<td>a data.frame, a matrix or a named list containing the data (see Details and Data format sections in mixtCompLearn documentation). If NULL, use the data in object.</td>
</tr>
<tr>
<td>type</td>
<td>if &quot;partition&quot;, returns the estimated partition. If &quot;probabilities&quot;, returns the probabilities to belong to each class (tik).</td>
</tr>
<tr>
<td>nClass</td>
<td>the number of classes of the mixture model to use from object. If NULL, uses the number maximizing the criterion.</td>
</tr>
<tr>
<td>...</td>
<td>other parameters of mixtCompPredict function.</td>
</tr>
</tbody>
</table>
Details

This function is based on the generic method "predict". For a more complete output, use mixtCompPredict function.

Value

if type = "partition", it returns the estimated partition as a vector. If type = "probabilities", it returns the probabilities to belong to each class (tik) as a matrix.

Author(s)

Quentin Grimonprez

See Also

mixtCompPredict

Examples

data(iris)

model <- list(  
  Sepal.Length = "Gaussian", Sepal.Width = "Gaussian",  
  Petal.Length = "Gaussian", Petal.Width = "Gaussian"  
)

resLearn <- mixtCompLearn(iris[-c(1, 51, 101), ], model = model, nClass = 1:3, nRun = 1)

# return the partition
predict(resLearn)

# return the tik for the 3 new irises for 2 and 3 classes
predict(resLearn, newdata = iris[c(1, 51, 101), ], type = "probabilities", nClass = 2)
predict(resLearn, newdata = iris[c(1, 51, 101), ], type = "probabilities", nClass = 3)
**Arguments**

- **x**: `MixtCompLearn` object
- **nVarMaxToPrint**: number of variables to display (including `z_class`)
- **nClass**: number of classes of the model to print
- **...**: Not used.

**Value**

No return value, called for side effects

**Author(s)**

Quentin Grimonprez

**See Also**

`mixtCompLearn` `mixtCompPredict`

**Examples**

```r
data(iris)

# run RMixtComp in unsupervised clustering mode and in basic mode
resLearn <- mixtCompLearn(iris[, -5], nClass = 2:4, nCore = 1)

print(resLearn)
print(resLearn, nClass = 3)
```

---

**prostate**

*Prostate Cancer Data*

**Description**

This data set was obtained from a randomized clinical trial comparing four treatments for \( n = 506 \) patients with prostatic cancer grouped on clinical criteria into two Stages 3 and 4 of the disease.

**Usage**

```r
data(prostate)
```
Format

A list containing of 2 elements `data` and `model`. `data` contains 506 individuals described by 12 variables:

- Age: Age (Continuous)
- HG: Index of tumour stage and histologic grade (Continuous)
- Wt: Weight (Continuous)
- AP: Serum prostatic acid phosphatase C (Continuous)
- SBP: Systolic blood pressure (Continuous)
- PF: Performance rating (Categorical)
- DBP: Diastolic blood pressure (Continuous)
- HX: Cardiovascular disease history (Categorical)
- SG: Serum haemoglobin (Continuous)
- BM: Bone metastasis (Categorical)
- SZ: Size of primary tumour (Continuous)
- EKG: Electrocardiogram code (Categorical)

Source


See Also

Other data: CanadianWeather, simData, titanic

Examples

data(prostate)

algo <- createAlgo(nInitPerClass = 50)

# run clustering
resLearn <- mixtCompLearn(prostate$data, prostate$model, algo,
                           nClass = 2:5, criterion = "ICL",
                           nRun = 3, nCore = 1)

summary(resLearn)
plot(resLearn)
Description
Simulated Heterogeneous data

Usage
data(simData)

Format
A list containing three elements: dataLearn, dataPredict and model.

• dataLearn is a list containing the data in the three accepted format (list, data.frame and matrix). Data consists of 200 individuals and 9 variables.
• dataPredict is a list containing the data in the three accepted format (list, data.frame and matrix). Data consists of 100 individuals and 8 variables.
• model is a list containing the model lists used for clustering model$unsupervised and classification model$supervised.

See Also
Other data: CanadianWeather, prostate, titanic

Examples
data(simData)
str(simData)

---

slopeHeuristic  Slope heuristic

Description
Criterion to choose the number of clusters

Usage
slopeHeuristic(object, K0 = floor(max(object$nClass) * 0.4))

Arguments
object  output of mixtCompLearn
K0  number of class for computing the constant value (see details)
Details

The slope heuristic criterion is: \( LL_k - 2C \times D_k \), with \( LL_k \) the loglikelihood for \( k \) classes, \( D_k \) the number of free parameters for \( k \) classes, \( C \) is the slope of the linear regression between \( D_k \) and \( LL_k \) for \( k > K_0 \).

Value

the values of the slope heuristic

Author(s)

Quentin Grimonprez

References


Jean-Patrick Baudry, Cathy Maugis, Bertrand Michel. Slope Heuristics: Overview and Implementation. 2010. hal-00461639

Examples

data(titanic)

## Use the MixtComp format
dat <- titanic

# refactor categorical data: survived, sex, embarked and pclass
dat$sex <- refactorCategorical(dat$sex, c("male", "female", NA), c(1, 2, "?"))
dat$embarked <- refactorCategorical(dat$embarked, c("C", "Q", "S", NA), c(1, 2, 3, "?"))
dat$survived <- refactorCategorical(dat$survived, c(0, 1, NA), c(1, 2, 3, "?"))
dat$pclass <- refactorCategorical(dat$pclass, c("1st", "2nd", "3rd"), c(1, 2, 3))

# replace all NA by ?
dat[is.na(dat)] <- "?"

# create model
model <- list(
  pclass = "Multinomial",
  survived = "Multinomial",
  sex = "Multinomial",
  age = "Gaussian",
  sibsp = "Poisson",
  parch = "Poisson",
  fare = "Gaussian",
  embarked = "Multinomial"
)

# create algo
algo <- createAlgo()
# run clustering
resLearn <- mixtCompLearn(dat, model, algo, nClass = 2:25, criterion = "ICL", nRun = 3, nCore = 1)

out <- slopeHeuristic(resLearn, K0 = 6)

summary.MixtCompLearn  
MixtCompLearn Object Summaries

Description

Summary of a MixtCompLearn object

Usage

## S3 method for class 'MixtCompLearn'
summary(object, nClass = NULL, ...)

Arguments

object  
MixtCompLearn object

nClass  
number of classes of the model to print

...  
Not used.

Value

No return value, called for side effects

Author(s)

Quentin Grimonprez

See Also

mixtCompLearn print.MixtCompLearn

Examples

data(iris)

# run RMixtComp in unsupervised clustering mode and in basic mode
resLearn <- mixtCompLearn(iris[, -5], nClass = 2:4, nCore = 1)

summary(resLearn)
summary(resLearn, nClass = 3)
titanic

Titanic data set

Description
The data set provides information on the passengers of Titanic.

Usage
data(titanic)

Format
A data.frame with 1309 individuals and 8 variables.

- survived: 0 = No, 1 = Yes (factor)
- pclass: ticket class 1st, 2nd, 3rd (factor)
- sex: male or female (factor)
- age: age in years
- sibsp: number of siblings/spouses aboard the Titanic
- parch: number of parents/children aboard the Titanic
- fare: ticket price in pounds
- embarked: port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton (factor)

Source
https://www.kaggle.com/c/titanic/data

See Also
Other data: CanadianWeather, prostate, simData

Examples

data(titanic)

head(titanic)

## Use the MixtComp format
dat <- titanic

# refactor categorical data: survived, sex, embarked and pclass
dat$sex <- refactorCategorical(dat$sex, c("male", "female", NA), c(1, 2, "?"))
dat$embarked <- refactorCategorical(dat$embarked, c("C", "Q", "S", NA), c(1, 2, 3, "?"))
dat$survived <- refactorCategorical(dat$survived, c(0, 1, NA), c(1, 2, "?")
dat$pclass <- refactorCategorical(dat$pclass, c("1st", "2nd", "3rd"), c(1, 2, 3))

# replace all NA by ?
dat[is.na(dat)] <- "?"

# create model
model <- list(
  pclass = "Multinomial",
  survived = "Multinomial",
  sex = "Multinomial",
  age = "Gaussian",
  sibsp = "Poisson",
  parch = "Poisson",
  fare = "Gaussian",
  embarked = "Multinomial"
)

# create algo
algo <- createAlgo()

# run clustering
resLearn <- mixtCompLearn(dat, model, algo, nClass = 2:15, criterion = "ICL", nRun = 3, nCore = 1)
summary(resLearn)
plot(resLearn)

## Use standard data.frame and R format because titanic contains only standard variables.
# mixtCompLearn in "basic" mode without model parameters and data as a data.frame.
# A Multinomial model is used for factor variables, a Poisson for integer
# and a Gaussian for numeric.
resLearn <- mixtCompLearn(titanic, nClass = 2:15, nRun = 3, nCore = 1)

# imputed model
getType(resLearn)
Index

* data
  CanadianWeather, 4
  prostate, 17
  simData, 19
  titanic, 22
* package
  RMixtComp-package, 2
* plot
  plot.MixtCompLearn, 13
  plotCrit, 14
availableModels, 2, 3, 8
CanadianWeather, 3, 4, 18, 19, 22
computeDiscrimPowerClass, 3
computeDiscrimPowerVar, 3
computeSimilarityClass, 3
computeSimilarityVar, 3
createAlgo, 2, 7, 8
extractMixtCompObject, 5
getBIC, 2
getCompletedData, 2
getEmpiricTik, 2
getICL, 2
getModel, 2
getParam, 2
getPartition, 2
getProportion, 2
getTik, 2
getType, 2
getVarNames, 2
heatmapClass, 3
heatmapTikSorted, 3
heatmapVar, 3, 9
histMisclassif, 3
mixtCompLearn, 2, 3, 6, 6, 14, 15, 17, 19, 21
mixtCompPredict (mixtCompLearn), 6
plot.MixtComp, 3
plot.MixtCompLearn, 2, 3, 13, 14
plotConvergence, 3
plotCrit, 3, 14, 14
plotDataBoxplot, 3, 13
plotDataCI, 3, 13
plotDiscrimClass, 3
plotDiscrimVar, 3
plotProportion, 3
predict.MixtComp, 15
print.MixtCompLearn, 2, 16
prostate, 3, 4, 17, 19, 22
RMixtComp-package, 2
rmultinom, 10
rnbinom, 10
rnorm, 10
rpois, 10
rweibull, 10
simData, 3, 4, 18, 19, 22
slopeHeuristic, 19
summary.MixtCompLearn, 2, 21
titanic, 3, 4, 18, 19, 22