Mixmod is a software for modelling quantitative/qualitative data written in C++ (www.mixmod.org).

Mixmod provides a bridge between the C++ core library and the R statistical computing environment. The Rmixmod package implements the BIC, Package implementing CV.

Many algorithms can be chained to obtain original fitting strategies (e.g., CEM then EM with results of CEM).

Many initialization strategies combining these algorithms are possible.

Cluster analysis is performed with the mixmodCluster(data, nbCluster) function.

Estimation and Selection

- Estimation of the mixture parameters is considered through Maximum Likelihood via the following algorithms:
  - the EM (Expectation Maximization)
  - the SEM (Stochastic EM)
  - the CEM (Chained EM)

- Algorithms can be chained to obtain original fitting strategies (e.g., CEM then EM with results of CEM).

Visualization

- The models and the number of clusters can be chosen by different criteria:
  - BIC (Bayesian Information Criteria)
  - ICL (Integrated Completed Likelihood, a classification version of BIC)
  - NEC (Entropy Criterion)
  - CV (Cross Validation)

Mixture Models

Mixture probability-density function (pdf): $f(x|\theta) = \sum_{k=1}^{K} \pi_k f(x|\lambda_k)$

- $f(x|\lambda_k)$ denotes a $d$-dimensional distribution parameterized by $\lambda_k$.
- $\pi_k$ are the mixing proportions.
- $\lambda_k$ are the components of the distribution.

Multivariate Gaussian mixture models

In the qualitative case, $\theta$ is the density of a Gaussian distribution with mean $\mu_k$ and variance matrix $\Sigma_k$. $\theta$ is a $d$-dimensional distribution parameterized by $\lambda_k$. 26 Gaussian models based on the eigenvalue decomposition of the variance matrices are available. They depend on constraints on the variance matrix: same variance matrix between clusters, spherical variance matrix, etc.

- Gaussian models are computed with the mixmodGaussianModel() function.

Multivariate multinomial mixture models

In the qualitative case, $\theta$ is a multinominal distribution with a center $\mu_k$ and the dispersion $\Sigma_k$ around this center for the $k$th variable of the $d$th component.

- $\lambda_k = (\mu_k, \Sigma_k)$

10 multinominal models are available. $\lambda_k$ can be independent of the variable $j$ independent of the component $k$ and independent of both the variable $j$ and the component $k$.

- Multinominal models are computed with the mixmodMultinomialModel() function.

Discriminant Analysis

Labels $z$ are known.

- Estimate the group $z_{i+1}$ of any new individual $x_{i+1}$ of $\mathbb{R}^d$ with unknown label.

Discriminant analysis in Mixmod is divided into two steps:

1. Learning step

- Obtaining a classification rule from the training observations

- Learning is computed with the mixmodLearn(data, knownPartition) function.

Arguments

- data: a data matrix
- knownPartition: vector containing the known labels $z$

Return values

- bestModel: a S4 object containing results of the best model (estimated $\pi_k$, $\lambda_k$, partition, etc.)
- results: a list of S4 objects containing results of all models.

Example

- $\text{cluster analysis of iris with a list of cluster from 2 to 8 clusters}$
- all the Gaussian models, the BIC, ICL and NEC model selection criteria and an original strategy
- $\text{K}$ mixture models
- $\text{ICL}$, $\text{NEC}$, $\text{strategy} = \text{mixmodStrategy } (\text{name} = \{\text{"SEM", "EM"}, \text{strategy} = \text{mixmodStrategy } (\text{name} = \{\text{"SEM", "EM"}, \text{initMethod} = \text{"random"})}$

2. Prediction step

- Assigning remaining observations to one of the groups

- Prediction is computed with the mixmodPredict(data, classificationRule) function.

Arguments

- data: a data matrix
- classificationRule: vector containing the known labels $z$

Return values

- partition: a vector containing the predicted partition
- proba: a matrix containing probabilities of the prediction.

Example

- prediction of the 10 remaining observations with the classification rule obtained in the previous step: $\text{mixmodPredict(iris\_remaining\_part, learn\_bestModel)}$

Reference