Package ‘pGME’

August 29, 2013

Type Package

Title Estimates the parameters of normal mixtures by penalized likelihood methods (with exact Newton's method in multivariate case).

Version 1.0

Date 2012-08-10

Author Grigory Alexandrovich, Florian Schwaiger

Maintainer Grigory Alexandrovich <alexandrovich@mathematik.uni-marburg.de>

Description

Estimating the parameters of normal mixtures can lead to difficulties (especially for small sample sizes), if each component of the mixture has possibly a different mean and standard deviation resp. covariance matrix, since then the likelihood is unbounded for any standard deviation parameter going to zero. Further, when estimating a mixture with too many components the parameters are not identifiable anymore since weight parameters can converge towards zero. Both mentioned cases can be avoided by using penalty functions in the log-likelihood.

License GPL-2

Depends methods,mclust,mvtnorm,rgl,ellipse

R topics documented:

`pGME-package`	*Estimates the parameters of finite normal mixtures by penalized likelihood methods (with the exact Newton's method in the multivariate case).*

Description

Estimating the parameters of finite normal mixtures can lead to difficulties (especially for small sample sizes), if each component of the mixture has possibly a different mean and standard deviation resp. covariance matrix, since then the likelihood is unbounded for any standard deviation parameter going to zero. Further, when estimating a mixture with too many components the parameters are not identifiable anymore since weight parameters can converge towards zero. Both mentioned cases can be avoided by using penalty functions in the log-likelihood.
The package provides functions for penalized maximum likelihood estimation of finite normal mixtures. Due to penalization the algorithm avoids singularities (i.e. prevents any variance to converges towads zero, where the likelihood is unbounded). Further, the penalization can provide a better fit when components are not well separated.

Details

Package:	pGME
Type:	Package
Version:	1.0
Date:	2012-08-10
License:	GPL-2
Depends:	methods,mclust,mvtnorm,rgl,ellipse

Given a dataset one can use the function `gaussianMixtureMLE` to estimate the (penalized) MLE. The output is (among other values) an object of the class `normalMixtureModel-class`, which can for example be ploted with the generic plot function `plot-methods`. Further, `simulate-methods` simulates a dataset from a given mixture object. Finally, a maximum a posteriori clustering can be computed using the function `maxAposteriori`.

Author(s)

Grigory Alexandrovich, Florian Schwaiger
Maintainer: Grigory Alexandrovich <alexandrovich@mathematik.uni-marburg.de>

References

Alexandrovich, G., "An exact Newton's method for ML estimation in a penalized Gaussian mixture model". Preprint. (2012)
Chen, J. and Tan,X. "Inference for Multivariate Normal Mixtures", Journal of Multivariate Analysis. (2009)
Chen, J. and Li, P., "Hypothesis Test for Normal Mixture Models: The EM approach", The Annals of Statistics. (2009)
Fraley, C. and Raftery A. "MCLUST Version 3 for R: Normal Mixture Modeling and Model-Based Clustering". (2007)

Vollmer, S., Holzmann, H. and Schwaiger, F., "Peaks vs. Components." To appear in: Review of Development Economics. (2012)

See Also

gaussianMixtureMLE, logliDerivatives, plot-methods, plotComponents, simulate-methods, maxAposteriori

Examples

```
#one dimensional case
m1 <- createNormalMixtureModel(p = c(0.3,0.4), mu = c(1,3,3.5), sigma = c(0.8,0.8,0.8))
plot(m1)
x <- simulate(object = m1, nsim = 250)
gaussianMixtureMLE(x = x, k = 3, object = m1)$estimatedModel
gaussianMixtureMLE(x = x, k = 3, penSig = 1, penP = 1, object = m1)$estimatedModel

#2 dimensional case
sigma <- array(dim=c(2,2,2))
sigma[,,1] <- 0.2*c(1,0.8,0.8,1)
sigma[,,2] <- 0.4*c(1,-0.5,-0.5,1)
m2 <- createNormalMixtureModel(p = 0.7, mu = cbind(c(1,1),c(3,3)), sigma = sigma)
plot(m2)
x <- simulate(object = m2, nsim = 700)
estimate <- gaussianMixtureMLE(x = x, k = 2, penSig = 1)
```

---

convert_object *Converts an object of the class normalMixtureModel into a vector.*

Description

Converts an object of the class `normalMixtureModel` into a vector $(\mu_1, \ldots, \mu_k, L_1, \ldots, L_k, q_1, \ldots, q_{k-1})$, where $L_i {L_i}^T = \Sigma_i^{-1}$ and $p_i = \frac{q_i}{q_1 + \ldots + q_{k-1} + 1}$. The lower triagonal matrices L_i are vectrorized row wise (see mat2vec).

Usage

```
convert_object(object)
```

Arguments

`object`	An object of the class normalMixtureModel-class.

Value

A vector with length $k(D + D(D + 1)/2) + k - 1$

Examples

```
# define means and covariaces of the components of a two-dimensional two-component mixtur

# means
mu = cbind(c(1,1),c(3,3))

# covariances
sigma = array(dim=c(2,2,2))
sigma[,,1] = 0.2*c(1,0.8,0.8,1)
sigma[,,2] = 0.4*c(1,-0.5,-0.5,1)


# Create a normalMixtureModel object
object <- createNormalMixtureModel(p = 0.7, mu = mu, sigma = sigma)

# simulate 700 points from the mixture
x <- simulate(object = object, nsim = 700)

# estimate the parameters of the mixture from the simulated sample
estimate <- gaussianMixtureMLE(x = x, k = 2, penSig = 1)

# extract the parameters from the returned object and convert them into a vector
pars <- convert_object(estimate$estimatedModel)
```

---

createNormalMixtureModel
creates a mixture model object

Description

This function can be used to create an object of the type `normalMixtureModel-class` (using the `new`-function is certainly possible, too).

Usage

```
createNormalMixtureModel(p, mu, sigma)
```

Arguments

`p`	weights of the mixture components (it is possible to enter all k or only the first k-1 weights)
`mu`	means of the k components
`sigma`	standard deviations of the k components

Value

An object of the type `normalMixtureModel-class`.

Examples

```
m0 = createNormalMixtureModel(p=c(0.5),mu=c(1,2),sigma=c(0.8,0.8))
```

`gaussianMixtureMLE` *estimating (penalized) MLE*

Description

This function estimates the parameters of a (multivariate) k-component normal mixture using a penalized maximum likelihood estimator.

Usage

```
gaussianMixtureMLE(x, k, object, penP = 0, penSig = 0, doFI = FALSE, tol_eps = 1
 tol_delta = 1e-11, tol_grad = 1e-11, tol_rlc = 1e-08, tol_em = 1e-06, verbose =
 bcl = 35, maxit = 10, hard_conv = FALSE, parallel = TRUE)
```

Arguments

`x`	the dataset, which should be a vector in case of one dimensional fitting and a matrix in the multivariate case
`k`	number of mixture components
`object`	optionally an object of type `normalMixtureModel-class`, its parameters will be used as starting points for the optimization
`penP`	non-negative penalty constant for the weights (default is 0, i.e. no penalization)
`penSig`	non-negative penalty constant for the standard deviations or resp. covariance matrix (default is 0, i.e. no penalization)
`doFI`	If TRUE, an estimate of the Fisher information matrix of the MLE will be returned (only in the multidimensional case).
`tol_eps`	Tolerance for the solver of the linear equation systems. A number with absolute value less than `tol_eps` is considered as zero (only in the multidimensional case).
`tol_delta`	Tolerance for a stopping criterion. If the 2-norm of the Newton's direction is less then `tol_delta`, the function returns the current value θ_k (only in the multidimensional case).
`tol_grad`	Tolerance for a stopping criterion. If the 2-norm of the gradient of the log-likelihood is less then `tol_grad`, the function returns the current value θ_k (only in the multidimensional case).
`tol_rlc`	Tolerance for a stopping criterion. If the relative log-likelihood change is less then `tol_rlc`, the function returns the current value θ_k (only in the multidimensional case).
`tol_em`	Tolerance for the stopping criterion for the preceding EM algorithm from the package Mclust. If the relative log-likelihood change during the EM iterations is less than tol_em, than the current value θ_k is beeing passed to the Newton's iteration (only in the multidimensional case).
`verbose`	If TRUE, some additional outputs will be produced (only for multidimensional case). Default value is FALSE.
`bcl`	Backtracking length. Maximal number of iterations of the backtracking routine during the line search.
`maxit`	The maximal number of iterations for the Newton's method.

hard_conv	If TRUE, the algorithm iterates until all stopping criteria are fulfilled. Default value is FALSE.
parallel	If TRUE, some parts of the calculation are carried out in parallel using OpenMP interface.

Details

One dimensional case:

In detail the objective function is not only the log-likelihood, but the sum of the log-likelihood, a penalty function depending on the sigmas and a penalty function depending on the weights, i.e.

$$loglike(\mu_1,\dots,\mu_k,\sigma_1,\dots,\sigma_k,p_1,\dots,p_{k-1}|X_1,\dots,X_n)+c_s\cdot pen_1(\sigma_1,\dots,\sigma_k,x)+c_p\cdot pen_2(p_1,\dots,p_k),$$

where c_s and c_p are non-negative constants. These constants determine how strong small values of the parameters should be penalized and thus avoided. The penalty functions are given by

$$pen_1(\sigma_1,\dots,\sigma_k,x) = -\sum_{i=1}^{k}\frac{s_n^2}{\sigma_i^2}+\log(\frac{s_n^2}{\sigma_i^2})$$

where s_n^2 is the empirical variance, and

$$pen_2(p_1,\dots,p_k) = \sum_{i=1}^{k}\log(p_i).$$

Choosing $c_s = c_p = 0$ yields the MLE and is the default option. If no starting points are supplied, then they are calculated using package `mclust`. Thus, if no penalization is used, the used starting point is already the MLE and is only slightly changed.

Multidimensional case:

The function also estimates the parameter of a multivariate k-component normal mixture by maximizing the penalized log-likelihood function:

$$loglike(\mu_1,\dots,\mu_k,\Sigma_1,\dots,\Sigma_k,q_1,\dots,q_{k-1}|X_1,\dots,X_n)+c\cdot pen(\Sigma_1,\dots,\Sigma_k,x).$$

The penalty function is given by

$$pen(\Sigma_1,\dots,\Sigma_k,x) = -\sum_{i=1}^{k} tr(S_x\Sigma_i^{-1})+\log|\Sigma_i|$$

Where c is a non-negative constant. In contrary to the one dimensional case only the covariances can be penalized.

In the multidimensional case the optimization is carried out with the exact Newton's method. If no starting points are supplied, then they are calculated using k-means and the EM-Algorithm. Due to the fact that Newton's method converges locally it is better to supply no starting point rather than a bad starting point. The function uses internally the following parameterization

$$loglike(\mu_1,\dots,\mu_k,L_1,\dots,L_k,q_1,\dots,q_{k-1}|X_1,\dots,X_n)+c\cdot pen(L_1,\dots,L_k),$$

where $L_i{L_i}^T = \Sigma_i^{-1}$ and $p_i = \frac{q_i^2}{q_1^2+\dots+q_{k-1}^2+1}$.

The function uses analytical derivatives.

Value

In the one-dimensional case a list with 4 entries:

`estimatedModel`
: the estimated model, which is of the type `normalMixtureModel-class`

`loglik`
: A vector with two components. First component: value of the log-likelihood, second component: value of the penalized log-likelihood.

`AIC`
: value of the aic

`BIC`
: value of the bic

In the multidimensional case a list with 6 entries:

`estimatedModel`
: the estimated model, which is of the type `normalMixtureModel-class`

`BIC`
: value of the bic

`loglik`
: A vector with two components. First component: value of the log-likelihood, second component: value of the penalized log-likelihood.

`numit`
: A vector with two components. First component: number of EM-iterations to find a starting point, second component: number of Newton's iterations.

`convergence`
: A String. Describes which stopping rule took effect.

`MLE_covariance`
: An estimate of the covariance matrix of the MLE (- inverse of the Fisher Information), if demanded.

Author(s)

Grigory Alexandrovich, Florian Schwaiger

References

Chen, J. and Tan,X. "Inference for Multivariate Normal Mixtures", Journal of Multivariate Analysis. (2009)
Chen, J. and Li, P., "Hypothesis Test for Normal Mixture Models: The EM approach", The Annals of Statistics. (2009)
Grigory Alexandrovich. An exact Newton's method for ML estimation in a penalized Gaussian mixture model.

Examples

```
#one dimensional case
m1 <- createNormalMixtureModel(p = c(0.3,0.4), mu = c(1,3,3.5), sigma = c(0.8,0.8,0.8))
plot(m1)
x <- simulate(object = m1, nsim = 250)
gaussianMixtureMLE(x = x, k = 3, object = m1)$estimatedModel
gaussianMixtureMLE(x = x, k = 3, penSig = 1, penP = 1, object = m1)$estimatedModel

#2 dimensional case
sigma <- array(dim=c(2,2,2))
sigma[,,1] <- 0.2*c(1,0.8,0.8,1)
sigma[,,2] <- 0.4*c(1,-0.5,-0.5,1)
m2 <- createNormalMixtureModel(p = 0.7, mu = cbind(c(1,1),c(3,3)), sigma = sigma)
plot(m2)
```

```
x <- simulate(object = m2, nsim = 700)
estimate <- gaussianMixtureMLE(x = x, k = 2, penSig = 1)
```

`logliDerivatives` *Calculates the analytical derivatives of the penalized log-likelihood function in the multivariate case.*

Description

Calculates the analytical derivatives of the penalized log-likelihood

$$loglike(\mu_1, \ldots, \mu_k, L_1, \ldots, L_k, q_1, \ldots, q_{k-1} | X_1, \ldots, X_n) + c \cdot penalty(L_1, \ldots, L_k)$$

of a multivariate (dimension $D > 1$) normal mixture with respect to the parameter vector θ.

Usage

```
logliDerivatives(object = NULL, parameter = NULL, x, prop = 0,
 pen = 0, grad = TRUE, hess = TRUE, parallel =  TRUE)
```

Arguments

`object`	An object of the class `normalMixtureModel`. The derivatives are evaluated at the parameters stored in this object. If not supplied, the argument `parameter` must be supplied.
`parameter`	A $kD + kD(D+1)/2 + k - 1$ vector at which the derivatives are evaluated. k is the number of components and D is the dimension. If it is not supplied, `object` must be supplied. If both supplied, only `parameter` is used.
`x`	Data matrix. Each row must be a vector of length D.
`prop`	Internal parameter.
`pen`	Positive real number or zero. The weight of the penalization term.
`grad`	Logical. If TRUE the gradient of the penalized log likelihood will be calculated.
`hess`	Logical. If TRUE the hessian of the penalized log likelihood will be calculated.
`parallel`	If TRUE, some parts of the calculation are carried out in parallel using OpenMP interface.

Details

The parameter vector is given by

$$\theta = (\mu_1, \ldots, \mu_k, L_1^{\triangle}, \ldots, L_k^{\triangle}, q_1, \ldots, q_{k-1}),$$

where μ_i is a D-vector (mean of the component i), $L_i^{\triangle}$ is a $D(D+1)/2$ vector, it is the half-vectorization of the Cholesky factor of the inverse of the i'th covariance matrix: $L_i L_i^T = \Sigma_i^{-1}$ and $q_1, \ldots, q_{k-1}$ are the weight parameters. The weight of the i'th compoment is thereby given by $\frac{q_i^2}{q_1^2 + \ldots + q_{k-1}^2 + 1}$. The length of θ is $kD + kD(D+1)/2 + k - 1$.

Value

A list with 3 entries:

`loglikelihood`
: A number. The value of the penalized log likelihood at the supplied parameter.

`gradient`
: A vector. The gradient of the penalized log likelihood at the supplied parameter.

`hessian`
: A matrix. The Hessian at the supplied parameter.

References

Alexandrovich, G., "An exact Newton's method for ML estimation in a penalized Gaussian mixture model".

Examples

```
# define means and covariaces of the components of a two-dimensional two-component mixtur

# means
mu = cbind(c(1,1),c(3,3))

# covariances
sigma = array(dim=c(2,2,2))
sigma[,,1] = 0.2*c(1,0.8,0.8,1)
sigma[,,2] = 0.4*c(1,-0.5,-0.5,1)


# Create a normalMixtureModel object
object <- createNormalMixtureModel(p = 0.7, mu = mu, sigma = sigma)

# simulate 700 points from the mixture
x <- simulate(object = object, nsim = 700)

# estimate the parameters of the mixture from the simulated sample
estimate <- gaussianMixtureMLE(x = x, k = 2, penSig = 1)

# calculate the derivatives of the log-likelihood
devs <- logliDerivatives(object = estimate$estimatedModel, x = x)

# .. or alternative
# extract the parameters from the returned object and convert them into a vector
pars <- convert_object(estimate$estimatedModel)
devs_2 <- logliDerivatives(parameter = pars, x = x)
```

`mat2vec` *This function produces a row wise half-vectorization of a $D \times D$ matrix.*

Description

This function converts a $D \times D$ matrix into a vector. It takes only the diagonal and the elements under the diagonal.

Usage

```
mat2vec(mat)
```

Arguments

`mat`	A square matrix (typically a symmetric or a lower triangular).

Value

A vector with length $D(D+1)/2$, where the elements are concatenated row wise up to the diagonal.

Examples

```
#create a lower triangular matrix and convert it into a vector.
mat <- rbind(c(1,0),c(2,3))
vec <- mat2vec(mat)
```

`maxAposteriori` *maximum a posteriori estimates*

Description

Find the maximum a posteriori estimates for all data points given a normal mixture model.

Usage

```
maxAposteriori(x,object,detail = FALSE,plot = TRUE,levels = NULL)
```

Arguments

`x`	a vector resp. matrix containing the dataset
`object`	normal mixture model which should be used, see `normalMixtureModel-class` or `createNormalMixtureModel`, possibly an estimated model using `gaussianMixtureMLE`
`detail`	when detail equals TRUE also the maximum a posteriori probabilities are returned
`plot`	If TRUE and datadimension is 1 or 2 a plot will be produced.
`levels`	If plot = TRUE, optionally a vector with entries in (0,1). Then the contours of the according levels are ploted (see function ellipse from package ellipse).

Value

Depending on the input value of detail, either the a posteriori clustering or also the a posteriori probabilities.

Examples

```
#one dimensional case
m1 = createNormalMixtureModel(p = c(0.5), mu = c(1,3), sigma = c(0.8,0.8))
x1 = simulate(object = m1, nsim = 250)
fit1 = gaussianMixtureMLE(x = x1, k = 2, penSig = 1, penP = 1, object = m1)$estimatedMode
clust1 = maxAposteriori(object = fit1, x = x1, detail = FALSE, plot = TRUE)


#2 dimensional case
sigma <- array(dim=c(2,2,2))
sigma[,,1] <- 0.2*c(1,0.8,0.8,1)
sigma[,,2] <- 0.4*c(1,-0.5,-0.5,1)
m2 <- createNormalMixtureModel(p = 0.7, mu = cbind(c(1,1),c(3,3)), sigma = sigma)
x2 <- simulate(object = m2, nsim = 700)
fit2 = gaussianMixtureMLE(x = x2, k = 2, penSig = 1, object = m2)$estimatedModel
#first plot
maxAposteriori(object = fit2, x = x2, detail = FALSE, plot = TRUE)
#second plot
maxAposteriori(object = fit2, x = x2, detail = TRUE,levels=c(0.4,0.9))
```

`normalMixtureModel-class` *class for normal mixtures*

Description

This class formalizes normal mixture models.

Objects from the Class

Objects can be created by calls of the function `createNormalMixtureModel`, using `new()` or as a part of the returned value of `gaussianMixtureMLE`.

Slots

`p`: weights of the mixture components

`mu`: means of the k components

`sigma`: standard deviations or covariance matrices of the k components

`dimension`: dimension of the dataset

Methods

plot see `plot-methods`

simulate `simulate-methods`

Examples

```
#one dimensional case
m0 = createNormalMixtureModel(p=c(0.3,0.4),mu=c(1,3,3.5),sigma=c(0.8,0.8,0.8))
plot(m0)


#2 dimensional case
s0 = array(dim=c(2,2,2))
s0[,,1] = 0.2*c(1,0.8,0.8,1)
s0[,,2] = 0.4*c(1,-0.5,-0.5,1)
model2 = createNormalMixtureModel(p=0.7,mu=cbind(c(1,1),c(3,3)),sigma=s0)
plot(model2)
```

`plot-methods` *plot the density of a normal mixture*

Description

This generic function plots the density of a given normal mixture model (e.g. of an object of type `normalMixtureModel-class`).

Methods

`signature(x = "normalMixtureModel")`

See Also

`plotComponents`

Examples

```
#one dimensional case
m0 = createNormalMixtureModel(p=c(0.5),mu=c(1,2),sigma=c(0.8,0.8))
plot(m0)

#2 dimensional case
sigma <- array(dim=c(2,2,2))
sigma[,,1] <- 0.2*c(1,0.8,0.8,1)
sigma[,,2] <- 0.4*c(1,-0.5,-0.5,1)
m2 <- createNormalMixtureModel(p = 0.7, mu = cbind(c(1,1),c(3,3)), sigma = sigma)
plot(m2)
```

`plotComponents` *plot single components of a normal mixture*

Description

This function plots the weighted single components of a given normal mixture in one figure.

Usage

```
plotComponents(object, add = FALSE, main = "")
```

Arguments

object	normal mixture model which should be used, see normalMixtureModel-class or createNormalMixtureModel, possibly an estimated model using gaussianMixtureMLE
add	select TRUE, to add the plot to an existing plot
main	main title of the plot

See Also

plot-methods

Examples

```
m0 = createNormalMixtureModel(p=c(0.3,0.4),mu=c(1,3,3.5),sigma=c(0.8,0.8,0.8))
plotComponents(object=m0)
```

---

show-methods *output on the console of a mixture model*

Description

This function is only necessary to provide a nice output of a normal mixture model on the console.

Examples

```
m0 = createNormalMixtureModel(p=c(0.5),mu=c(1,2),sigma=c(0.8,0.8))
m0
```

---

simulate-methods *simulate data of a normal mixture*

Description

This generic function simulates a dataset of a given normal mixture model (e.g. of an object of type normalMixtureModel-class).

Methods

signature(object = "normalMixtureModel", nsim = "numeric")

Examples

```
#one dimensional case
m1 <- createNormalMixtureModel(p = c(0.3,0.4), mu = c(1,3,3.5), sigma = c(0.8,0.8,0.8))
x <- simulate(object = m1, nsim = 250)
plot(density(x))

#2 dimensional case
sigma <- array(dim = c(2,2,2))
sigma[,,1] <- 0.2*c(1,0.8,0.8,1)
sigma[,,2] <- 0.4*c(1,-0.5,-0.5,1)
m2 <- createNormalMixtureModel(p = 0.7, mu = cbind(c(1,1),c(3,3)), sigma = sigma)
x <- simulate(object = m2, nsim = 700)
plot(x, pch = 19, cex = 0.7)
```

Index