Documentation of the OpenTURNS-Mixmod module

Documentation built from package -

October 14, 2022

Abstract

The purpose of this document is to present the OpenTURNS-Mixmod module, which enables to build mixtures of multidimensional Normal distributions from a multidimensional sample. This construction is done with a control over the number of atoms in the mixture and over the covariance structure of these atoms. Some capabilities to perform classification are also provided.
This document is organised according to the Open TURNS documentation :

- a *Reference Guide which gives some theoretical basis on bayseian networks,*
- *a Use cases Guide which details scripts in python (the Textual Interface langage of Open TURNS) and helps the User to learn as quickly as possible the manipulation of the otmixmod module,*
- *the User Manual which details the otmixmod objects and give the list of their methods,*
- *the Examples Guide which provides at the moment only one example performed with the otmixmod module.*

Contents

1 Reference Guide

The Mixmod (MIXture MODelling) library and executable provide efficient algorithms for density estimation, clustering or discriminant analysis problems.

1.1 Mixtures

The probability density function of a mixture is a weighted sum of densities:

$$f(x) = \sum_i \alpha_i p_i(x) \qquad\qquad \sum_i \alpha_i = 1 \quad \text{with } 0 \leq \alpha_i \leq 1$$

1.2 References

MIXMOD estimates the mixture parameters through maximum likelihood via the EM (Expectation Maximization, Dempster et al. 1977), and the SEM (Stochastic EM, Celeux and Diebolt 1985) algorithm or through classification maximum likelihood via the CEM algorithm (Clustering EM , Celeux and Govaert 1992).

1 MIXMOD, www.mixmod.org/

2 Use Cases Guide

This section presents the main functionalities of the module *otmixmod* in their context.

2.1 Which python modules to import ?

In order to use the functionalities described in this documentation, it is necessary to import :

- the *openturns* python module which gives access to the Open TURNS functionalities,
- the *otmixmod* module which links the *openturns* functionalities and MIXMOD .

Python script for this use case :

```
# Load OpenTURNS to manipulate distributions
from openturns import *
# Load the link between OT and MIXMOD
from otmixmod import *
```

2.2 Creation of a Factory

In *otmixmod*, it is possible to create a factory for the mixture distribution. The parameters are:

- the number of atoms (mandatory argument),
- the covariance model (optional argument), $CovarianceModel$.

We will apply it to a $NumericalSample$ to get the mixture.

2.2.1 UC : Creation of a Covariance Model (optional)

A Covariance Model is the covariance model assumed for the several atoms of a mixture.

Requirements	• none
Results	• a CovarianceModel variable : *myCovarianceModel*, **type:** a CovarianceModel

Many covariance models are implemented in MIXMOD. The covariance models that are available are listed in tab 1 (cf. MIDMOD documentation for their meaning).

Python script for this use case:

```
# Create a CovarianceModel
myCovModel = Gaussian_pk_Lk_C()
```

Model	Categories
Gaussian_p_L_I	Spherical
Gaussian_p_Lk_I	
Gaussian_p_L_B	Diagonal
Gaussian_p_Lk_B	
Gaussian_p_L_Bk	
Gaussian_p_Lk_Bk	
Gaussian_p_L_C	General
Gaussian_p_Lk_C	
Gaussian_p_L_D_Ak_D	
Gaussian_p_Lk D_Ak_D	
Gaussian_p_L_Dk_A_Dk	
Gaussian_p_Lk_Dk_A_Dk	
Gaussian_p_L_Ck	
Gaussian_p_Lk_Ck	
Gaussian_pk_L_I	Spherical
Gaussian_pk_Lk_I	
Gaussian_pk_L_B	Diagonal
Gaussian_pk_Lk_B	
Gaussian_pk_L_Bk	
Gaussian_pk_Lk_Bk	
Gaussian_pk_L_C	General
Gaussian_pk_Lk_C	
Gaussian_pk_L_D_Ak_D	
Gaussian_pk_Lk D_Ak_D	
Gaussian_pk_L_Dk_A_Dk	
Gaussian_pk_Lk_Dk_A_Dk	
Gaussian_pk_L_Ck	
Gaussian_pk_Lk_Ck	

Table 1: The 28 Gaussian Models (Covariance Structure)

2.3 Creation of a Factory

The mixture factory is built from the number of atoms and the covariance model.

Requirements	• the atoms number : *atomsNumber*, **type:** UnsignedInteger • a CovarianceModel variable, optional: *myCovarianceModel* (default value $Gaussian\_pk\_Lk\_C$) **type:** CovarianceModel
Results	• a Mixture factory: *myMixtureFactory*, **type:** MixtureFactory

Python script for this use case:

```
atomsNumber = 3
factory = MixtureFactory(atomsNumber, myCovModel)
```

2.4 Estimation of the Mixture parameters

It is now possible to estimate the mixture parameters from a numerical sample.

Requirements	• a Mixture factory : *myMixtureFactory*, **type:** MixtureFactory • a numerical sample : *mySample*, **type:** NumericalSample
Results	• a Mixture distribution : *myEstimatedMixture*, **type:** Distribution

Python script for this use case:

```
# Estimate the mixture parameters
# We estimate all the parameters of the Mixture distribution from sample
estimatedDistribution = factory.build(sample)

# Display the resulted distribution with its parameters
print "Estimated distribution=", estimatedDistribution

# If the sample has a dimension 2, we draw the estimated pdf and the sample
if sample.getDimension() == 2 :
```

```
g = estimatedDistribution.drawPDF()
c = Cloud(sample)
c.setColor("red")
c.setPointStyle("bullet")
ctmp = g.getDrawable(0)
g.setDrawable(Drawable(c), 0)
g.addDrawable(ctmp)
g.draw("testMixtureEstimation")
```

The estimated distribution can be manipulated as a classical mixture distribution. So, in the case of a 1D- or 2D-sample it is possible to draw the estimated pdf and the sample on the same graph, cf. figure 1.

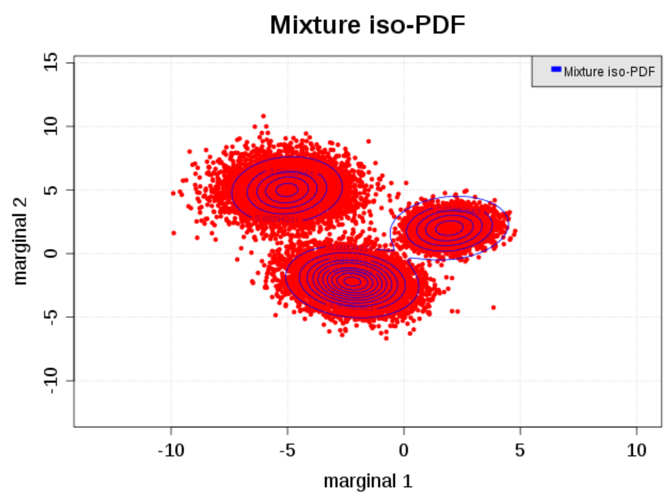

Figure 1: Estimated mixture and Numerical Sample

2.5 Estimation of the Mixture parameters and classification

It is now possible to estimate the mixture parameters from a numerical sample and to classify its points.

Requirements	• a Mixture factory : *myMixtureFactory*, **type:** MixtureFactory • a numerical sample : *mySample*, **type:** NumericalSample
Results	• a Mixture distribution : *myEstimatedMixture*, **type:** Distribution • an Indices : *labels*, **type:** Indices • a NumericalPoint : *BICLogLikelihood*, **type:** NumericalPoint

Python script for this use case:

```
# Estimate the mixture parameters
# We estimate all the parameters of the Mixture distribution from a sample,
# the labels of the points in the sample and the corresponding BIC Log Likelihood

bestLogLikelihood = -1e100
bestNbClusters = 0
bestLabels = Indices(0)
for nbClusters in range(1, 11):
    factory = MixtureFactory(nbClusters)
    labels = Indices(0)
    logLikelihood = NumericalPoint(0)
    estimatedDistribution = factory.build(sample, labels, logLikelihood)
    # Only the second component (i.e with index 1) is usefull for selection purpose
    if logLikelihood[1] > bestLogLikelihood:
        bestLogLikelihood = logLikelihood[1]
        bestNbClusters = nbClusters
        bestLabels = labels
print "best nb clusters=", bestNbClusters, "BIC log-likelihood=", bestLogLikelihood

# Classify the points
partition = list(NumericalSample(0, sample.getDimension()) for i in range(bestNbClusters
for i in range(sample.getSize()):
    partition[labels[i]].add(sample[i])
# Print the partition
for i in range(bestNbClusters):
    print "cluster", i, "=", partition[i]
```

The partitioning can be used to build local meta-models for example, in a mixture of experts approach.

3 User Manual

This section gives an exhaustive presentation of the objects and functions provided by the *otmixmod* module, in the alphabetic order.

3.1 CovarianceModel

3.1.1 CovarianceModelImplementation

Usage :

CovarianceModelImplementation ()

Arguments : none

Value : a CovarianceModelImplementation is the implementation of a CovarianceModel and is one to the following classes: Gaussian_p_L_I, Gaussian_p_Lk_I, Gaussian_p_L_B, Gaussian_p_Lk_B, Gaussian_p_L_Bk, Gaussian_p_Lk_Bk, Gaussian_p_L_C, Gaussian_p_Lk_C, Gaussian_p_L_D_Ak_D, Gaussian_p_Lk D_Ak_D, Gaussian_p_L_Dk_A_Dk, Gaussian_p_Lk_Dk_A_Dk, Gaussian_p_L_Ck, Gaussian_p_Lk_Ck, Gaussian_pk_L_I, Gaussian_pk_Lk_I, Gaussian_pk_L_B, Gaussian_pk_Lk_B, Gaussian_pk_L_Bk, Gaussian_pk_Lk_Bk, Gaussian_pk_L_C, Gaussian_pk_Lk_C, Gaussian_pk_L_D_Ak Gaussian_pk_Lk D_Ak_D, Gaussian_pk_L_Dk_A_Dk, Gaussian_pk_Lk_Dk_A_Dk, Gaussian_pk_L_Ck, Gaussian_pk_Lk_Ck.

Some methods :

convertToMixmod

Usage : $convertToMixmod()$

Arguments : none

Value : a String that is the mixmod name of the covariance model.

3.2 MixtureFactory

Usage :

$MixtureFactory(atomsNumber)$

$MixtureFactory(atomsNumber, covarianceModel)$

Arguments :

$atomsNumber$: a UnsignedInteger, the number of atoms to consider in the muxture.

$covarianceModel$: a CovarianceModel, the covariance model assumed for the several atoms of a mixture.

Some methods :

build

Usage : $build(sample)$

Usage : $build(sample, labels, BICLogLikelihood)$

Arguments :

$sample$: a NumericalSample of dimension $n \geq 1$

$labels$: an Indices that will be filled by the method.

BICLogLikelihood : a NumericalPoint that will be filled by 3 values: the log-likelihood of the estimated model, the corrected log-likelihood taking the number of parameters into account, and the entropy.

Value : a Mixture.

getAtomsNumber

Usage : *getAtomsNumber*()

Arguments : none

Value : an UnsignedInteger, which is the number of atoms in the mixture factory.

getCovarianceModel

Usage : *getCovarianceModel*()

Arguments : none

Value : a CovarianceModel, which is the covariance model assumed for the several atoms of a mixture.

setAtomsNumber

Usage : *setAtomsNumber*(*myAtomNumber*)

Arguments :

myAtomNumber : an UnsignedInteger, which is the number of atoms to set in the mixture factory.

Value : none

setCovarianceModel

Usage : *getCovarianceModel*(*myCovarianceModel*)

Arguments :

myCovarianceModel : a CovarianceModel, which is the covariance model assumed for the several atoms of a mixture.

Value : none

4 Examples Guide

This section presents some full-length examples of studies using the module.

4.1 Test case presentation

In this test case, we create a sample from a mixture and we try to estimate the mixture parameters from the sample. It is not a really an example of a study but it shows how to use this module.
The optimal number of clusters is not supposed to be known, and will be estimated as well.
We are in dimension 2, and the reference mixture is defined from 3 normal distributions:

$$f(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x)$$

with

- $f_1 = N(\mu_1, \sigma_1, R_1)$ with $\mu_1 = (-2.2, -2.2)$, $\sigma_1 = (1.2, 1.2)$, $R_1 = \begin{pmatrix} 1 & -0.2 \\ -0.2 & 1 \end{pmatrix}$ and $\alpha_1 = 0.5$,
- $f_2 = N(\mu_2, \sigma_2, R_2)$ with $\mu_2 = (2, 2)$, $\sigma_2 = (0.8, 0.8)$, $R_2 = \begin{pmatrix} 1 & 0.1 \\ 0.1 & 1 \end{pmatrix}$ and $\alpha_2 = 0.25$,
- $f_3 = N(\mu_3, \sigma_3, R_3)$ with $\mu_3 = (-5, 5)$, $\sigma_3 = (1.4, 1.4)$, $R_3 = \begin{pmatrix} 1 & 0. \\ 0. & 1 \end{pmatrix}$ and $\alpha_3 = 0.25$.

Then we print the parameters of the mixture (more precisely the distributions of the collection to build this mixture) and we can see that it is similar. We obtain the following distributions :

- $\hat{f}_1 = N(\mu_1, \sigma_1, R_1)$ with $\mu_1 = (-2.215, -2.197)$, $\sigma_1 = (1.168, 1.163)$, $R_1 = \begin{pmatrix} 1 & -0.137 \\ -0.137 & 1 \end{pmatrix}$ and $\alpha_1 = 0.498$,
- $\hat{f}_2 = N(\mu_2, \sigma_2, R_2)$ with $\mu_2 = (1.958, 2.009)$, $\sigma_2 = (1.175, 1.156)$, $R_2 = \begin{pmatrix} 1 & 0.136 \\ 0.136 & 1 \end{pmatrix}$ and $\alpha_2 = 0.255$,
- $\hat{f}_3 = N(\mu_3, \sigma_3, R_3)$ with $\mu_3 = (-5.020, 5.005)$, $\sigma_3 = (1.107, 1.221)$, $R_3 = \begin{pmatrix} 1 & 0.096 \\ 0.096 & 1 \end{pmatrix}$ and $\alpha_3 = 0.246$,

The drawing obtained in this example (with 2000 points) is on figure 1.

4.2 Python script

```python
# -*- coding: iso-8859-1 -*-
from openturns import *
from otmixmod import *

draw = True

##########################################################################################
# Create a multidimensional sample from a mixture of Normal
dim = 2
size = 20000
coll = DistributionCollection(0)
```

```
R = CorrelationMatrix(dim)


# First atom
for i in range(dim-1):
    R[i,i+1] = -0.2
mean = NumericalPoint(dim, -2.2)
sigma = NumericalPoint(dim, 1.2)
d = Distribution(Normal(mean, sigma, R))
coll.add(d)


# Second atom
R = CorrelationMatrix(dim)
for i in range(dim-1):
    R[i,i+1] = 0.1
mean = NumericalPoint(dim, 2.0)
sigma = NumericalPoint(dim, 0.8)
d = Distribution(Normal(mean, sigma, R))
coll.add(d)


# Third atom
mean = NumericalPoint( (-5.0, 5.0) )
sigma = NumericalPoint(dim, 1.4)
R = CorrelationMatrix(dim)
d = Distribution(Normal(mean, sigma, R))
coll.add(d)


coll[0].setWeight(0.5)
coll[1].setWeight(0.25)
coll[2].setWeight(0.25)


# Reference mixture
mixture = Mixture(coll)

# Creation of the numerical Sample from which we will estimate
# the parameters of the mixture.
sample = mixture.getNumericalSample(size)


##########################################################################
# Creation of the mixture factory
myAtomsNumber = 3
myCovModel = Gaussian_pk_L_Dk_A_Dk()
```

```python
bestLL = -1e100
bestMixture = Mixture()
bestNbClusters = 0
stop = False
nbClusters = 1
while not stop:
    factory = MixtureFactory(nbClusters, myCovModel)
    # Estimation of the parameters of the mixture
    labels = Indices(0)
    logLikelihood = NumericalPoint(0)
    estimatedDistribution = factory.build(sample, labels, logLikelihood)
    stop = logLikelihood[1] <= bestLL
    if not stop:
        bestLL = logLikelihood[1]
        bestNbClusters = nbClusters
        bestMixture = estimatedDistribution
    nbClusters += 1
print "best number of atoms=", bestNbClusters
myAtomsNumber = bestNbClusters
estimatedDistribution = bestMixture
# Some printings to show the result
print "Covariance Model used=",myCovModel.convertToMixmod()

print ""
print "Estimated distribution. Mixture composed of :"
for i in xrange(myAtomsNumber):
    d = estimatedDistribution.getDistributionCollection()[i]
    print i,"-  Mean = ",d.getMean()
    print i,"-  Sigma = ",d.getStandardDeviation()
    print i,"-  CorrelationMatrix = ",d.getCorrelation()
    print i,"-  Weight = ",d.getWeight()
    print ""

# Some drawings
if draw & (sample.getDimension() == 2):
g = estimatedDistribution.drawPDF()
c = Cloud(sample)
c.setColor("red")
c.setPointStyle("bullet")
ctmp = g.getDrawable(0)
g.setDrawable(Drawable(c), 0)
g.add(ctmp)
g.draw("testMixMod")
```