Finite Mixtures of Regression and Mixtures-of-Experts Models of Gaussian Data

Based on the MATLAB® package ***bayesf Version 2.0*** of S. Frühwirth-Schnatter

Autors: Kathrin Plankensteiner
Olivia Bluder

Contact: kathrin.plankensteiner@k-ai.at
olivia.bluder@k-ai.at

KAI – Kompetenzzentrum für Automobil- und Industrie-Elektronik GMBH
Europastr. 8
9524 Villach, AUSTRIA

Villach,
24.11.2011

Contents

1 Preface

Modeling data based on finite mixture distributions is a fast developing area with a lot of different applications. These models provide a straightforward method to derive a detailed representation and description for given data. Anyhow, inference is challenging.

This package is an extension of the MATLAB toolbox ***bayesf Version 2.0*** developed by S. Frühwirth-Schnatter. It was developed during a PhD and Master project at KAI-Kompetenzzentrum für Automobil-und Industrie Elektronik GmbH.

Major changes, constraints and extensions of ***bayesMoE Version 1.0*** compared to the package ***bayesf Version 2.0***, are the following:

- Supported by MATLAB 2011 and newer versions; might work on previous versions too
- Data must be a mixture of univariate normal distributions
- Priors for the regression coefficients are hierarchical or non-hierarchical normal distributions
- Priors for the variances are (hierarchical) inverse Gamma distributions
- Data can be censored
- Regression model can be
    - a finite (mixture) regression model or
    - a Mixtures-of-Experts model
- Cross validation (leave one out) can be performed
- Demos are available for:
    - Identifying the number of components (`demo_FindNumberOfComponents.m – censored data, demo_FindNumberOfComponts_uncensored.m – uncensored data`)
    - Fitting and predicting finite mixtures of regression models (`demo_mixreg.m – censored data, demo_mixreg_uncensored.m - uncensored data`)
    - Fitting and predicting Mixtures-of-Experts models (`demo_MoE.m – censored data, demo_MoE_uncensored.m – censored data`)

Attention:

- All modified functions from the original toolbox have been renamed to '*_MOE'.
- The code of each function gives you first information about the author and modifier.
- Demos that contain censored data take a lot of computational time!

Detailed information about the theoretical background, methods, function outputs and used data can be found here:

- S. Frühwirth-Schnatter: *Finite Mixture and Markov Switching Models*. Springer Series in Statistics (2006)

- Documentation ***bayesf Version 2.0:*** http://www.jku.at/ifas/content/e108280/e108502/e108556/e108675/book_matlab_version_2.0.pdf

- R. Kohavi: *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*. *in* 'IJCAI' , pp. 1137-1145 (1995)

- O.Bluder: *Statistical Modeling based on Physical Parameters of Smart Power Switches under Cycle Stress*. Dissertation, Alpen-Adria Universität Klagenfurt (2011)

- K.Plankensteiner*: Application of Bayesian Models to Predict Smart Power Switch Lifetime*, Diplomarbeit, Alpen-Adria Universität Klagenfurt (2011)

Please be aware that the program and functions are mainly tested for the data it was developed for. This package may still contain some coding errors and may not work for all sorts of data.

In case of errors or problems when using this package, please inform K. Plankensteiner or O. Bluder by writing an e-mail to

kathrin.plankensteiner@k-ai.at
olivia.bluder@k-ai.at

Finally we kindly ask to acknowledge the use of the packages ***bayesf 2.0*** and ***bayesMoE*** if you use results obtained by this package in any publication.

2 Fitting Finite Mixtures of Regression Models

To fit a finite mixture of regression models to data, five steps are necessary:

1. Preparing the data
2. Defining the model
3. Choosing the prior
4. Initializing the MCMC
5. Running the MCMC

2.1 Preparing the Data

`data` is a structure array with four obligatory fields:

- `data.y`: 1xN vector with observed data points
- `data.N`: size of data
- `data.X`: (d+1)xN matrix with a leading 1 and covariates
- `data.censor`: 1xN vector (δ) containing 0 and 1, such that:

$$\delta_i = \begin{cases} 0, & \text{if } y_i \text{ uncensored,} \\ 1, & \text{if } y_i \text{ censored.} \end{cases}$$

Mixtures-of-Experts (MoE) model

If the regression model is a Mixtures-of-Experts (MoE) model, additional fields are required:

- `data.Xweights`: contains either the empirical estimated mixture weights in a KxN matrix OR the values of covariates for the regression of mixture weights.
- `data.weightfun`: name of the function to model the mixture weights. This implies that the function to model the weights needs to be saved in an .m file with the name 'weightfun'.

ATTENTION: if only the intercept is required (=> fitting a finite mixture of distributions to the data) use a vector of ones instead of the matrix of covariates:

```
data.X = ones(1, data.N)
```

2.2 Defining the Model

A finite mixture model is a structure array, called `model`, with three obligatory fields to define:

- `model.K`: number of components
- `model.dist`: underlying density of the data within one mixture component
- `model.d`: number of regression coefficients for one component

Mixtures-of-Experts (MoE) model

If the regression model is a Mixtures-of-Experts (MoE) model, additional fields have to be defined:

- `model.MOE=true;`
- `model.weightfun=data.weightfun;`
- `model.indicmod.dist = 'FixedWeights';` (meaning that the weights have fixed values, instead of distributions. Hence, they can be modeled by a function.)

ATTENTION: the MoE_Toolbox can only handle:

- `model.dist= 'Normal'`

2.3 Choosing the Prior

Based on `data` and `model`, the following prior distributions can be applied:

- Normal distributions for the regression coefficients and hierarchical inverse Gamma distributions for the variances

```
prior=priordefine_MOE(data, model);
```

- Normal distributions for the regression coefficients and inverse Gamma distributions for the variances

```
prior.hier = false;
prior=priordefine_MOE(data, model, prior);
```

The information is stored in the structure array `prior`. If prior information is available, it can be included by writing directly into the corresponding arrays of `prior`.

2.4 Initializing the MCMC

Call now the function `mcmcstart_MOE()` to initialize the MCMC. This can be done by typing:

```
[data, model, mcmc]= mcmcstart_MOE(data, model)
```

This function defines the vector of initial allocations (`data.S`) as well as default values for the burn-in period (`mcmc.burnin`), the number of samples to draw (`mcmc.M`) and the number of allocations to store (`mcmc.storeS`). If the numbers of allocations to store is smaller than the number of samples, the last `mcmc.storeS` are stored. The default values are:

```
mcmc.M= 5000;
mcmc.burnin=1000;
mcmc.storeS=500;
mcmc.storepost=true;
mcmc.startpar=false;
mcmc.ranperm=true;
```

ATTENTION: Check if the vector of initial allocations `data.S` is sorted corresponding to the given prior information. This means that e.g. the first entries of the arrays in `prior` should be the corresponding prior information for data `data.y` with allocation 1 ($data.S_i = 1$). If this is not given, the allocations need to be permuted; else wrong prior information will applied.

2.5 Running the MCMC

Based on the data, the defined model, prior and MCMC properties (stored in `data, model, prior, mcmc`) Bayesian inference using MCMC is carried out when calling the function `mcmc_MOE()`:

```
mcmcout= mcmc_MOE(data, model, prior, mcmc);
```

2.6 Bayesian Inference Based on the MCMC Draws

Different ways to explore the Bayesian inference are provided:

- MCMC draws can be plotted with :
  `mcmcplot_MOE(mcmcout)`

- The value of the marginal likelihood can be determined with:
```
[marlik,mcmcout]=mcmcbf_MOE(data,mcmcout);
```

- Posterior allocation probabilities and classification can be plotted with:
```
[clust]=mcmcclust(data, mcmcout);
mcmcclustplot(data, clust);
```

- Parameter estimation can be done with
```
[est,mcmcout]=mcmcestimate_MOE(mcmcout);
```

- The fit can be plotted with (e.g. using the posterior mode estimates):
```
data.model=est.pm; %takes the posterior mode estimate
dataplot(data);
```

- Model discrimination can be evaluated by plotting PIT values:
```
PIT= PIT_mixtures(datared, mcmcout);
```

- Goodness of fit criteria like AIC, BIC etc. can be determined with
```
[ic,mcmcout]=mcmcic_MOE(data,mcmcout);
```

3 Predicting New Data

Based on the MCMC draws new data can be predicted. For this purpose the structure array `data_future` must include the following fields:

- `data_future.X`: (d+1) x M covariate matrix with leading 1, where M is the number of data to predict.

Mixtures-of-Experts (MoE) model

If the regression model is a Mixtures-of-Experts (MoE) model, additional fields are obligatory:

- `data_future.Xweights`: contains either the empirical estimated mixing weights in a K x M matrix OR the values of covariates for the regression of mixing weights.
- `data_future.weightfun`: name of the function to model the mixture weights

Call the function `mcmcpreddens_MOE` and data will be predicted based on the MCMC draws:
```
[fig,pred]=mcmcpreddens_MOE(data_future, mcmcout)
```

ATTENTION: if `data_future.y` is also available, the predicted data is compared with the real observation.

4 Cross Validation Evaluation

A defined model can be evaluated by using leave-one-out doing cross

To use this function, prepare the data (stored in structure array `data` - see section 1), define the model (stored in structure array `model` - see section 2) and choose the prior (stored in the structure array `prior` - see section 3).

If single results of the cross validation evaluation should be stored, define additionally:
`model.StoreResult=true;`

Mixtures-of-Experts (MoE) model

In case of a Mixtures-of-Experts (MoE) model, two cases are possible:

1) model development and prediction make use of the same `data.Xweights` and `data.weightfun`
2) model development and prediction make use of different `Xweights` and `weightfun` (e.g. `model development` with empirical estimated weights, prediction based on an approximation function/model for the mixing weights). In this case:
   - `model.MOE_pred=true;`

   Now define:
   - `model.MOE_Xweights`; Xweights to evaluate the weights based on model.MOE_weightfun
   - `model.MOE_weightfun`: name of the function to model the mixing weights, based on `model.MOE_Xweights`.

Call the function `CrossValidation_MOE()` to perform cross validation evaluation:
```
CrossVal=CrossValidation_MOE(data, 'mcmc_MOE', model, prior);
```

Due to visual reasons, only if the data contains less than 21 subsets with equal covariates, the results of cross validation are plotted automatically.

ATTENTION:

- Make sure that the columns of `data.y` and `data.X` are ordered, such that data points with same covariates are next to each other.
- Cross validation makes only sense if dataset consists of subsets in matrix X. (e.g. compare data.X in `datareg_MOE.m`)