MixEst: An Estimation Toolbox for Mixture Models

Reshad Hosseini[1] **and Mohamadreza Mash'al**[1]

1 School of ECE, College of Engineering, University of Tehran

DOI: 10.21105/joss.00524

Software

- Review
- Repository
- Archive

Submitted: 15 December 2017
Published: 07 January 2018

Summary

Mixture models are powerful statistical models used in many applications ranging from density estimation to clustering and classification. When dealing with mixture models, there are many issues that the experimenter should be aware of and needs to solve. The MixEst toolbox is a powerful and user-friendly package for MATLAB that implements several state-of-the-art approaches to address these problems. Additionally, MixEst gives the possibility of using manifold optimization for fitting the density model, a feature specific to this toolbox. MixEst simplifies using and integration of mixture models in statistical models and applications. For developing mixture models of new densities, the user just needs to provide a few functions for that statistical distribution and the toolbox takes care of all the issues regarding mixture models.

Introduction

Mixture models are an integrated and fundamental component in many machine learning problems ranging from clustering to regression and classification (McLachlan and Peel 2000). Estimating the parameters of mixture models is a challenging task due to the need to solve the following issues in mixture modeling:

- Unboundedness of the likelihood: This problem occurs when one component gets a small number of data points and its likelihood becomes infinite (Ciuperca, Ridolfi, and Idier 2003).
- Local maxima: The log-likelihood objective function for estimating the parameters of mixture models is non-concave and has many local maxima (Ueda et al. 2000).
- Correct number of components: In many applications, it is needed to find the correct number of components (Khalili and Chen 2007).

Addressing these issues for a mixture density when it is not available in common mixture modeling toolboxes will cost a lot of time and effort for the experimenter. MixEst addresses all these issues not only for already implemented densities, but also for densities that the user may implement. By implementing densities, we mean implementing a few simple functions which will be briefly discussed in Model Development Section.

This toolbox provides a framework for applying manifold optimization for estimating the parameters of mixture models. This is an important feature of this toolbox, because recent empirical evidence shows that manifold optimization can surpass expectation maximization in the case of mixtures of Gaussians (Hosseini and Sra 2015). It also opens the door for large-scale optimization by using stochastic optimization methods. Stochastic optimization also allows solving the likelihood unboundedness problem mentioned above, without the need of implementing a penalizing function for the parameters of the density.

While several libraries are available for working with mixture models, to the best of our knowledge, none of them offers a modular and flexible framework that allows for fine-tuning the model structure or can provide universal algorithms for estimating model parameters solving all the problems listed above. A review of features available in some libraries can be seen in Comparison Section.

In the next section, we give a short overview of the toolbox and its features.

About the MixEst Toolbox

This toolbox offers methods for constructing and estimating mixtures for joint density and conditional density modeling, therefore it is applicable to a wide variety of applications like clustering, regression and classification through probabilistic model-based approach. Each distribution in this toolbox is a structure containing a manifold structure representing parameter space of the distribution along with several function handles implementing density-specific functions like log-likelihood, sampling, etc. Distribution structures are constructed by calling factory functions with some appropriate input arguments defining the distribution. For example for constructing a mixture of one-dimensional Gaussians with 2 components, it will suffice to write the following commands in MATLAB:

```
Dmvn = mvnfactory(1);
Dmix = mixturefactory(Dmvn, 2);
```

As an example of how to evoke a function handle, consider generating 1000 samples from the previously defined mixture:

```
theta.D{1}.mu = 0; theta.D{1}.sigma = 1; % mean and variance of the 1st component
theta.D{2}.mu = 5; theta.D{2}.sigma = 2; % mean and variance of the 2nd component
theta.p = [0.8 0.2]; % weighting coefficients of components
data = Dmix.sample(theta, 1000);
```

Each distribution structure exposes a common interface that optimization algorithms in the toolbox can use to estimate its parameters. In addition to the EM algorithm which is a commonly implemented method in available libraries, our toolbox also makes optimization on manifolds available featuring procedures like early-stopping and mini-batching to avoid overfitting. For optimization on manifolds, our toolbox depends on optimization procedures of an excellent toolbox called Manopt (Boumal et al. 2014). In addition to optimization algorithms of Manopt like steepest descent, conjugate gradient and trust regions methods, the user can also use our implementation of Riemmanian LBFGS method.

Model Development

MixEst includes many joint and conditional distributions to model data ranging from continuous to discrete and also directional. Some users, however, may want to apply the tools developed in this toolbox for mixtures of a distribution not available in the toolbox yet. To this end, the user needs to write a factory function that constructs a structure for the new distribution.

Each distribution structure has a field named "M" determining the manifold of its parameter space. For example for the case of multivariate Gaussian distribution, this is a product manifold of a positive definite manifold and a Euclidean manifold:

```
% datadim is the function input argument determining the dimensionality of data
muM = euclideanfactory(datadim);
sigmaM = spdfactory(datadim);
D.M = productmanifold(struct('mu', muM, 'sigma', sigmaM));
```

The manifold of parameter space completely determines how parameter structure is given to or is returned by different functions. The structure of parameters for multivariate Gaussian would have two fields, a mean vector "mu" and a covariance matrix"sigma".

To use the estimation tools of the toolbox, two main functions have to be implemented. The *weighted log-likelihood* (wll) function and a function for computing the gradient of sum-wll with respect to the distribution parameters. The syntax for calling the wll function is:

```
llvec = D.llvec(theta, data);
```

The input argument `theta` is a structure containing the input parameters of the corresponding distribution. The second input argument `data` can be either a data matrix or a structure having several fields such as the data matrix and weights, which is interpreted using the `mxe_readdata` function. The output argument `llvec` is a vector with entries equal to wll for each datum (each column) in the data matrix.

The function to compute the gradient of sum-wll has the following syntax:

```
llgrad = D.llgrad(theta, data);
```

The input arguments are similar to the function `llvec`. The output argument `llgrad` is a structure similar to the input argument `theta` returning the gradient of sum-wll with respect to each parameter.

Some other (optional) functions that can be implemented for distributions are:

- `init`: This is for initializing the estimator using the data.
- `estimatedefault`: If the maximum wll has a structure that allows fast optimization (or has a closed-form solution), this estimator can be implemented in this function. When this function is not present, the Riemmanian optimization is called in the maximization step of EM algorithm.
- `llgraddata`: This function computes the gradient of wll with respect to the data. It is required in some special cases such as when the distribution is used as the radial component of an elliptically-contoured distribution or as the components in independent component analysis.
- `ll`: This function is sum-wll (sum of the output vector of `llvec` function). Sometimes it is faster to write this function differently than just calling `llvec` and summing up its output vector.

Two other functions that can be used in the split-and-merge algorithms to avoid local maxima of mixture models are `kl` (for computing KL-divergence) and `entropy` (for computing entropy). If the user wants to evoke a maximum-a-posteriori estimate, the functions `penalizerparam`, `penalizercost` and `penalizergrad` need to be implemented.

Feature Comparison

To demonstrate the richness of features in MixEst, we are comparing its features with several other well-known packages in The following table. Among many toolboxes available for mixture modeling, we select those that are feature-rich and representative. These packages are Sklearn (Pedregosa et al. 2011), Mclust (Fraley and Raftery 1999), FlexMix (Leisch 2004), Bayes Net (Murphy 2001) and MixMod (Biernacki et al. 2006). We include Bayes Net to demonstrate what a generic Bayesian graphical modeling toolbox can do. Sklearn is a powerful machine learning toolbox containing many tools, among others tools specific for mixture modeling. MixMod also provides bindings for Scilab and Matlab.

	MixEst	SKlearn	Mclust	FlexMix	Bayes Net	MixM
Programming language	Matlab	Python	R	R	Matlab	C++
Approaches for solving local minima	SM	IDMM	HC	—	—	—
Manifold optimization	Yes	No	No	No	No	No
Bayesian approaches for inference	MAP	VB	MAP	—	MAP	SM
Large-scale optimization	MB	—	—	—	—	SEM
Having tools for model selection	Yes	No	Yes	No	No	Yes
Automatic model selection	CSM	IDMM	—	—	—	—
Ease of extensibility	Easy	—	—	Easy	Medium	—
Having mixtures of experts	Yes	No	No	No	Yes	No
Having mixtures of classifiers	Yes	No	No	No	Yes	No
Having mixtures of regressors	Yes	No	No	Yes	Yes	No

[Table: SM stands for split-and-merge approach, IDMM stands for infinite dirichlet mixture models, HC stands for initialization using hierarchical clustering; MAP stands for maximum-a-posteriori, VB stands for variational Bayes; SEM stands for stochastic EM, MB stands for mini-batching; CSM stands for competitive split-and-merge]

References

Biernacki, Christophe, Gilles Celeux, Gérard Govaert, and Florent Langrognet. 2006. "Model-Based Cluster and Discriminant Analysis with the Mixmod Software." *Computational Statistics and Data Analysis* 51 (2). Elsevier:587–600.

Boumal, Nicolas, Bamdev Mishra, P.-A. Absil, and Rodolphe Sepulchre. 2014. "Manopt, a Matlab Toolbox for Optimization on Manifolds." *Journal of Machine Learning Research* 15:1455–9.

Ciuperca, Gabriela, Andrea Ridolfi, and Jérôme Idier. 2003. "Penalized Maximum Likelihood Estimator for Normal Mixtures." *Scandinavian Journal of Statistics* 30 (1):45–59.

Fraley, Chris, and Adrian E Raftery. 1999. "MCLUST: Software for Model-Based Cluster Analysis." *Journal of Classification* 16 (2). Springer:297–306.

Hosseini, Reshad, and Suvrit Sra. 2015. "Manifold Optimization for Gaussian Mixture Models." *arXiv Preprint arXiv:1506.07677*, June.

Khalili, Abbas, and Jiahua Chen. 2007. "Variable Selection in Finite Mixture of Regression Models." *Journal of the American Statistical Association* 102 (479):1025–38.

Leisch, Friedrich. 2004. "FlexMix: A General Framework for Finite Mixture Models and Latent Class Regression in R." *Journal of Statistical Software* 11 (8):1–18.

McLachlan, Geoffrey, and David Peel. 2000. *Finite Mixture Models*. New Jersey: John Wiley; Sons.

Murphy, Kevin P. 2001. "The Bayes Net Toolbox for Matlab." *Computing Science and Statistics* 33:2001.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12:2825–30.

Ueda, Naonori, Ryohei Nakano, Zoubin Ghahramani, and Geoffrey E. Hinton. 2000. "Split and Merge EM Algorithm for Improving Gaussian Mixture Density Estimates." *The Journal of VLSI Signal Processing* 26 (1):133–40.