em: A Generic Function of the EM Algorithm for Finite Mixture Models in R

Dongjie Wu
Københavns Universitet

Abstract

The *em* package estimates finite mixture models using the expectation-maximization (EM) algorithm. It follows R's feature of generic functions and the function `em()` can be applied after a model fitting with one component using R's pre-existing functions and packages.

Currently, it supports the following models: linear models (`lm()`), generalized linear models (`glm()`), generalized non-linear model (`gnm()`), survival models (mainly conditional logistic regression) (`survival::clogit()`), multinomial models(`nnet::multinom()`).

Keywords: R, the EM Algorithm, finite mixture models, model based clustering, latent class regression.

1. Introduction

Finite mixture models (FMMs) are widely applied in both natural science and social science to model unobserved heterogeneity[1]. This modelling technique assumes that data can be divided into unobserved groups known as latent classes, each following a distinct probability density or a regression model with unique model parameters.

One reason for the extensive usage of finite mixture modelling is its flexibility. Its core assumption of unobserved heterogeneity can be applied to a variety of models including generalized linear regression models (GLMs), generalized non-linear regression models (GNMs), survival analysis, etc. It can also be used as a model-based clustering technique with different data structures, for example, categorical data, multidimensional data or hierarchical data (Vermunt 2008). In addition, FMMs can parameterize probabilities of belonging to latent classes (i.e. a concomitant model (Wedel 2002)). FMMs can also be applied to data with a nested or hierarchical structure (Vermunt and Magidson 2005).

In general, FMMs can be estimated by the following four methods: Methods of moments (MoM), Maximum log-likelihood estimation (MLE), Bayesian methods, and unsupervised machine learning methods (Figueiredo and Jain 2002). MLE using the *Expectation-maximization* (EM) algorithm is the mainstream method of estimating FMM models due to its performance and accuracy. However, it is relatively computationally intensive as it involves iteratively computing log-likelihood values.

There are many packages or software available for FMM models: e.g. *fmm* (Deb 2007) and *gllamm* (Rabe-Hesketh, Skrondal, and Pickles 2004) in Stata, *flexmix* (Leisch 2004), *mixtools*

[1]See McLachlan, Lee, and Rathnayake (2019) for an introduction to finite mixture models

(Benaglia, Chauveau, Hunter, and Young 2010) and *mclust* (Scrucca, Fop, Murphy, and Raftery 2016) in R, etc. Although these packages provide a wide range of opportunities for fitting finite mixture models, they do not cover all models, including some of those that are used my area of research. The em package thus fills this gap. To ensure user-friendliness, the em package is based on generic functions in R, which integrates better with other functions and packages in R and makes implementing FMM models more flexible and straightforward.

In section 2, we define FMMs and some of their extensions. In section 3, we present EM algorithms and the approaches we use to fit FMMs. In section 4, we introduce the *em* package and the generic *em* function for fitting FMMs in R. Section 5 demonstrates some examples of using *em*. Section 6 provides a short summary of the paper.

2. Finite Mixture Models

2.1. The Finite Mixture Models

Finite mixture models can be described in the following equations given J components:

$$f(\mathbf{y}|\mathbf{x},\phi) = \sum_{j=1}^{J} \pi_j (f_j(\mathbf{y}|\mathbf{x},\theta_j)) \tag{1}$$

where $\sum_{j=1}^{J} \pi_j = 1$, $\mathbf{y}$ is a dependent variable with conditional density f, $\mathbf{x}$ is a vector of independent variables, π_j is the prior probability of component j, θ_j is the component specific parameters for the density function f_j.

The model $f_j(\mathbf{y}|\mathbf{x},\theta_j)$ can be one of a wide range of models: probability distributions, generalized linear models (GLM), generalized non-linear models (GNM), survival models, categorical models, etc.

Given the fitted model, one can compute the posterior probability that observation i belongs to class j using the following equation:

$$Pr(j|\mathbf{x},\phi,\mathbf{y}) = \pi_k \frac{f_k(\mathbf{y}|\mathbf{x},\theta_j)}{\sum_{j=1}^{J} \pi_j (f_j(\mathbf{y}|\mathbf{x},\theta_j))} \tag{2}$$

The prior probability can also be parameterized by a vector of variables $\mathbf{z}$.

$$f(\mathbf{y}|\mathbf{x},\phi) = \sum_{j=1}^{J} \pi(\mathbf{z}) (f_j(\mathbf{y}|\mathbf{x},\theta_j)) \tag{3}$$

where $\pi(\mathbf{z})$ is a function that determines the prior probability by a vector of variables $\mathbf{z}$. The equation 3 is known as a concomitant model.

2.2. Hierarchical Mixture Models

3. Fitting FMM using the EM Algorithm and its Extensions

The EM algorithm is the mainstream approach to fitting finite mixture models. It runs iteratively through an expectation step (E-step) and a maximization step (M-step). Given an

FMM, the E-step computes posterior probabilities given the estimation in the last step, while the M-step estimates the model by maximizing the log-likelihood function. The log-likelihood function generated by an FMM can be rewritten:

$$L_I(\theta|\mathbf{y},\mathbf{x}) = \sum_{i\in N} \log \sum_{j\in J} \pi_j f_j(y_i|x_i,\theta_j) \tag{4}$$

An FMM can be treated as a model with missing data represented by latent classes. Therefore, we can assume a variable w that captures which class each observation belongs to. In this case, the data $\{\mathbf{y}, w\}$ becomes complete data. Based on this complete data, we can write the following complete-data log-likelihood function.

$$L_C(\theta|\mathbf{y},\mathbf{x},w) = \sum_{i\in N} \log \sum_{j\in J} w_i \pi_j f_j(y_i|x_i,\theta_j) \tag{5}$$

Since the EM is computationally intensive due to its iterative nature, we provide two extensions of the EM algorithm in the *em* package. One is the Categorical EM (CEM) and the other is the Stochastic EM (SEM). The CEM assigns each observation to a class for which the posterior probability is higher, whereas SEM randomly assigns each observation to a certain class based on the observation's posterior probabilities for each class (i.e., weighted random allocation with weights given by the posterior probabilities).

Using either CEM or SEM, we can assign each observation to a latent class after an E-step, and estimate the model respectively for each class. This approach requires no fitting for a mixed version of log-likelihood functions. Therefore, it could be more computational efficient when the model is easier to fit given one class.

However, this classification approach may not work well when a one-class model has a flat log-likelihood function such as a conditional logit model. In this case, it is difficult to identify variation by latent classes by simply assigning observations to each latent class. To fit this type of models, we adopt an EM algorithm with a complete-data log-likelihood. A complete-data log-likelihood increases steepness of the log-likelihood function so that the EM algorithm can better differentiate classes and search for the maximum.

To improve the performance of model-fitting, we make use of the *Rcpp* and *RcppArmadillo* packages to embed c++ code in R. The R programs make uses of the `optim` function in R to call the embedded c++ code. In an E-step, we also use the c++ code to compute posterior probabilities. The use of c++ massively reduces the computational time of estimating a complete-data log-likelihood function and, therefore, improves the computational efficiency.

For running the EM algorithm, we also provide several methods to generate starting values. Starting values are crucial for fitting a mixture model as the log-likelihood function of a mixture model usually has several local optimums. Choosing a starting value far from the global optimum can lead to a less optimal result. In the package, we created two arguments `init.method` and `optim.start` for methods generating starting values. `init.method` generates the starting clustering for each observation. The available methods are: `random` – random assignment to a cluster that each observation is allocated to, `kmeans` – k-mean clustering of data, and `hc` – model-based agglomerative hierarchical clustering. An m-step runs immediately after the clustering to provide the starting values for the model parameters.

If the complete log-likelihood function is used in the estimation, we use `optim.start` for generating the starting value of MLE. In the current version, it supports two methods: `random`

generates a set of random values and `sample5` generates five sets of random values and runs for each set an MLE with each set for five time. Values with the largest log-likelihood are retained as starting values of next round of MLE.

For simple models like the generalized linear regression models, we recommend using `kmeans` or `random` to generate the staring values. For more complex models like a conditional logit model, we recommend fitting a complete log-likelihood function with `sample5`.

4. Using the Generic em Function

As a generic function, the *em* function can take different models as its input. Currently, it supports the following models implemented in both the R base package and external packages: linear models (`lm()`), generalized linear models (`glm()`), generalized non-linear model (`gnm()`), survival models (mainly conditional logistic regression) (`survival::clogit()`), multinomial models(`nnet::multinom()`). The following function describes the default *em* function:

```
em(object, latent=2, verbose=F,
    init.method = c("random", "kmeans", "hc"),
    init.prob = NULL, algo= c("em", "cem", "sem"),
    cluster.by=NULL, max_iter=500,
    abs_tol=1e-4, concomitant=list(...),
    use.optim=F, optim.start=c("random","sample5"), ...)
```

where `object` is a model object such as `lm()` and `glm()`, `latent` determines the number of latent classes, `verbose` allows printing the fitting process, `init.method` provides three different initialization methods, `init.prob` allows users to determine the staring prior probabilities, `algo` provides three extensions of EM algorithms, `cluster.by` is a variable specifying the level at which the clustering should occur (for hierarchical data), `max_iter` determines the maximum number for iteration, `abs_tol` is the absolute tolerance – the difference between two log-likelihood values, `concomitant` allows users to specify a concomitant model, `use.optim` decides whether to use the function `optim()` to maximize log-likelihood functions, and `optim.start` determines the method to generate the starting values for `optim()`.[2]

5. Examples

5.1. Mixture of Generalized Linear Models

In the first example, we mix two linear models described in the following two equations with mixing probabilities 30% and 70% respectively:

$$y = -10 + 0.1x \tag{6}$$
$$y = 20 - 0.1x \tag{7}$$

The models can be fitted and presented using the following lines of codes in R.

[2]For more detail, please check the *em* documentations.

```
R> library("em")
R> fit.lm <- lm(yn~x, data=simreg)
R> summary(fit.lm)

Call:
lm(formula = yn ~ x, data = simreg)

Residuals:
    Min      1Q  Median      3Q     Max
-25.720 -18.166   7.212   9.926  17.206

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.73792    1.22736   8.749   <2e-16 ***
x           -0.04715    0.18710  -0.252    0.801
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.59 on 998 degrees of freedom
Multiple R-squared:  6.362e-05,        Adjusted R-squared:  -0.0009383
F-statistic: 0.0635 on 1 and 998 DF,  p-value: 0.8011

R> results <- em(fit.lm, latent=2, verbose=F)
R> summary(results)

Call:
em.default(object = fit.lm, latent = 2, verbose = F)

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
1.(Intercept) -9.88442    0.18569 -53.230   <2e-16 ***
1.x            0.07305    0.02819   2.591   0.0097 **
2.(Intercept) 19.74555    0.25442  77.611   <2e-16 ***
2.x           -0.05942    0.03886  -1.529   0.1265
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Prior Probabilities:
Comp.1: 0.31
Comp.2: 0.69


logLik: -2975, AIC: 5964, BIC: 5999.
```

In the code block showing above, we use `summary()` to print and summarize results after the use of `em()`. We organize the presentation of results in a similar look comparing with the summary of an baseline one-class model (like `lm()`). The difference is that we add a

prefix 1. or 2. to represent which class the coefficients belong to. In addition, we present prior probabilities below the table of coefficients to show the proportion of each class in the mixture model.

The summary of `em` fitting shows that the results are close to the true values. After model fitting, we can use other generic functions in R for post-estimations and results printing. The supporting generic functions in the current version are as follows: `summary()`, `print()`, `predict()`, `update()`, `logLik()`,`df()`, `plot()`.

```
R> fmm_fit <- predict(results)
R> str(fmm_fit)

List of 3
 $ components:List of 2
  ..$ : Named num [1:1000] -9.56 -9.59 -9.42 -9.71 -9.46 ...
  .. ..- attr(*, "names")= chr [1:1000] "1" "2" "3" "4" ...
  ..$ : Named num [1:1000] 19.5 19.5 19.4 19.6 19.4 ...
  .. ..- attr(*, "names")= chr [1:1000] "1" "2" "3" "4" ...
 $ mean      : num [1:1000, 1] 10.5 10.5 10.4 10.5 10.5 ...
 $ prob      : chr "prior"
 - attr(*, "class")= chr [1:3] "predict" "predict.em" "predict.prior"

R> fmm_fit_post <- predict(results, prob="posterior")
R> str(fmm_fit_post)

List of 3
 $ components:List of 2
  ..$ : Named num [1:1000] -9.56 -9.59 -9.42 -9.71 -9.46 ...
  .. ..- attr(*, "names")= chr [1:1000] "1" "2" "3" "4" ...
  ..$ : Named num [1:1000] 19.5 19.5 19.4 19.6 19.4 ...
  .. ..- attr(*, "names")= chr [1:1000] "1" "2" "3" "4" ...
 $ mean      : num [1:1000, 1:2] -3.84e-42 -8.02e-35 -8.03e-42 -1.02e-30 -5.12e-43 ...
 $ prob      : chr "posterior"
 - attr(*, "class")= chr [1:3] "predict" "predict.em" "predict.posterior"
```

In the above coding block, we use function `predict()` to predict values based on the results from the model fitting. In the default mode, `predict()` produces the fitted values for each component saved in the list `components` and computes the weighted mean based on the fitted values for each component and the prior probability π:

$$\hat{y} = \sum_{j \in J} \pi_j \hat{y_j} \tag{8}$$

The results for the weighted sum are saved in a vector `mean`.

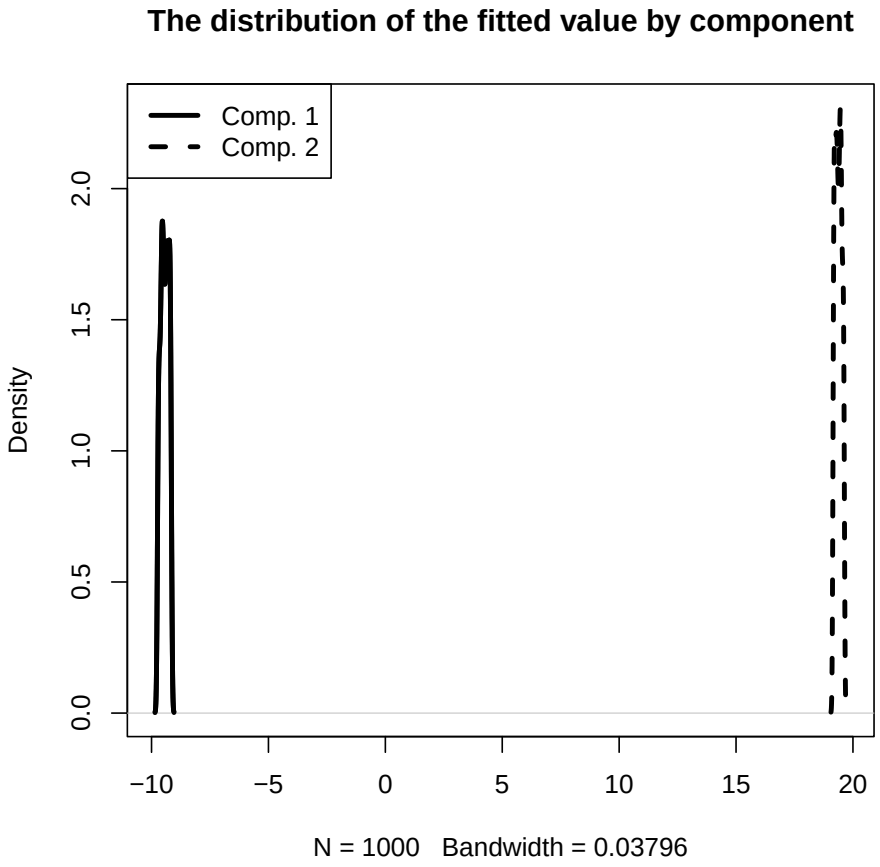
We can set the argument `prob="posterior"` to compute the fitted values weighted by the posterior probability using the following equation:

$$\hat{y} = \sum_{j \in J} \pi_{ij} \hat{y_j} \tag{9}$$

where π_{ij} is the posterior probability.

For data visualization, we use `plot()` to produce graphs after the model fitting `em()`. In the current version, `plot()` supports producing density plots by component and histograms of posterior probabilities. One can set up the figure using the argument `by`.

Figure 1: The counterfactual distribution of the predicted by component



In the default mode, `plot()` produces a graph of distributions of fitted values for each component (See Figure 1).

Figure 2 shows a histogram of posterior probability distributions.

We can also use a concomitant model to parameterize the prior probabilities. For example, we can assume the concomitant model in the following equations:

$$y_c = \pi_1(z)f_1(x) + \pi_2(z)f_2(x) \quad (10)$$

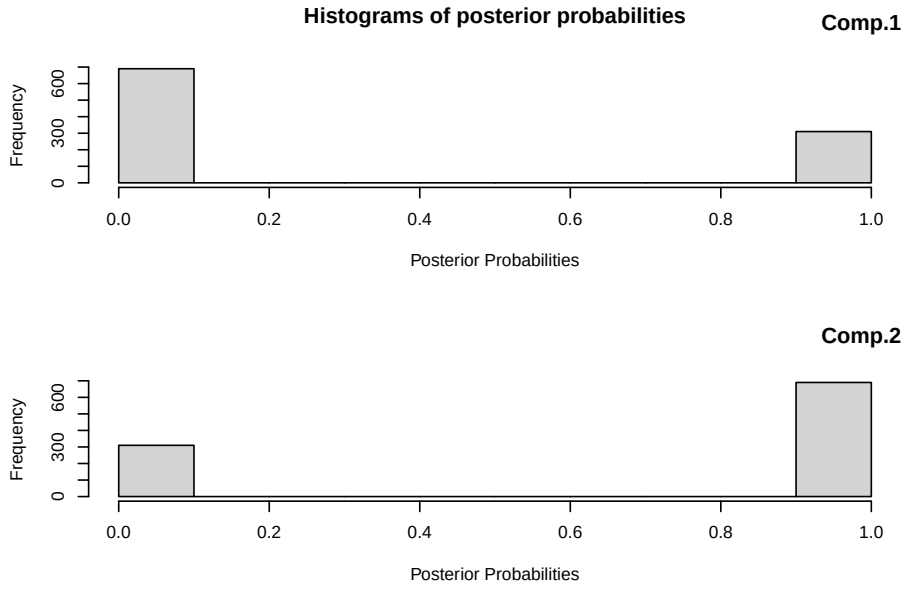
$$\pi_1 + \pi_2 = 1 \quad (11)$$

$$\pi(z) = g(z) \quad (12)$$

Given the equations, we run the following lines of codes:

```
R> formula <- yc ~ x
R> formula_c <- ~ z
```

Figure 2: Histograms of posterior probabilities



```
R> lm_fit <- lm(formula, data=simreg)
R> results <- em(lm_fit, concomitant=list(formula=formula_c, data=simreg))
R> summary(results)

Call:
em.default(object = lm_fit, concomitant = list(formula = formula_c,
    data = simreg))

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
1.(Intercept)  20.69606    0.26784  77.271  < 2e-16 ***
1.x            -0.23917    0.04072  -5.874 5.79e-09 ***
2.(Intercept) -10.13260    0.18380 -55.129  < 2e-16 ***
2.x             0.09922    0.02806   3.536 0.000424 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Prior Probabilities:
Comp.1: 0.339
Comp.2: 0.661

Concomitant model:
              Estimate Std. Error t value Pr(>|t|)
2.(Intercept)   0.1843     0.1280   1.440    0.150
2.z            -0.2978     0.2233  -1.334    0.183

logLik: -2948, AIC: 5912, BIC: 5951.
```

In the summary of results, we print the results from the concomitant model including the logistic regression of $\pi(z)$.

5.2. Mixture of Conditional Logistic Regression Models

In this example, a three-level outcome y is generated by a mixture of two multinomial regressions with a three-category variable x described in the following equations.
Class 1:

$$ln\frac{Pr(y_i = 2)}{Pr(y_i = 1)} = 2.1x_2 + 4.1x_3 \quad (13)$$

$$ln\frac{Pr(y_i = 3)}{Pr(y_i = 1)} = 8.1x_2 - 0.1x_3 \quad (14)$$

Class 2:

$$ln\frac{Pr(y_i = 2)}{Pr(y_i = 1)} = 20.3x_2 + 1.9x_3 \quad (15)$$

$$ln\frac{Pr(y_i = 3)}{Pr(y_i = 1)} = 10.5x_2 + 0.7x_3 \quad (16)$$

The probability of the mixture is 0.3 and 0.7 for two classes, respectively.

```
R> library("em")
R> library("survival")
R> fmla <- chosen2 ~ 0 + a2_x2 + a2_x3 + a3_x2 + a3_x3+ strata(id)
R> cfit <- clogit(fmla, simclogit)
R> emfit <- em(cfit, latent=2, verbose=F, init.method="kmeans", use.optim=T, optim.start="

Random start: 1
Iteration 1: (EM) log likelihood = 14323.5221
Iteration 2: (EM) log likelihood = 14033.9953
Iteration 3: (EM) log likelihood = 14004.4842
Iteration 4: (EM) log likelihood = 14001.2086
Iteration 5: (EM) log likelihood = 14000.8126
Random start: 2
Iteration 1: (EM) log likelihood = 14323.5221
Iteration 2: (EM) log likelihood = 13987.5831
Iteration 3: (EM) log likelihood = 13996.8977
Iteration 4: (EM) log likelihood = 14003.7911
Iteration 5: (EM) log likelihood = 14005.0099
Random start: 3
Iteration 1: (EM) log likelihood = 14323.5221
Iteration 2: (EM) log likelihood = 13814.0416
Iteration 3: (EM) log likelihood = 13767.1343
Iteration 4: (EM) log likelihood = 13761.0806
Iteration 5: (EM) log likelihood = 13760.0858
Random start: 4
Iteration 1: (EM) log likelihood = 14323.5221
Iteration 2: (EM) log likelihood = 14198.8056
Iteration 3: (EM) log likelihood = 14244.7413
Iteration 4: (EM) log likelihood = 14249.8756
Iteration 5: (EM) log likelihood = 14250.3362
Random start: 5
Iteration 1: (EM) log likelihood = 14323.5221
Iteration 2: (EM) log likelihood = 14474.4036
Iteration 3: (EM) log likelihood = 14471.6073
Iteration 4: (EM) log likelihood = 14471.5818
Iteration 5: (EM) log likelihood = 14471.6738
Iteration 0: (EM) log likelihood = 13759.9111
Iteration 1: (EM) log likelihood = 13759.5697
Iteration 2: (EM) log likelihood = 13759.5447
Iteration 3: (EM) log likelihood = 13759.5073
Iteration 4: (EM) log likelihood = 13759.5583
Iteration 5: (EM) log likelihood = 13759.5316
Iteration 6: (EM) log likelihood = 13759.529
Iteration 7: (EM) log likelihood = 13759.5186
Iteration 8: (EM) log likelihood = 13759.5082
```

```
Iteration 9: (EM) log likelihood = 13759.4992
Iteration 10: (EM) log likelihood = 13759.4988
Iteration 11: (EM) log likelihood = 13759.4959
Iteration 12: (EM) log likelihood = 13759.4972
Iteration 13: (EM) log likelihood = 13759.4948
Iteration 14: (EM) log likelihood = 13759.4951
Iteration 15: (EM) log likelihood = 13759.4945
Iteration 16: (EM) log likelihood = 13759.4948
Iteration 17: (EM) log likelihood = 13759.4942
Iteration 18: (EM) log likelihood = 13759.4946
Iteration 19: (EM) log likelihood = 13759.4951
Iteration 20: (EM) log likelihood = 13759.4959
Iteration 21: (EM) log likelihood = 13759.496
Iteration 22: (EM) log likelihood = 13759.4979
Iteration 23: (EM) log likelihood = 13759.4965
Iteration 24: (EM) log likelihood = 13759.4978
Iteration 25: (EM) log likelihood = 13759.498
Iteration 26: (EM) log likelihood = 13759.4987
Iteration 27: (EM) log likelihood = 13759.4987

R> print(summary(emfit))

Call:
NULL

Coefficients:
             coef exp(coef)  se(coef)      z Pr(>|z|)
1.a2_x2 7.261e+00 1.424e+03 1.000e+00  7.260 3.87e-13 ***
1.a2_x3 3.989e+00 5.398e+01 6.665e-02 59.846  < 2e-16 ***
1.a3_x2 6.753e+00 8.568e+02 1.001e+00  6.750 1.48e-11 ***
1.a3_x3 1.256e+00 3.513e+00 7.831e-02 16.044  < 2e-16 ***
2.a2_x2 1.084e+01 5.082e+04 1.000e+00 10.834  < 2e-16 ***
2.a2_x3 8.993e-01 2.458e+00 6.665e-02 13.494  < 2e-16 ***
2.a3_x2 8.904e+00 7.358e+03 1.001e+00  8.899  < 2e-16 ***
2.a3_x3 5.079e-01 1.662e+00 7.831e-02  6.486 8.84e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Prior Probabilities:
Comp.1: 0.6721
Comp.2: 0.3279


logLik: -7954, AIC: 15927, BIC: 15992.
```

Using the above code block, we make use of the function `clogit()` in the `survival` package and fit an FMM with two latent classes. We set `use.optim` to `TRUE` so that the `optim()`

function is used with the embedded c++ code. To generate a start value for the M-step, we produce a set of random values for the parameters for five times. For each set of random values, we run the maximum log-likelihood estimate for five steps. We pick the values with the largest log-likelihood as the starting value.

The results are well-separated given two classes though they are not exactly the true values.

6. Summary

The main challenge for implementing the EM algorithm and finite mixture modelling is that one could apply the mixture framework on any model. It would take immense effort and work if all models need to be built in the mixture form. In the *em* package, we make use of the feature of generic functions in R so that our `em` function can attach to other available statistical models in R. Our package can, therefore, focus on algorithm development, results presentation and post-estimations.

The current version of `em` supports most of generalized linear regressions implemented by function `lm` and `glm` in R. It also supports generalized non-linear regressions (`gnm`), multinomial regression models (`nnet::multinom`) and survival models (`survival::clogit`). Other models we are working on are mixture effect models (the `lme4` package) and mixture of distributions (the `fitdistrplus` package) and regressions with panel data (the `plm` package).

When fitting a finite mixture model, many factors (such as starting values, algorithms and models we mixed) can determine whether the maximum log-likelihood value will converge. The `em` package provides several extensions of EM algorithms and different methods to generate starting values for fitting. In addition, we provide a completed-data log-likelihood function for cases when a normal log-likelihood function is too flat. For estimating a completed-data log-likelihood function, we built a c++ module for speeding up the maximizing procedure so that a log-likelihood value is computed in c++ for each iteration. The c++ module massively reduces the computational time for the maximization part of the EM algorithm. These algorithms and methods are written independently in functions and objects so that more methods can be developed and included conveniently if required.

Acknowledgments

The research leading to the results presented in this article has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 851293). I am thankful to Dr. Kristian Bernt Karlson for his professional guidance and valuable advice.

References

Benaglia T, Chauveau D, Hunter DR, Young DS (2010). "mixtools: an R package for analyzing mixture models." *Journal of statistical software*, **32**, 1–29.

Deb P (2007). "FMM: Stata module to estimate finite mixture models." Statistical Software Components, Boston College Department of Economics. URL https://ideas.repec.org/c/boc/bocode/s456895.html.

Figueiredo M, Jain A (2002). "Unsupervised learning of finite mixture models." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(3), 381–396. doi:10.1109/34.990138.

Leisch F (2004). "Flexmix: A general framework for finite mixture models and latent glass regression in R."

McLachlan GJ, Lee SX, Rathnayake SI (2019). "Finite mixture models." *Annual review of statistics and its application*, **6**, 355–378.

Rabe-Hesketh S, Skrondal A, Pickles A (2004). "GLLAMM manual."

Scrucca L, Fop M, Murphy TB, Raftery AE (2016). "mclust 5: clustering, classification and density estimation using Gaussian finite mixture models." *The R journal*, **8**(1), 289.

Vermunt JK (2008). "Latent class and finite mixture models for multilevel data sets." *Statistical methods in medical research*, **17**(1), 33–51.

Vermunt JK, Magidson J (2005). "Hierarchical mixture models for nested data structures." In *Classification—The ubiquitous challenge*, pp. 240–247. Springer.

Wedel M (2002). "Concomitant variables in finite mixture models." *Statistica Neerlandica*, **56**(3), 362–375.

Affiliation:

Dongjie Wu
Sociologisk Institut
Københavns Universitet
Øster Farimagsgade 5
1353 Copenhagen
E-mail: dongjie.wu@soc.ku.dk