Journal of Statistical Software

October 2004, Volume 11, Issue 8. http://www.jstatsoft.org/

FlexMix: A General Framework for Finite Mixture Models and Latent Class Regression in R

Friedrich Leisch
Technische Universität Wien

Abstract

FlexMix implements a general framework for fitting discrete mixtures of regression models in the R statistical computing environment: three variants of the EM algorithm can be used for parameter estimation, regressors and responses may be multivariate with arbitrary dimension, data may be grouped, e.g., to account for multiple observations per individual, the usual formula interface of the S language is used for convenient model specification, and a modular concept of driver functions allows to interface many different types of regression models. Existing drivers implement mixtures of standard linear models, generalized linear models and model-based clustering. FlexMix provides the E-step and all data handling, while the M-step can be supplied by the user to easily define new models.

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

1. Introduction

Finite mixture models have been used for more than 100 years, but have seen a real boost in popularity over the last decade due to the tremendous increase in available computing power. The areas of application of mixture models range from biology and medicine to physics, economics and marketing. On the one hand these models can be applied to data where observations originate from various groups and the group affiliations are not known, and on the other hand to provide approximations for multi-modal distributions (Everitt and Hand 1981; Titterington, Smith, and Makov 1985; McLachlan and Peel 2000).

In the 1990s finite mixture models have been extended by mixing standard linear regression models as well as generalized linear models (Wedel and DeSarbo 1995). An important area of application of mixture models is market segmentation (Wedel and Kamakura 2001), where finite mixture models replace more traditional cluster analysis and cluster-wise regression techniques as state of the art. Finite mixture models with a fixed number of components are usually estimated with the expectation-maximization (EM) algorithm within a maximum

likelihood framework (Dempster, Laird, and Rubin 1977) and with MCMC sampling (Diebolt and Robert 1994) within a Bayesian framework.

The R environment for statistical computing (R Development Core Team 2004) features several packages for finite mixture models, including **mclust** for mixtures of multivariate Gaussian distributions (Fraley and Raftery 2002b,a), **fpc** for mixtures of linear regression models (Hennig 2000) and **mmlcr** for mixed-mode latent class regression (Buyske 2003).

There are three main reasons why we have chosen to write yet another software package for EM estimation of mixture models:

- The existing implementations did not cover all cases we needed for our own research (mainly marketing applications).
- While all R packages mentioned above are open source and hence can be extended by the user by modifying the source code, we wanted an implementation where extensibility is a main design principle to enable rapid prototyping of new mixture models.
- We include a sampling-based variant of the EM-algorithm for models where weighted maximum likelihood estimation is not available. FlexMix has a clean interface between E- and M-step such that variations of both are easy to combine.

This paper is organized as follows: First we introduce the mathematical models for latent class regression in Section 2 and shortly discuss parameter estimation and identifiability. Section 3 demonstrates how to use FlexMix to fit models with the standard driver for generalized linear models. Finally, Section 4 shows how to extend FlexMix by writing new drivers using the well-known model-based clustering procedure as an example.

All computations and graphics in this paper have been done with **flexmix** version 1.0-0 and R version 2.0.0 using Sweave (Leisch 2002). The newest release version of **flexmix** is always available from the Comprehensive R Archive Network at `http://cran.R-project.org`, an up-to-date version of this paper is contained as a package vignette.

2. Latent class regression

Consider finite mixture models with K components of form

$$h(y|x,\psi) = \sum_{k=1}^{K} \pi_k f(y|x,\theta_k) \tag{1}$$

$$\pi_k \geq 0, \quad \sum_{k=1}^{K} \pi_k = 1$$

where y is a (possibly multivariate) dependent variable with conditional density h, x is a vector of independent variables, π_k is the prior probability of component k, θ_k is the component specific parameter vector for the density function f, and $\psi = (\pi_1, , \ldots, \pi_K, \theta_1', \ldots, \theta_K')'$ is the vector of all parameters.

If f is a univariate normal density with component-specific mean $\beta_k' x$ and variance σ_k^2, we have $\theta_k = (\beta_k', \sigma_k^2)'$ and Equation (1) describes a mixture of standard linear regression models, also called *latent class regression* or *cluster-wise regression* (DeSarbo and Cron 1988). If f

is a member of the exponential family, we get a mixture of generalized linear models (Wedel and DeSarbo 1995), known as *GLIMMIX* models in the marketing literature (Wedel and Kamakura 2001). For multivariate normal f and $x \equiv 1$ we get a mixture of Gaussians without a regression part, also known as *model-based clustering*.

The posterior probability that observation (x, y) belongs to class j is given by

$$\mathsf{P}(j|x, y, \psi) = \frac{\pi_j f(y|x, \theta_j)}{\sum_k \pi_k f(y|x, \theta_k)} \tag{2}$$

The posterior probabilities can be used to segment data by assigning each observation to the class with maximum posterior probability. In the following we will refer to $f(\cdot|\cdot, \theta_k)$ as *mixture components* or *classes*, and the groups in the data induced by these components as *clusters*.

2.1. Parameter estimation

The log-likelihood of a sample of N observations $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ is given by

$$\log L = \sum_{n=1}^{N} \log h(y_n|x_n, \psi) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k f(y_n|x_n, \theta_k) \right) \tag{3}$$

and can usually not be maximized directly. The most popular method for maximum likelihood estimation of the parameter vector ψ is the iterative EM algorithm (Dempster *et al.* 1977):

Estimate the posterior class probabilities for each observation

$$\hat{p}_{nk} = \mathsf{P}(k|x_n, y_n, \hat{\psi})$$

using Equation (2) and derive the prior class probabilities as

$$\hat{\pi}_k = \frac{1}{N} \sum_{n=1}^{N} \hat{p}_{nk}$$

Maximize the log-likelihood for each component separately using the posterior probabilities as weights

$$\max_{\theta_k} \sum_{n=1}^{N} \hat{p}_{nk} \log f(y_n|x_n, \theta_k) \tag{4}$$

The E- and M-steps are repeated until the likelihood improvement falls under a pre-specified threshold or a maximum number of iterations is reached.

The EM algorithm cannot be used for mixture models only, but rather provides a general framework for fitting models on incomplete data. Suppose we augment each observation (x_n, y_n) with an unobserved multinomial variable $z_n = (z_{n1}, \ldots, z_{nK})$, where $z_{nk} = 1$ if (x_n, y_n) belongs to class k and $z_{nk} = 0$ otherwise. The EM algorithm can be shown to maximize the likelihood on the "complete data" (x_n, y_n, z_n); the z_n encode the missing class information. If the z_n were known, maximum likelihood estimation of all parameters would be easy, as we could separate the data set into the K classes and estimate the parameters θ_k for each class independently from the other classes.

If the weighted likelihood estimation in Equation (4) is infeasible for analytical, computational, or other reasons, then we have to resort to approximations of the true EM procedure by assigning the observations to disjoint classes and do unweighted estimation within the groups:

$$\max_{\theta_k} \sum_{n:z_{nk}=1} \log f(y_n|x_n, \theta_k)$$

This corresponds to allow only 0 and 1 as weights.

Possible ways of assigning the data into the K classes are

- **hard** assignment to the class with maximum posterior probability p_{nk}, the resulting procedure is called maximizing the *classification likelihood* by Fraley and Raftery (2002b). Another idea is to do
- **random** assignment to classes with probabilities p_{nk}, which is similar to the sampling techniques used in Bayesian estimation (although for the z_n only).

Well known limitations of the EM algorithm include that convergence can be slow and is to a local maximum of the likelihood surface only. There can also be numerical instabilities at the margin of parameter space, and if a component gets to contain only a few observations during the iterations, parameter estimation in the respective component may be problematic. E.g., the likelihood of Gaussians increases without bounds for $\sigma^2 \to 0$. As a result, numerous variations of the basic EM algorithm described above exist, most of them exploiting features of special cases for f.

2.2. Identifiability

An open question is still identifiability of many mixture models. A comprehensive overview of this topic is beyond the scope of this paper, however, users of mixture models should be aware of the problem:

Relabelling of components: Mixture models are only identifiable up to a permutation of the component labels. For EM-based approaches this only affects interpretation of results, but is no problem for parameter estimation itself.

Overfitting: If a component is empty or two or more components have the same parameters, the data generating process can be represented by a smaller model with fewer components. This kind of unidentifiability can be avoided by requiring that the prior weights π_k are not equal to zero and that the component specific parameters are different.

Generic unidentifiability: It has been shown that mixtures of univariate normal, gamma, exponential, Cauchy and Poisson distributions are identifiable, while mixtures of discrete or continuous uniform distributions are not identifiable. A special case is the class of mixtures of binomial and multinomial distributions which are only identifiable if the number of components is limited with respect to, e.g., the number of observations per person. See Everitt and Hand (1981), Titterington *et al.* (1985), Grün (2002) and references therein for details.

FlexMix tries to avoid overfitting because of vanishing prior probabilities by automatically removing components where the prior π_k falls below a user-specified threshold. Automated

diagnostics for generic identifiability are currently under investigation. Relabelling of components is in some cases more of a nuisance than a real problem ("component 2 of the first run may be component 3 in the second run"), more serious are interactions of component relabelling and categorical predictor variables, see Grün and Leisch (2004) for a discussion and how bootstrapping can be used to assess identifiability of mixture models.

3. Using FlexMix

The standard M-step `FLXglm()` of FlexMix is an interface to R's generalized linear modelling facilities (the `glm()` function). As a simple example we use artificial data with two latent classes of size 100 each:

$$\begin{aligned} \text{Class 1:} \quad & y = 5x + \epsilon \\ \text{Class 2:} \quad & y = 15 + 10x - x^2 + \epsilon \end{aligned}$$

with $\epsilon \sim N(0, 9)$ and prior class probabilities $\pi_1 = \pi_2 = 0.5$, see the left panel of Figure 1. We can fit this model in R using the commands

```
> library(flexmix)
> data(NPreg)
> m1 = flexmix(yn ~ x + I(x^2), data = NPreg, k = 2)
> m1

Call:
flexmix(formula = yn ~ x + I(x^2), data = NPreg,
    k = 2)

Cluster sizes:
  1   2
100 100

convergence after 15 iterations
```

and get a first look at the estimated parameters of mixture component 1 by

```
> parameters(m1, component = 1)

$coef
(Intercept)           x      I(x^2)
-0.20989331  4.81782414  0.03615728

$sigma
[1] 3.476362
```

and

```
> parameters(m1, component = 2)
```

```
$coef
(Intercept)           x      I(x^2)
 14.7168295   9.8466698  -0.9683534

$sigma
[1] 3.479809
```

for component 2. The paramter estimates of both components are close to the true values. A cross-tabulation of true classes and cluster memberships can be obtained by

```
> table(NPreg$class, m1@cluster)

    1  2
  1 95  5
  2  5 95
```

The summary method

```
> summary(m1)

Call:
flexmix(formula = yn ~ x + I(x^2), data = NPreg,
    k = 2)

       prior size post>0 ratio
Comp.1 0.494  100    145 0.690
Comp.2 0.506  100    141 0.709

`log Lik.' -642.5453 (df=9)
AIC: 1303.091   BIC: 1332.775
```

gives the estimated prior probabilities $\hat{\pi}_k$, the number of observations assigned to the corresponding clusters, the number of observations where $p_{nk} > \delta$ (with a default of $\delta = 10^{-4}$), and the ratio of the latter two numbers. For well-seperated components, a large proportion of observations with non-vanishing posteriors p_{nk} should also be assigned to the corresponding cluster, giving a ratio close to 1. For our example data the ratios of both components are approximately 0.7, indicating the overlap of the classes at the cross-section of line and parabola.

Histograms or rootograms of the posterior class probabilities can be used to visually assess the cluster structure (Tantrum, Murua, and Stuetzle 2003), this is now the default plot method for `"flexmix"` objects (Leisch 2004). Rootograms are very similar to histograms, the only difference is that the height of the bars correspond to square roots of counts rather than the counts themselves, hence low counts are more visible and peaks less emphasized.

Usually in each component a lot of observations have posteriors close to zero, resulting in a high count for the corresponing bin in the rootogram which obscures the information in the other bins. To avoid this problem, all probabilities with a posterior below a threshold are

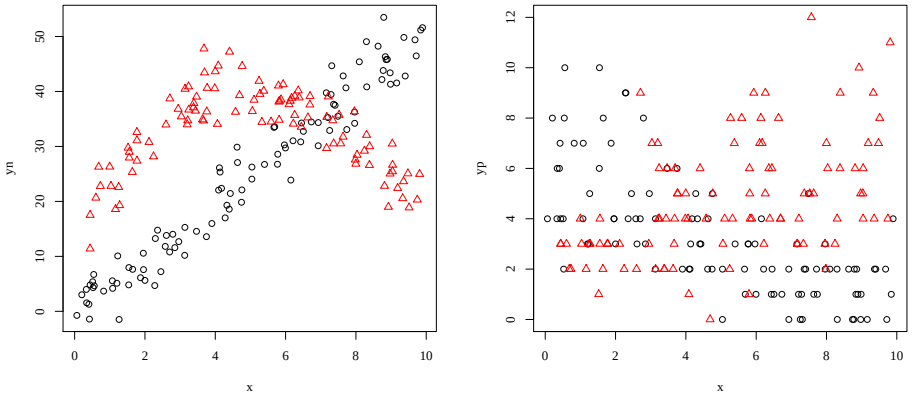

Figure 1: Standard regression example (left) and Poisson regression (right).

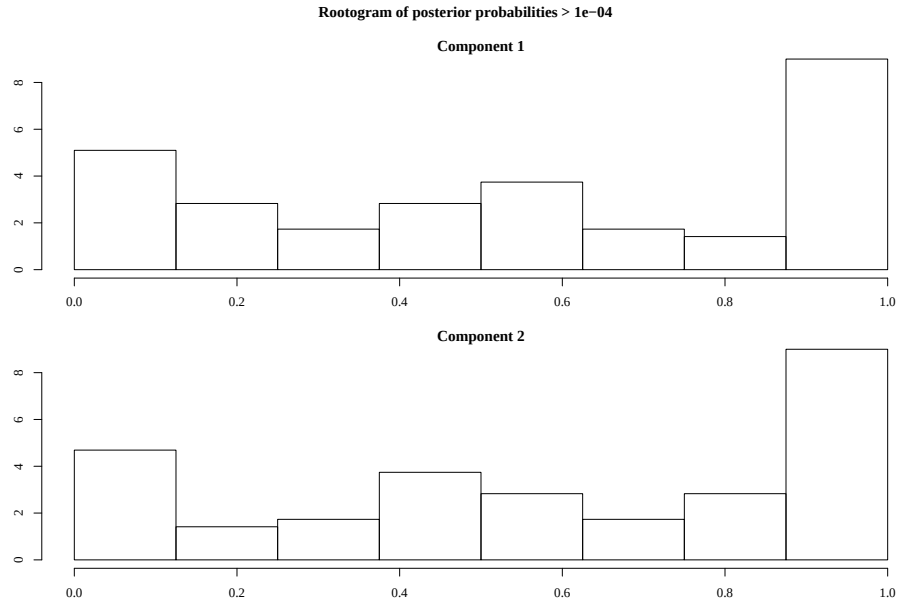

Figure 2: The plot method for `"flexmix"` objects, here obtained by `plot(m1)`, shows rootograms of the posterior class probabilities.

ignored (we again use 10^{-4}). A peak at probability 1 indicates that a mixture component is well seperated from the other components, while no peak at 1 and/or significant mass in the middle of the unit interval indicates overlap with other components. In our simple example the components are medium well separated, see Figure 2.

Tests for significance of regression coefficients can be obtained by

```
> rm1 = refit(m1)
> summary(rm1)

Call:
refit(m1)

Component 1 :
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.208996   0.673900 -0.3101   0.7568
x            4.817015   0.327447 14.7108   <2e-16
I(x^2)       0.036233   0.032545  1.1133   0.2669

-------------
Component 2 :
             Estimate Std. Error t value  Pr(>|t|)
(Intercept) 14.717541   0.890843  16.521 < 2.2e-16
x            9.846148   0.390385  25.222 < 2.2e-16
I(x^2)      -0.968304   0.036951 -26.205 < 2.2e-16
```

Function `refit()` fits weighted generalized linear models to each component using the standard R function `glm()` and the posterior probabilities as weights, see `help("refit")` for details.

The data set `NPreg` also includes a response from a generalized linear model with a Poisson distribution and exponential link function. The two classes of size 100 each have parameters

$$\begin{aligned} \text{Class 1:} \quad & \mu_1 = 2 - 0.2x \\ \text{Class 2:} \quad & \mu_2 = 1 + 0.1x \end{aligned}$$

and given x the response y in group k has a Poisson distribution with mean e^{μ_k}, see the right panel of Figure 1. The model can be estimated using

```
> m2 = flexmix(yp ~ x, data = NPreg, k = 2, model = FLXglm(family = "poisson"))
> summary(m2)

Call:
flexmix(formula = yp ~ x, data = NPreg, k = 2, model = FLXglm(family = "poisson"))

       prior size post>0 ratio
Comp.1 0.532  112    197 0.569
Comp.2 0.468   88    200 0.440

`log Lik.' -440.6425 (df=5)
AIC: 891.285   BIC: 907.7767
```

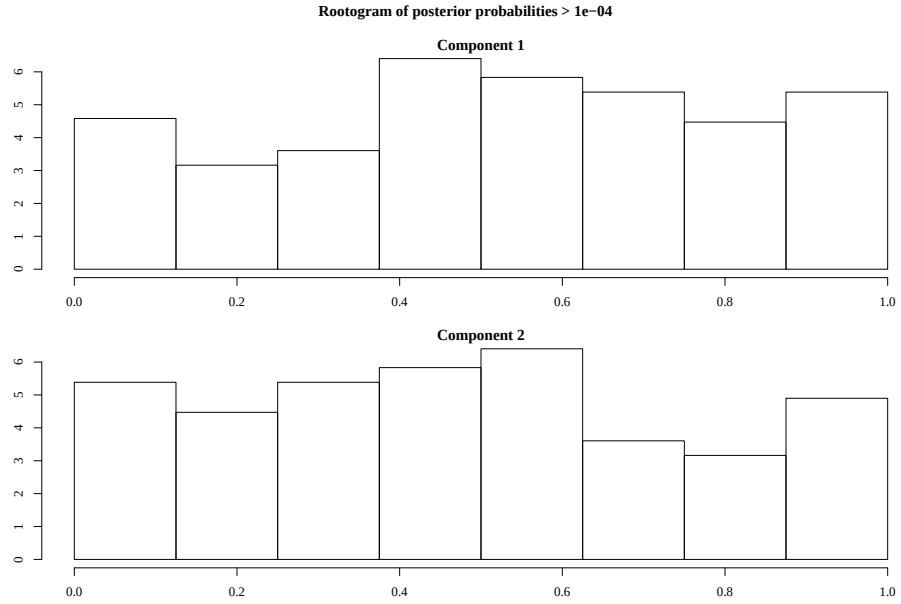

Figure 3: `plot(m2)`

Both the summary table and the rootograms in Figure 3 clearly show that the clusters of the Poisson response have much more overlap. For our simple low-dimensional example data the overlap of the classes is obvious by looking at scatterplots of the data. For data in higher dimensions this is not an option. The rootograms and summary tables for `"flexmix"` objects work off the densities or posterior probabilities of the observations and thus do not depend on the dimensionality of the input space. While we use simple 2-dimensional examples to demonstrate the techniques, they can easily be used on high-dimensional data sets or models with complicated covariate structures.

3.1. Multiple independent responses

If the response $y = (y_1, \ldots, y_D)'$ is D-dimensional and the y_d are mutually independent the mixture density in Equation (1) can be written as

$$\begin{aligned} h(y|x, \psi) &= \sum_{k=1}^{K} \pi_k f(y|x, \theta_k) \\ &= \sum_{k=1}^{K} \pi_k \prod_{d=1}^{D} f_d(y|x, \theta_{kd}) \end{aligned}$$

To specify such models in FlexMix we pass it a list of models, where each list element corresponds to one f_d, and each can have a different set of dependent and independent variables. To use the Gaussian and Poisson responses of data `NPreg` simultaneously, we use the model specification

```
> m3 = flexmix(~x, data=NPreg, k=2,
+              model=list(FLXglm(yn~.+I(x^2)),
+                         FLXglm(yp~., family="poisson")))
```

Note that now three model formulas are involved: An overall formula as first argument to function `flexmix()` and one formula per response. The latter ones are interpreted relative to the overall formula such that common predictors have to be specified only once, see `help("update.formula")` for details on the syntax. The basic principle is that the dots get replaced by the respective terms from the overall formula. The rootograms show that the posteriors of the two-response model are shifted towards 0 and 1 (compared with either of the two univariate models), the clusters are now well-separated.

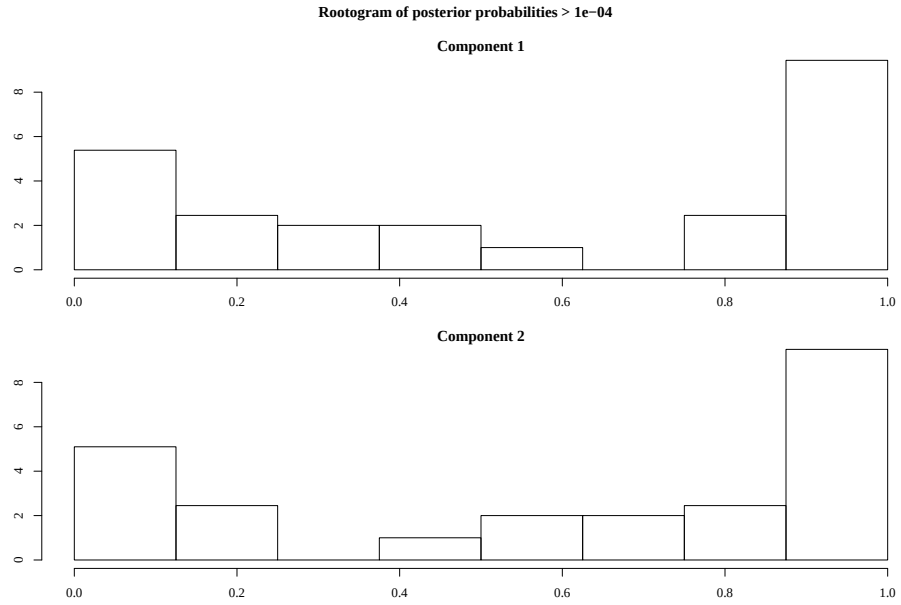

Figure 4: `plot(m3)`

3.2. Repeated measurements

If the data are repeated measurements on M individuals, and we have N_m observations from individual m, then the log-likelihood in Equation (3) can be written as

$$\log L = \sum_{m=1}^{M} \sum_{n=1}^{N_m} \log h(y_{mn}|x_{mn}, \psi), \qquad \sum_{m=1}^{M} N_m = N$$

and the posterior probability that individual m belongs to class j is given by

$$\mathsf{P}(j|m) = \frac{\pi_j \prod_{n=1}^{N_m} f(y_{mn}|x_{mn}, \theta_j)}{\sum_k \pi_k \prod_{n=1}^{N_m} f(y_{mn}|x_{mn}, \theta_k)}$$

where (x_{mn}, y_{mn}) is the n-th observation from individual m. As an example, assume that the data in `NPreg` are not 200 independent observations, but 4 measurements each from 50 persons such that $\forall m : N_m = 4$. Column `id2` of the data frame encodes such a grouping and can easily be used in FlexMix:

```
> m4 = flexmix(yn ~ x + I(x^2) | id2, data = NPreg,
+     k = 2)
> summary(m4)
```

```
Call:
flexmix(formula = yn ~ x + I(x^2) | id2, data = NPreg,
    k = 2)

       prior size post>0 ratio
Comp.1   0.5  100    100     1
Comp.2   0.5  100    100     1

`log Lik.' -2246.26 (df=9)
AIC: 4510.52   BIC: 4540.205
```

Note that convergence of the EM algorithm is much faster with grouping and the two clusters are now perfectly separated.

3.3. Control of the EM algorithm

Details of the EM algorithm can be tuned using the `control` argument of function `flexmix()`. E.g., to use a maximum number of 15 iterations, report the log-likelihood at every 3rd step and use hard assignment of observations to clusters (cf. page 4) the call is

```
> m5 = flexmix(yn ~ x + I(x^2), data = NPreg, k = 2,
+     control = list(iter.max = 15, verbose = 3, classify = "hard"))

Classification: hard
   3 Log-likelihood :    -718.2839
   6 Log-likelihood :    -712.2280
   9 Log-likelihood :    -711.1906
  12 Log-likelihood :    -711.0684
  13 Log-likelihood :    -711.0680
converged
```

Another control parameter (`minprior`, see below for an example) is the minimum prior probability components are enforced to have, components falling below this threshold (the current default is 0.05) are removed during EM iteration to avoid numerical instabilities for components containing only a few observations. Using a minimum prior of 0 disables component removal.

3.4. Automated model search

In real applications the number of components is unknown and has to be estimated. Tuning the minimum prior parameter allows for simplistic model selection, which works surprisingly well in some situations:

```
> m6 = flexmix(yp ~ x + I(x^2), data = NPreg, k = 4,
+     control = list(minprior = 0.2))
> m6
```

```
Call:
flexmix(formula = yp ~ x + I(x^2), data = NPreg,
    k = 4, control = list(minprior = 0.2))

Cluster sizes:
  1   2
 79 121

convergence after 142 iterations
```

Although we started with four components, the algorithm converged at the correct two component solution.

A better approach is to fit models with an increasing number of components and compare them using AIC or BIC. As the EM algorithm converges only to the next local maximum of the likelihood, it should be run repeatedly using different starting values. The function `stepFlexmix()` can be used to repeatedly fit models, e.g.,

```
> m7 = stepFlexmix(yp ~ x + I(x^2), data = NPreg, control = list(verbose = 0),
+     K = 1:5, nrep = 5)

1 : * * * * *
2 : * * * * *
3 : * * * * *
4 : * * * * *
5 : * * * * *
```

runs `flexmix()` 5 times for $K = 1, 2, \ldots, 5$ components, totalling in 25 runs. It returns a list with the best solution found for each number of components, each list element is simply an object of class `"flexmix"`. To find the best model we can use

```
> sapply(m7, BIC)

       1        2        3        4        5
946.7477 925.9972 942.1553 960.0626 960.9347
```

and choose the number of components minimizing the BIC.

4. Extending FlexMix

One of the main design principles of FlexMix was extensibility, users can provide their own M-step for rapid prototyping of new mixture models. FlexMix was written using S4 classes and methods (Chambers 1998) as implemented in R package **methods**.

The central classes for writing M-steps are `"FLXmodel"` and `"FLXcomponent"`. Class `"FLXmodel"` specifies how the model is fitted using the following slots:

fit: A `function(x,y,w)` returning an object of class `"FLXcomponent"`.

weighted: Logical, specifies if the model may be fitted using weighted likelihoods. If `FALSE`, only hard and random classification are allowed (and hard classification becomes the default).

formula: Formula relative to the overall model formula, default is `.~.`.

name: A character string describing the model, this is only used for print output.

The remaining slots of class `"FLXmodel"` are used internally by FlexMix to hold data, etc. and omitted here, because they are not needed to write an M-step driver. The most important slot doing all the work is `fit` holding a function performing the maximum likelihood estimation described in Equation (4). The `fit()` function returns an object of class `"FLXcomponent"` which holds a fitted component using the slots:

logLik: A `function(x,y)` returning the log-likelihood for observations in matrices `x` and `y`.

predict: A `function(x)` predicting `y` given `x`.

df: The degrees of freedom used by the component, i.e., the number of estimated parameters.

parameters: An optional list containing model parameters.

In a nutshell class `"FLXmodel"` describes an *unfitted* model, whereas class `"FLXcomponent"` holds a *fitted* model.

4.1. Writing an M-step driver

Figure 5 shows an example driver for model-based clustering. We use function `dmvnorm()` from package **mvtnorm** for calculation of multivariate Gaussian densities. In line 5 we create a new `"FLXmodel"` object named `retval`, which is also the return value of the driver. All drivers should take a formula as their first argument, this formula is directly passed on to `retval`. In most cases authors of new FlexMix drivers need not worry about formula parsing etc., this is done by `flexmix` itself. In addition we have to declare whether the driver can do weighted ML estimation (`weighted=TRUE`) and give a name to our model.

The remainder of the driver creates a `fit()` function, which takes regressors `x`, response `y` and weights `w`. For multivariate Gaussians the maximum likelihood estimates correspond to mean and covariance matrix, the standard R function `cov.wt()` returns a list containing estimates of the weighted covariance matrix and the mean for given data. Our simple example performs clustering without a regression part, hence x is ignored. If `y` has D columns, we estimate D parameters for the mean and $D(D-1)/2$ parameters for the covariance matrix, giving a total of $(3D + D^2)/2$ parameters (line 11). As an additional feature we allow the user to specify whether the covariance matrix is assumed to be diagonal or a full matrix. For `diagonal=TRUE` we use only the main diagonal of the covariance matrix (line 14) and the number of parameters is reduced to $2D$.

In addition to parameter estimates, `flexmix()` needs a function calculating the log-likelihood of given data x and y, which in our example is the log-density of a multivariate Gaussian. In addition we have to provide a function predicting y given x, in our example simply the mean of the Gaussian. Finally we create a new `"FLXcomponent"` as return value of function `fit()`.

```
mymclust <- function (formula = .~., diagonal = TRUE)
{
    require("mvtnorm")

    retval <- new("FLXmodel", weighted = TRUE, formula = formula,
                  name = "my model-based clustering")

    retval@fit <- function(x, y, w) {

        para <- cov.wt(y, wt = w)[c("center", "cov")]
        df <- (3 * ncol(y) + ncol(y)^2)/2

        if (diagonal) {
            para$cov <- diag(diag(para$cov))
            df <- 2 * ncol(y)
        }

        logLik <- function(x, y) {
            dmvnorm(y, mean = para$center, sigma = para$cov,
                log = TRUE)
        }

        predict <- function(x) {
            matrix(para$center, nrow = nrow(y),
                   ncol = length(para$center), byrow = TRUE)
        }

        new("FLXcomponent", parameters = para, df = df,
            logLik = logLik, predict = predict)
    }
    retval
}
```

Figure 5: M-step for model-based clustering: `mymclust` is a simplified version of the standard FlexMix driver `FLXmclust`.

Note that our internal functions `fit()`, `logLik()` and `predict()` take only `x`, `y` and `w` as arguments, but none of the model-specific parameters like means and covariances, although they use them of course. R uses *lexical scoping* rules for finding free variables (Gentleman and Ihaka 2000), hence it searches for them first in the environment where a function is defined. E.g., the `fit()` function uses the variable `diagonal` in line 13, and finds it in the environment where the function itself was defined, which is the body of function `mymclust()`. Function `logLik()` uses the list `para` in line 19, and uses the one found in the body of `fit()`.

Function `flexmix()` on the other hand never sees the model parameters, all it uses are function calls of form `fit(x,y,w)` or `logLik(x,y)`, which are exactly the same for all kinds of mixture models. In fact, it would not be necessary to even store the component parameters in the `"FLXcomponent"` object, they are there only for convenience such that users can easily extract and use them after `flexmix()` has finished. Lexical scope allows to write clean interfaces in a very elegant way, the driver abstracts all model details from the FlexMix main engine.

4.2. Example: Using the driver

As a simple example we use the four 2-dimensional Gaussian clusters from data set `Nclus`. Fitting a wrong model with diagonal covariance matrix is done by

```
> data(Nclus)
> m1 = flexmix(Nclus ~ 1, k = 4, model = mymclust())
> summary(m1)

Call:
flexmix(formula = Nclus ~ 1, k = 4, model = mymclust())

       prior size post>0 ratio
Comp.1 0.159   92    165 0.558
Comp.2 0.269  149    174 0.856
Comp.3 0.397  213    488 0.436
Comp.4 0.175   96    172 0.558

`log Lik.' -2447.3 (df=19)
AIC: 4932.6   BIC: 5014.489
```

The result can be seen in the left panel of Figure 6, the result is "wrong" because we forced the ellipses to be parallel to the axes. The overlap between three of the four clusters can also be inferred from the low ratio statistics in the summary table (around 0.5 for components 1, 3 and 4), while the much better separated upper left cluster has a much higher ratio of 0.85. Using the correct model with a full covariance matrix can be done by setting `diagonal=FALSE` in the call to our driver `mymclust()`:

```
> m2 = flexmix(Nclus ~ 1, k = 4, model = mymclust(diagonal = FALSE))
> summary(m2)

Call:
flexmix(formula = Nclus ~ 1, k = 4, model = mymclust(diagonal = FALSE))

       prior size post>0 ratio
Comp.1 0.177   96    132 0.727
Comp.2 0.368  204    247 0.826
Comp.3 0.272  150    176 0.852
Comp.4 0.182  100    112 0.893

`log Lik.' -2223.677 (df=23)
AIC: 4493.355   BIC: 4592.483
```

5. Summary and outlook

The primary goal of FlexMix is extensibility, this makes the package ideal for rapid development of new mixture models. There is no intent to replace packages implementing more

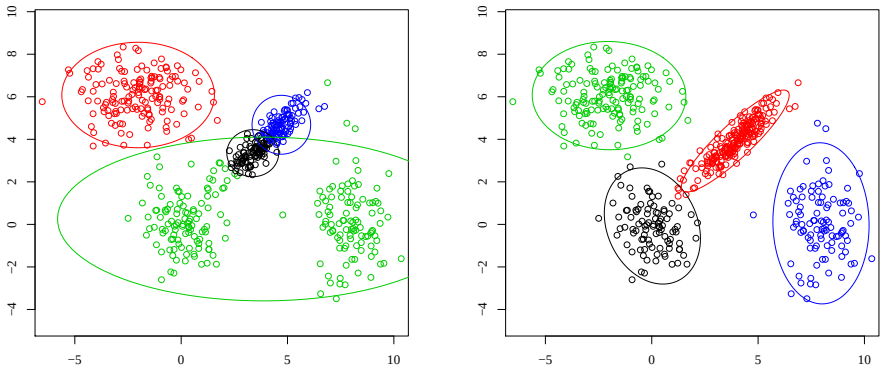

Figure 6: Fitting a mixture model with diagonal covariance matrix (left) and full covariance matrix (right).

specialized mixture models like **mclust** for mixtures of Gaussians, FlexMix should rather be seen as a complement to those. By interfacing R's facilities for generalized linear models, FlexMix allows the user to estimate complex latent class regression models.

Using lexical scope to resolve model-specific parameters hides all model details from the programming interface, FlexMix can in principle fit almost arbitrary finite mixture models for which the EM algorithm is applicable. The downside of this is that FlexMix can in principle fit almost arbitrary finite mixture models, even models where no proper theoretical results for model identification etc. are available.

We are currently working on a toolset for diagnostic checks on mixture models to test necessary identifiability conditions for those cases where results are available. We also want to implement newer variations of the classic EM algorithm, especially for faster convergence. Another plan is to have an interactive version of the rootograms using `iPlots` (Urbanek and Theus 2003) such that the user can explore the relations between mixture components, possibly linked to background variables. Other planned extensions include covariates for the prior probabilities and to allow to mix different distributions for components, e.g., to include a Poisson point process for background noise.

Acknowledgments

This research was supported by the Austrian Science Foundation (FWF) under grant SFB#010 ('Adaptive Information Systems and Modeling in Economics and Management Science').

References

Buyske S (2003). *R Package* `mmlcr`*: Mixed-Mode Latent Class Regression*. Version 1.3.2, URL http://www.stat.rutgers.edu/~buyske/software.html.

Chambers JM (1998). *Programming with data: A guide to the S language*. Springer Verlag, Berlin, Germany.

Dempster A, Laird N, Rubin D (1977). "Maximum Likelihood from Incomplete Data via the EM-Alogrithm." *Journal of the Royal Statistical Society, B*, **39**, 1–38.

DeSarbo WS, Cron WL (1988). "A Maximum Likelihood Methodology for Clusterwise Linear Regression." *Journal of Classification*, **5**, 249–282.

Diebolt J, Robert CP (1994). "Estimation of finite mixture distributions through Bayesian sampling." *Journal of the Royal Statistical Society, Series B*, **56**, 363–375.

Everitt BS, Hand DJ (1981). *Finite Mixture Distributions*. London: Chapman and Hall.

Fraley C, Raftery AE (2002a). "MCLUST: Software for Model-Based Clustering, Discriminant Analysis and Density Estimation." *Technical Report 415*, Department of Statistics, University of Washington, Seattle, WA, USA. URL http://www.stat.washington.edu/raftery.

Fraley C, Raftery AE (2002b). "Model-based clustering, discriminant analysis and density estimation." *Journal of the American Statistical Association*, **97**, 611–631.

Gentleman R, Ihaka R (2000). "Lexical scope and statistical computing." *Journal of Computational and Graphical Statistics*, **9**(3), 491–508.

Grün B (2002). *Identifizierbarkeit von multinomialen Mischmodellen*. Master's thesis, Technische Universität Wien. Kurt Hornik and Friedrich Leisch, advisors.

Grün B, Leisch F (2004). "Bootstrapping Finite Mixture Models." In J Antoch (ed.), "Compstat 2004 — Proceedings in Computational Statistics," pp. 1115–1122. Physika Verlag, Heidelberg, Germany. ISBN 3-7908-1554-3.

Hennig C (2000). "Identifiability of Models for Clusterwise Linear Regression." *Journal of Classification*, **17**, 273–296.

Leisch F (2002). "Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), "Compstat 2002 — Proceedings in Computational Statistics," pp. 575–580. Physika Verlag, Heidelberg, Germany. ISBN 3-7908-1517-9, URL http://www.ci.tuwien.ac.at/~leisch/Sweave.

Leisch F (2004). "Exploring the Structure of Mixture Model Components." In J Antoch (ed.), "Compstat 2004 — Proceedings in Computational Statistics," pp. 1405–1412. Physika Verlag, Heidelberg, Germany. ISBN 3-7908-1554-3.

McLachlan G, Peel D (2000). *Finite Mixture Models*. John Wiley and Sons Inc.

R Development Core Team (2004). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Tantrum J, Murua A, Stuetzle W (2003). "Assessment and Pruning of Hierarchical Model Based Clustering." In "Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining," pp. 197–205. ACM Press, New York, NY, USA. ISBN:1-58113-737-0.

Titterington D, Smith A, Makov U (1985). *Statistical Analysis of Finite Mixture Distributions*. Chichester: Wiley.

Urbanek S, Theus M (2003). "iPlots — High Interaction Graphics for R." In K Hornik, F Leisch, A Zeileis (eds.), "Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria," ISSN 1609-395X, URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/.

Wedel M, DeSarbo WS (1995). "A mixture likelihood approach for generalized linear models." *Journal of Classification*, **12**, 21–55.

Wedel M, Kamakura WA (2001). *Market Segmentation - Conceptual and Methodological Foundations*. Kluwer Academic Publishers, Boston, MA, USA, 2nd edition.

Affiliation:

Friedrich Leisch
Institut für Statistik & Wahrscheinlichkeitstheorie
Technische Universität Wien
Wiedner Hauptstraße 8-10/1071
1040 Wien, Austria
E-mail: Friedrich.Leisch@tuwien.ac.at
URL: http://www.ci.tuwien.ac.at/~leisch/

Journal of Statistical Software
October 2004, Volume 11, Issue 8.
http://www.jstatsoft.org/

Submitted: 2004-04-19
Accepted: 2004-10-18